$$D = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & .9 & .9 & .1 \\ .7 & 1.2 & 1.2 & .7 \\ .1 & .9 & .9 & 1 \end{pmatrix}$$

$$\approx \begin{pmatrix} 1 & 0 \\ .6 & .6 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & .9 & .9 & .1 \\ .1 & .9 & .9 & 1 \end{pmatrix}$$

$$D_A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} = \theta\begin{pmatrix} 1 & 0 \\ .6 & .6 \\ 0 & 1 \end{pmatrix} \cdot \theta\begin{pmatrix} 1 & .9 & .9 & .1 \\ .1 & .9 & .9 & 1 \end{pmatrix}$$

$$D_B = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} = \theta\left(\theta\begin{pmatrix} 1 & 0 \\ .6 & .6 \\ 0 & 1 \end{pmatrix} \cdot \theta\begin{pmatrix} 1 & .9 & .9 & .1 \\ .1 & .9 & .9 & 1 \end{pmatrix}\right)$$

Fig. 5.15: Approximation of a binary matrix D with two overlapping biclusters (top) applying NMF (second from above) and the factorizations resulting from thresholding the factor matrices to binary matrices in elementary algebra (second from below) and Boolean algebra (below). Biclusters are highlighted.

1. *Optimize the following objective with proximal gradient descent:*

$$\begin{aligned} &\min_{X,Y} \|D - YX^\top\|^2 + \lambda_x \langle \boldsymbol{\Lambda}(X), \mathbf{1}\rangle + \lambda_y \langle \boldsymbol{\Lambda}(Y), \mathbf{1}\rangle \\ &s.t.\ X \in \mathbb{R}^{d\times r},\ Y \in \mathbb{R}^{N\times r}. \end{aligned} \tag{5.34}$$

   That is, perform the PALM update steps from Equations 5.29 and 5.30 for $F(X, Y) = \|D - YX^\top\|^2$ *and the regularizing functions*

$$\phi_x(X) = \lambda_x \langle \boldsymbol{\Lambda}(X), \mathbf{1}\rangle, \qquad \phi_y(Y) = \lambda_y \langle \boldsymbol{\Lambda}(Y), \mathbf{1}\rangle.$$

2. *Return the binary matrices that result from a suitable thresholding operation of the relaxed result. That is, perform a grid search on the set* $\mathcal{T} = \{0, 0.05, 0.1, \dots, 1\}$*:*

$$\begin{aligned} (\rho_x^\star, \rho_y^\star) &= \arg\min_{\rho_x,\rho_y} \{\|D - \theta_{\rho_y}(Y_t) \odot \theta_{\rho_x}(X_t^\top)\|^2 \mid \rho_x, \rho_y \in \mathcal{T}\} \\ (X, Y) &= (\theta_{\rho_x^\star}(X_t), \theta_{\rho_y^\star}(Y_t)) \end{aligned}$$

The matrix or vector $\mathbf{1}$ indicates a constant one matrix, whose dimensionality can be inferred from context. The Frobenius inner product $\langle \Lambda(X), \mathbf{1}\rangle = \sum_{i,s} \Lambda(X_{is})$ sums the penalization terms over all matrix entries. As a default value, we employ a natural choice for the regularization weight $\lambda_x = \lambda_y = 1$.

The procedure of PAL-Tiling describes the general framework for the optimization of the Boolean matrix factorization error. Recent contributions in the field of Boolean matrix factorization also incorporate a mechanism to automatically determine the rank of the factorization. To this end, objective functions other than the approximation error in Frobenius norm are commonly optimized in BMF. So far, we have proposed two approaches to facilitate the optimization with PAL-Tiling. The rank determination can be integrated into an outer loop of PAL-Tiling, where the models of various ranks are compared. One approach uses the well-established Minimum Description Length (MDL) principle to determine the rank as the one yielding the minimum description length of the model. Here, the objective in Boolean algebra is the employed description length [310]. The other approach is to optimize the Boolean approximation error, and to conduct statistical tests on the probability that at least one of the clusters is generated by noise [311].

5.4.6 BROCCOLI – Optimizing Tri-Factorization Biclusterings

While the final thresholding step for the optimization of Boolean factorizations is needed to translate the fuzzy clustering structure to the Boolean algebra, the biclustering models of checkerboard and block-diagonal clustering use the elementary algebra matrix product. As a result, the relaxed formulation of the objective with the nonbinary penalization terms in Equation 5.34 will have the same optimizers as the corresponding binary matrix factorization objective (cf. Table 5.9), if the penalizing weights λ_x and λ_y are large enough.

After every gradient descent step of one of the cluster indicator matrices, the prox-operator is applied and pushes the matrix towards binary values. Hence, if the nonbinary penalization weights λ_x and λ_y are large enough, then we will get binary matrices after a couple of iterations. However, in this case, we also risk converging to a local optimum close to the initialization. This would make our method even more sensitive to the initialization than it already is due to the nonconvexity of the objective. In turn, if we choose a value for λ that is too small, then the optimum of the penalized objective might not return binary matrices. We circumvent these issues by gradually increasing the regularization weights throughout the optimization process. In addition, we employ individual regularization weights. To this end, we introduce the regularization weights as optimization parameters that are as large as possible in the optimal solution. We achieve this by subtracting the sum of the regularization parameters $\langle \boldsymbol{\lambda}_x, \mathbf{1}\rangle + \langle \boldsymbol{\lambda}_y, \mathbf{1}\rangle$ from the objective function value (cf. Equation 5.35).

Algorithm Specification 2 (Broccoli). *Given a data matrix $D \in \mathbb{R}^{N\times d}$ and a biclustering optimization problem, such as*

$$\min_{X,Y,C} \|D - YCX^\top\|^2 + \phi_c(C) \qquad \text{s.t. } Y \in \{0,1\}^{N\times r}, X \in \{0,1\}^{d\times r}, C \in \mathbb{R}^{r\times r}.$$

1. *Optimize the following objective with proximal gradient descent:*

$$\begin{aligned}\min_{\substack{X,Y,C,\\ \boldsymbol{\lambda}_x,\boldsymbol{\lambda}_y}} \|D - YCX^\top\|^2 + \langle \boldsymbol{\lambda}_x, \Lambda(X) - \mathbf{1}\rangle + \langle \boldsymbol{\lambda}_y, \Lambda(Y) - \mathbf{1}\rangle + \phi_c(C)\\ s.t.\ (\boldsymbol{\lambda}_x)_{is}, (\boldsymbol{\lambda}_y)_{js} \le \lambda_+ \textit{ for all } 1 \le i \le d,\ 1 \le j \le N,\ 1 \le s \le r.\\ Y \in \mathbb{R}^{N\times r}, X \in \mathbb{R}^{d\times r}, C \in \mathbb{R}^{r\times r}\end{aligned} \tag{5.35}$$

That is, perform the PALM update steps in an alternating fashion for X,C, λ_x, Y, C, and at last for λ_y. Where $F(X,Y,C) = \|D - YCX^\top\|^2$, and the regularizing functions

$$\phi_x(X) = \langle \boldsymbol{\lambda}_x, \Lambda(X)\rangle \qquad \phi_y(Y) = \langle \boldsymbol{\lambda}_y, \Lambda(Y)\rangle$$

The parameter λ_+ in Equation 5.35 is employed as a placeholder for the maximally required regularization weights $\boldsymbol{\lambda}_x$ and $\boldsymbol{\lambda}_y$ such that the optimizing factor matrices Y and X of Equation 5.35 are binary. Bounding the regularization weights above by the parameter λ_+ ensures that the objective in Equation 5.35 is well-defined. However, we do not need to determine the parameter λ_+ in practice.

The parameter matrices $\boldsymbol{\lambda}_x$ and $\boldsymbol{\lambda}_y$ are the regularization weights of the non-binary penalization terms $\Lambda(X)$ and $\Lambda(Y)$. The Frobenius inner product

$$\langle \boldsymbol{\lambda}, \Lambda(X)\rangle = \sum_{i,s} \boldsymbol{\lambda}_{is}\Lambda(X_{is})$$

sums the elementwise penalization terms weighted by the parameters $\boldsymbol{\lambda}$.

The implementation details of the Broccoli optimization scheme can be found in Hess, Pio, Hochstenbach, and Ceci [312]. In contrast to the Boolean factorization framework PAL-Tiling, the initialization plays an important role for Broccoli. Instead of the vanilla PALM optimization method, Broccoli employs stochastic proximal gradient descent [187].

5.4.7 Experiments

We highlight here a few results from the applications of the PAL-Tiling instance Primp [310] and the Broccoli implementation using a nonnegative matrix factorization for initialization [312]. More experiments than the ones displayed here can be found in the corresponding literature [309, 310, 311, 312].

We compare the PAL-Tiling instance Primp (henceforth indicated as PAL-Tiling) with the available implementations of the BMF methods Panda+[13], Mdl4bmf[14], and Nassau.[14]

We compare Broccoli with six competitors: two methods based on a nonnegative relaxation (henceforth denoted by N [447] and NN [165]), two methods based on an

13 http://hpc.isti.cnr.it/~claudio/web/archives/20131113/index.html.
14 http://people.mpi-inf.mpg.de/~skaraev/.

orthogonal relaxation (henceforth denoted by O [725] and OO [165]) and the biclustering methods FABIA [318] and FLOC [716]. Since N, NN, O and OO return fuzzy membership values for each observation, we binarize the result for comparison purposes. For each sample (observation or feature) we set the top-k fuzzy cluster indicator values to one, where k is the number of ground truth clusters the sample belongs to. Note that in this way we provide our competitors with additional background knowledge, that is not available in real-world scenarios. The goal is to estimate how good the clustering, derived from a relaxed result, could potentially be, if supported by additional knowledge (e.g., from domain experts).

Quality Metrics We quantify how well a computed cluster indicator matrix matches the ground truth by an adaptation of the micro-averaged F_1-measure, known from multi-class classification tasks. We compute a one-to-one matching τ between computed and ground truth clustering and compute the average F_1-measure of the matched clusters. That is,

$$F_Y = \frac{1}{r}\sum_{s=1}^{r} F_1(Y_{\cdot s}, Y^\star_{\cdot \tau_Y(s)}), \qquad F_X = \frac{1}{r}\sum_{t=1}^{r} F_1(X_{\cdot t}, X^\star_{\cdot \tau_X(t)}).$$

We return the average F_1-score of the feature and observation clusters:

$$F = \tfrac{1}{2}(F_Y + F_X).$$

The F_1-measure has values between zero and one. The closer it approaches one, the more the computed clustering matches the ground truth. The plots that display the F-measure indicate its average value with error bars having the length of twice the standard deviation.

5.4.8 Synthetic Dataset Experiments

PAL-Tiling We generate data matrices according to the scheme established by Karaev, Miettinen, and Vreeken [356], Lucchese, Orlando, and Perego [451], and Miettinen and Vreeken [492]. We generate (1600×500) and (1000×800) dimensional datasets as outlined by Hess, Morik, and Piatkowski [310]. Given dimensions d, N, and noise parameter p, a factorization of rank $r^\star = 25$ is generated by uniformly randomly drawing each tile $(X^\star_{\cdot s}, Y^\star_{\cdot s})$ from all tiles of size $|X^\star_{\cdot s}| \in [0.01d, 0.1d]$ and $|Y^\star_{\cdot s}| \in [0.01N, 0.1N]$. Finally, each bit entry $(Y^\star \odot X^{\star\top})_{ji}$ is flipped with probability p.

We compare the effects of the matrix dimensions and aggregate results over 10 generated matrices with dimensions 1000×800 and 1600×500. Figure 5.16 plots the F_1-measure and the rank of the returned BMF against the percentage of noise. NASSAU particularly strongly underestimates the rank for the 1600×500 dimensional matrices. Here, NASSAU returns close or equal to zero tiles, even if the noise is low. This effect can

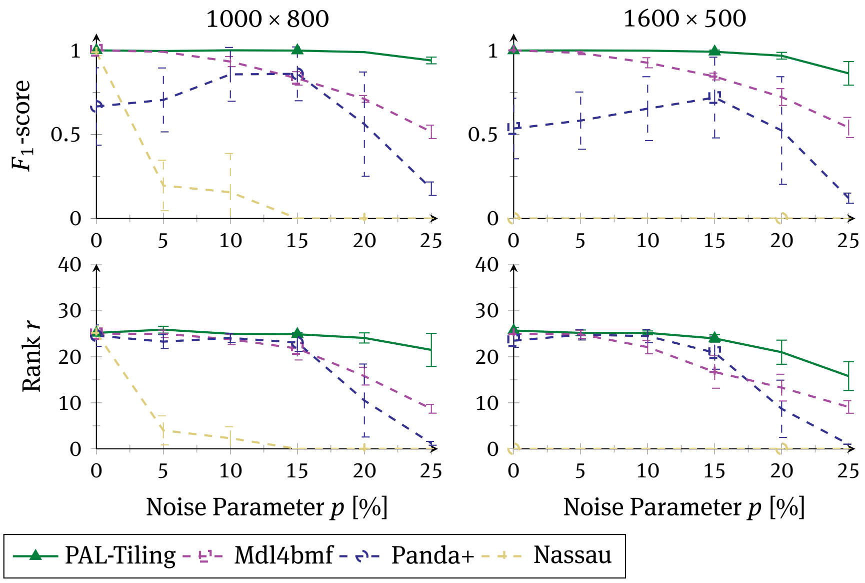


Fig. 5.16: Variation of Bernoulli noise parameter p for 1000 × 800 and 1600 × 500 dimensional data. Comparison of F_1-measures (the higher the better) and the estimated rank of the calculated Boolean matrix factorization (the closer to 25 the better) for varying levels of noise, i.e., p is indicated on the x-axis (best viewed in color).

actually be alleviated if we transpose the matrix, which makes Nassau perform similar to Mdl4BMF, yet with a stronger tendency to underestimate the rank. We observe that all algorithms tend to underestimate the rank the more the noise increases. This culminates in the replication of almost none of the tiles at the highest noise level for the algorithms Panda+ and Nassau. Panda+ yields correct rank estimations up to a noise of 15 %, but its fluctuating F-measure indicates that planted tiles are not correctly recovered after all. Mdl4bmf shows a robust behavior. Its suitable rank estimations up to a noise of 15 % are mirrored in a high F-measure. PAL-Tiling is characterized by overall high values in the F_1-measure. The experiments demonstrate a high robustness of the proposed BMF optimization scheme PAL-Tiling to noise on synthetic data.

Broccoli For the biclusterings generated by a tri-factorization, we create a set of synthetic clusterings with overlap and outliers by sampling every cluster indicator matrix by a Bernoulli distribution. Entry $X^\star_{it}$ and $Y^\star_{js}$ is equal to one with probability $q = 0.2$. Thereby, we ensure that a cluster contains at least 1 % of the data points/features, that are exclusively assigned to this particular cluster. The core matrix is sampled as a sparse matrix containing uniformly distributed values $C_{st} \in [0, 5]$. The probability that a non-diagonal element is not equal to zero is equal to $1/r$. The data matrix is generated

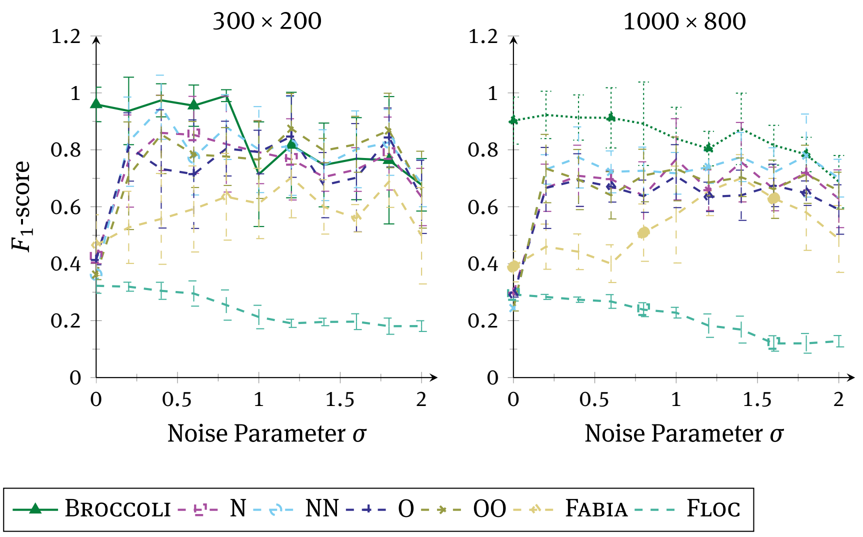


Fig. 5.17: Variation of the Gaussian noise parameter σ, comparison of F-measures (the higher the better) for 300 × 200 data matrices with three row- and column-clusters and 1000 × 800 data matrices with five row- and column-clusters.

by adding random Gaussian noise to the ground truth factorization:

$$D_{ji} = [Y^\star_{j\cdot} C X^{\star\top}_{i\cdot} + \epsilon_{ji}]_{\geq 0},$$

where $\epsilon_{ji} \sim \mathcal{N}(0, \sigma)$ and the operator $[\cdot]_{\geq 0}$ projects negative values to zero. We generate for every noise variance $\sigma \in \{0, 0.2, 0.4, \dots, 2\}$ and dimensionality $(N, d) \in \{(300, 200), (1000, 800)\}$ five datasets. For the smaller 300 × 200 dataset, we choose a rank of $r = 3$ and for the larger 1000 × 800 dataset, we choose a rank of $r = 5$.

In Figure 5.17, we plot the F_1-measure against the Gaussian noise parameter σ. The maximum value of σ is 2.0. For $\sigma = 2$, roughly $1/3$ of the noise samples are larger than or equal to 1.0, and about $2/3$ of all noise samples have an absolute value larger than or equal to 1.0 in expectation. We see that throughout the increase of the noise parameter, Broccoli attains a high F-measure, close to 1.0, which slightly drops when the noise variance exceeds 1.0. The methods N, NN, O, and OO, which are based on orthogonal and nonnegative relaxations seem largely unaffected by the noise parameter, and attain on average an F_1-score between 0.7 and 0.8. Fabia and Floc attain the lowest F_1-score of all competitors, where the F-score of Fabia has a tendency to increase with the noise parameter up to 0.7. This is possibly due to the fact that Fabia and Floc do not explicitly handle the possible presence of noise in the data.

5.4.9 Qualitative Experiments

PAL-Tiling In this experiment, we explore how the algorithms relate to their actual cognition of structure and noise, and illustrate what their biclusters look like. Image data allows us to visually inspect the resulting factorizations and to intuitively assess the captured relevant sub-structures.

We employ a standard representation of images: the RGB888 pixel format. Each of the $w \times h$ pixels is represented by 24 bits, using 8 bits per color (red, green and blue). In order to convert an image into a set of observations, we divide it into blocks (patches) of 4×4 pixels, resulting in a total of $\frac{w}{4} \times \frac{h}{4}$ observations per image. We adopt this representation from computer vision, where image patches are a standard preprocessing step for raw pixel data [340]. Within each block, let $(r, g, b)_{l,k}$ denote the pixel at row l and column k, where $r, g, b \in \{0, 1\}^8$ are the 8-bit binary representation of its red, green, and blue color values. We model the concatenation of all 16 pixels within one block as one observation

$$\left[(r,g,b)_{1,1}, (r,g,b)_{1,2}, (r,g,b)_{1,3}, (r,g,b)_{1,4}, (r,g,b)_{2,1}, \dots, (r,g,b)_{4,4}\right] \quad (5.36)$$

which has a length of $24 \cdot 16 = 384$ bits.

This way, we process a selection of "aliens" from the classic game *Space Invaders*. Reconstruction results and top-4 patterns of the Space Invaders image are shown in Figure 5.18. All methods reconstruct at least the shape of the aliens. In terms of color, however, the results diverge. Panda+ and Nassau interpret all colors as negative noise effects on the color white; white has a binary representation of 24 ones. PAL-Tiling and Mdl4bmf reconstruct all three colors of the original image, yet the reconstruction of Mdl4bmf exhibits injections of white blocks. Hence, only PAL-Tiling is capable of reconstructing the color information correctly.

Having a look at derived biclusters, the greedy processes of Panda+ and Nassau become particularly apparent: Panda+ and Nassau overload the first factor with all the shape information. The remaining factors reduce the quantitative reconstruction error, but have no deeper interpretation. Mdl4bmf tries to model one type of alien by each bicluster. Although this would result in a reasonable description of the image, the actual extraction of tiles suffers from the greedy implementation. For example, we can see that the first tile captures information about the yellow aliens as well as strayed parts of other aliens. This unfortunate allocation of tiles results in the injection of white blocks in the reconstruction image. PAL-Tiling separates by its tiles the three basic color channels that are actually used to mix the colors that appear in the original image. The results of this qualitative experiment illustrate the benefits of a non-greedy minimization procedure.

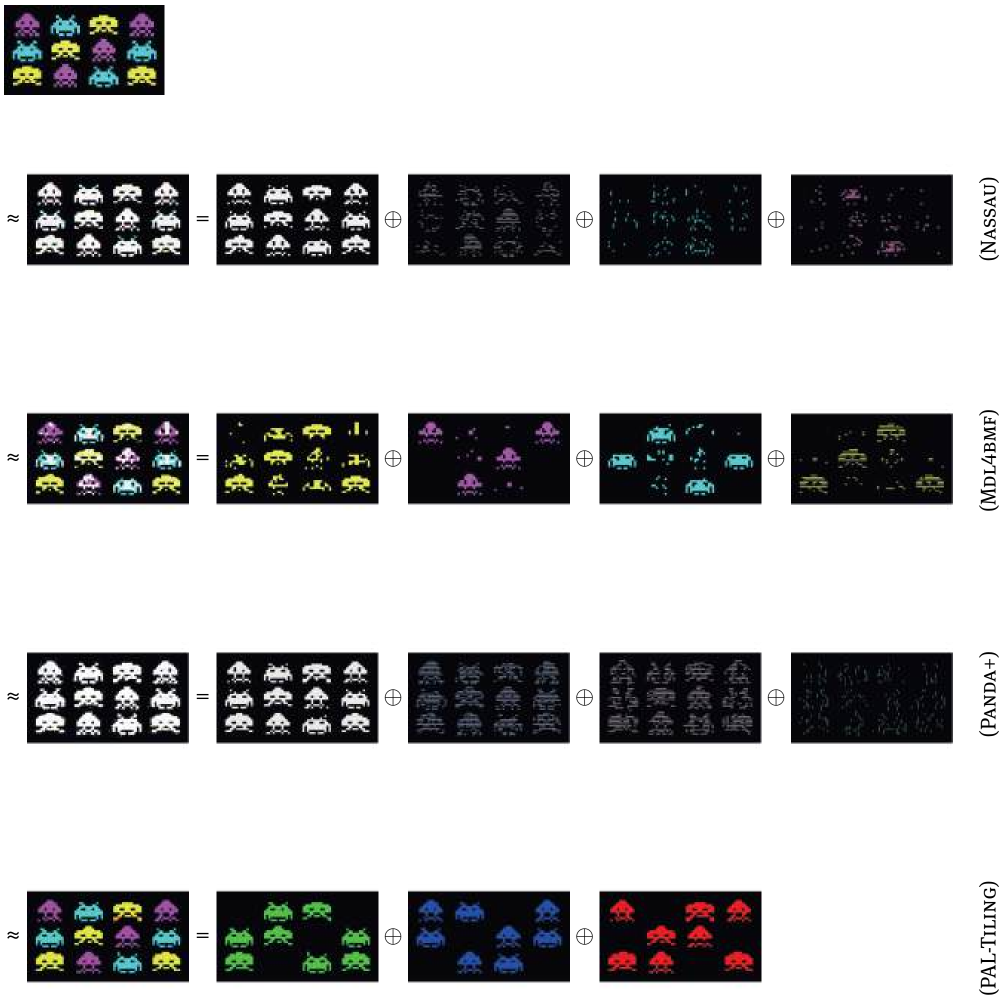


Fig. 5.18: Reconstructions of the Space Invaders image and visualizations of the top-4 outer products. Best viewed in color.

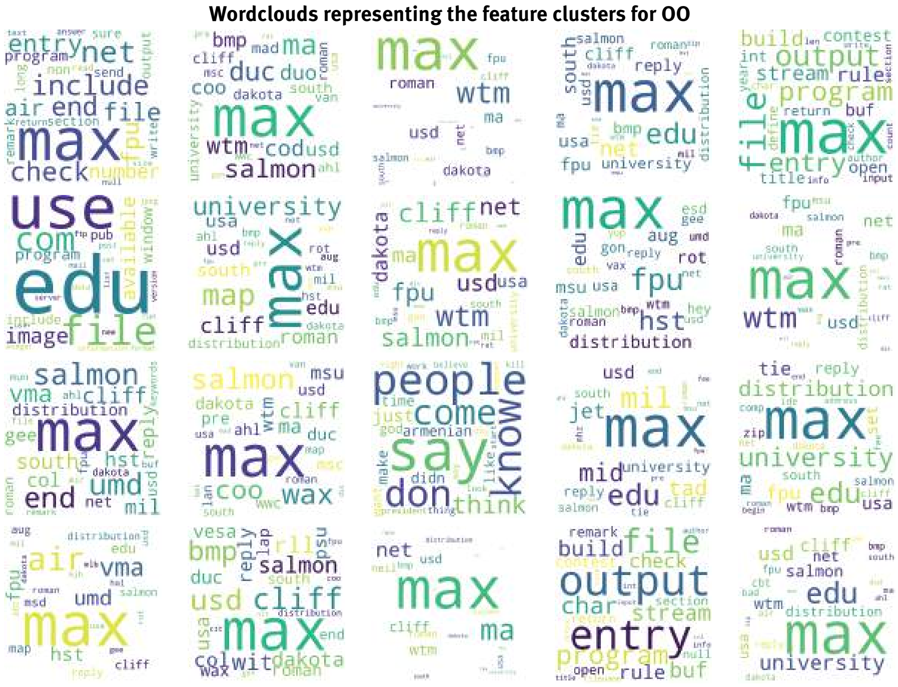


Fig. 5.19: Illustration of derived word-clusters by the method OO on the 20 Newsgroups dataset. The size of a word reflects its weight in the corresponding cluster ($X_{\cdot s}$).

Broccoli We perform a qualitative inspection of the results by means of the *20 Newsgroups* dataset.[15] The 20 Newsgroups dataset is a collection of posts belonging to one of twenty topics that are hierarchically organized. We process the textual data as a data matrix, reflecting for $N = 11\,314$ posts the term-frequency of $d = 6643$ lemmatized words. We apply the methods Broccoli, NN, and OO to derive $r = 20$ row- and column-clusters. Fabia and Floc were not able to successfully process such a large dataset.

The obtained column-clusters (the feature clusters that in this case are clusters of words) are shown in Figures 5.19–5.20. We display here only the fuzzy cluster indicators of the orthogonal relaxation method OO; the results from NN were very similar [312]. The size of word i in the wordcloud of a fuzzy cluster s corresponds to the assigned weight $X_{is} \geq 0$. In turn, the binary word-indicators of Broccoli are visualized such that the size of a word in the cloud is proportional to the inverse of the number of clusters the word is assigned to. That is, those words that are unique to the respective cluster are larger than those words which are assigned to multiple clusters. Looking at the visualizations

15 http://qwone.com/~jason/20Newsgroups/.

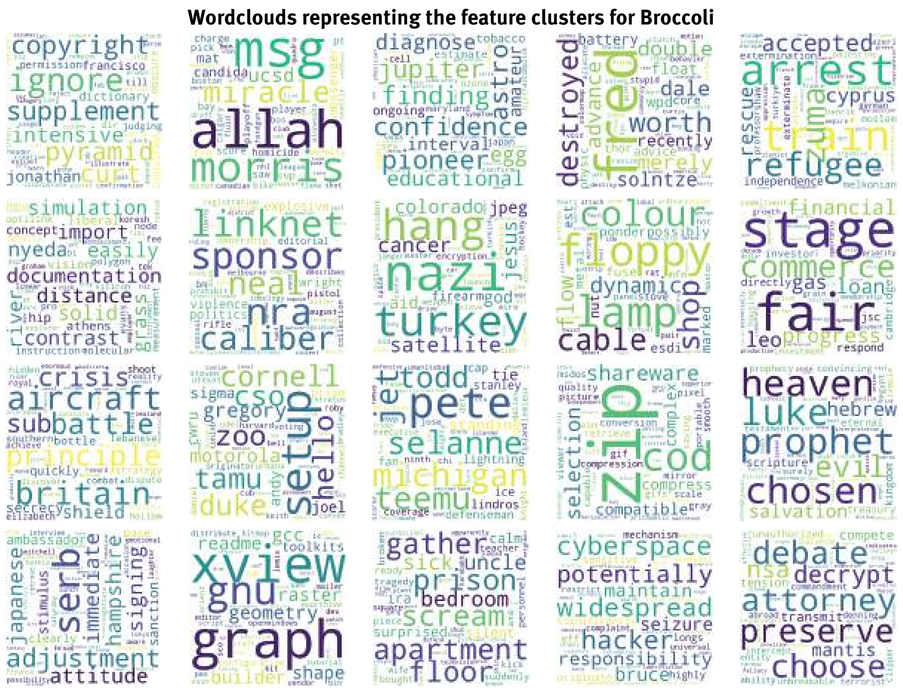


Fig. 5.20: Illustration of derived word-clusters by Broccoli on the 20 Newsgroups dataset. The size of a word reflects its weight in the corresponding cluster ($X_{\cdot s}$).

of clusters, we see that the word *max* pops up prominently in many clusters. The word *max* obtains comparably very high term frequencies. The average term frequency of a word is equal to 1.59, and 99 % of all words have a term frequency smaller than or equal to 8. The word *max* occurs in 149 posts and obtains term frequencies in $[1, 800]$. Hence, the word *max* attains exceptionally high term frequencies in a few posts and exhibits therewith a special role. The unusual high term frequency of the word *max* is handled differently among the clustering methods. While OO gives a high weight to this word in almost all clusters, Broccoli ignores the high frequency of this word. This demonstrates the more modular approach of biclustering with binary cluster-indicator matrices and a robustness to outliers.

We can detect meaningful clusters that address a specific topic for all clustering methods. Comparing the addressed topics among the clustering methods, we see that Broccoli provides a distinctive view on the dataset, identifying, for example, a *religion* cluster that is not featured by the method OO. Hence, although Broccoli's optimization makes use of a relaxed objective, its results still offer another view on the data with respect to that provided by the relaxed counterparts.

5.4.10 Applications, Impact, and Future Work

In this work, we have reviewed the optimization methods for clusterings that have a matrix factorization objective. Our comparison of popular clustering objectives has shown that the majority of methods is designed to find partitioning clusters, adhering to the EXCLUSIVITY CONSTRAINT: every point is assigned to exactly one cluster. Suitable adaptations of Lloyd's algorithm guarantee the convergence to a local optimum of the objective function subject to the partitioning and particularly binary constraints. This offers an undeniable advantage over other practical alternatives that relax the binary constraint or heuristics. The major drawback of relaxing approaches is the discretization step, in which theoretical guarantees, which might be provided for solutions of the relaxed objective, are usually lost. However, the exclusivity constraint, enabling the alternating minimization according to Lloyd, is not feasible in some applications. Overlapping and nonexhaustive clusterings are more likely to represent the *true* model when it comes among other things to the clustering of text or genomic data. In this case, the theoretical foundation regarding the efficient optimization of corresponding objectives is leaky.

We have proposed a general optimization framework for overlapping clusterings by means of proximal alternating minimization. In particular, we have proposed two approaches to optimize biclustering objectives, where the exclusivity assumption is most often inept. The method PAL-TILING is designed for the optimization of a Boolean matrix factorization, which is used to derive overlapping and non-exhaustive clusters of binary data. The method BROCCOLI is designed for a biclustering of real-valued data, based on a tri-factorization.

Our experimental analysis highlights the power of the proposed optimization approach on two instances: the MDL-based BMF method PRIMP (denoted here as PAL-TILING) [310] and the NMF initialization of the BROCCOLI framework [312]. Our experiments on synthetic data indicate in particular the robustness of our proposed optimization approach to noise, amount of overlap, and number of outliers (cf. Figures 5.16 and 5.17). Our qualitative inspection of found clusters indicates the meaningfulness of the found clusters (cf. Figures 5.20 and 5.18).

This makes PAL-TILING and BROCCOLI a theoretically founded, practically well-performing and flexible approach that has the potential to spark further research on the optimization of non-exclusive clusterings in particular, and on the learning of discrete structures in general.

Future Work The proposed optimization approaches are flexible and have the potential to found a standard method for the optimization of clustering structures based on matrix factorization that does not require the exclusivity assumption. Note, that many popular clustering methods are based on (or can be viewed as) a matrix factorization: k-means, spectral clustering, and variants of deep clustering. In addition, techniques to cope with specific data characteristics in matrix factorization can easily be transferred to the optimization scheme adopted in PAL-Tiling and Broccoli.

In addition, nonconvex optimization is an ongoing field of research. There are stochastic [163, 187], accelerated [427], and inertial [581] variants of the PALM optimization scheme. That is, the power of the proposed optimization framework grows with the research on the underlying optimization schemes. An analysis of proposed nonconvex optimization schemes for the optimization subject to binary constraints and clusterings in particular, would be a topic of further research.

6 Hardware-Aware Execution

Efficient learning has been the focus of research for decades. Many studies explore various software/hardware techniques to improve the efficiency of learning process while preserving the accuracy of the derived learning models. Along with the various demands of applications nowadays, from simple like image recognition to advanced like fundamental steering, how to deploy learned models and execute them efficiently has been of key interests in the industry. Considering various resource constraints such as throughput, timeliness, and energy consumption imposed by the targeted scenarios and the adopted hardware platforms, most machine learning techniques, which often rely on high-performance computers and clusters, must be carefully redesigned to fulfill the assigned missions at edge devices while addressing the efficiency of resource usage.

To this end, we summarize in this chapter relevant research conducted in CRC 876, which is oriented to the awareness of hardware execution, and supplement two external contributions to cover a broader spectrum of this research direction. Unlike most existing techniques for executing neural networks on Field-Programmable Gate Arrays (FPGAs), we focus on the of learning, which is actually more computationally demanding (see Section 6.1). In addition, we exploit modern graphics processing units (GPUs) for efficient database query processing (see Section 6.2) and study how parallelization on multicore systems should be deployed for accelerating extreme multi-label classification (see Section 6.3). At the end, we present our RAMBO framework, which can efficiently optimize machine learning models even on heterogeneous distributed systems (see Section 6.4). The mentioned techniques in this chapter tend to reveal different perspectives to achieve efficient learning on various hardware platforms. Although it is not possible to cover all relevant techniques, the introduced insights should clearly reveal that a proper usage of hardware can be very effective, especially for the efficiency of learning process.

https://doi.org/10.1515/9783110785944-006

6.1 FPGA-Based Backpropagation Engine for Feed-Forward Neural Networks

Wayne Luk
Ce Guo

Abstract: Feed-Forward Networks (FFNs), or multilayer perceptrons, are fundamental network structures for deep learning. Although feed-forward networks are structurally uncomplicated, their training procedure is computationally expensive. It is challenging to design customized hardware for training due to the diversity of operations in forward- and backward-propagation processes. In this contribution, we present an approach to train such networks using Field-Programmable Gate Arrays (FPGAs). This approach facilitates the design of reconfigurable architectures by reusing the same set of hardware resources for different operations in backpropagation. In our empirical study, a prototype implementation of the architecture on a Xilinx UltraScale+ VU9P FPGA achieves up to a 5.2 times speedup over the PyTorch platform running on 8 threads on a workstation with two Intel Xeon E5-2643 v4 CPUs.

6.1.1 Introduction

The majority of FPGA-based deep learning architectures are for inference procedures, which makes predictions using pre-trained networks. However, training is a computationally demanding procedure that limits the application of deep neural networks. Backpropagation is the core process in the training procedure of neural networks. This contribution discusses an FPGA-based architecture for backpropagation for Feed-Forward Networks (FFNs). An FFN consists of multiple layers of connected nodes. Each node in a layer takes the weighted sum of the activation signals from the previous layer and generates an activation signal by evaluating a nonlinear activation function.

We study FFNs for two reasons. First, as stand-alone models, they are useful in learning problems with unstructured information such as non-image and non-sequential data. For instance, in the event classification problem for the Higgs boson [3], the correlation between attributes are difficult to capture with other neural networks such as Convolutional Neural Networks (CNNs), while FFNs can provide decent accuracy. Second, as deep network components, FFNs usually appear as the decision-maker in convolutional neural networks; they also serve as generators and discriminators in Generative Adversarial Networks (GANs) [266].

Although FFNs appear less complicated than other neural types of neural networks, training FFNs efficiently on FPGAs is challenging. For instance, in convolutional neural networks, a layer of nodes may share a small set of parameters. This parameter-sharing property naturally reduces on-chip memory usage. However, FFNs do not have similar properties as each connection between a pair of nodes carries a unique scalar parameter, resulting in a high memory bandwidth usage.

Unlike research in general hardware design for neural network training like [431] and [374], we pay attention to the features and limitations of reconfigurable hardware. The following summarizes the challenges that we face.

1. Challenge of diverse arithmetic operations. The hardware resources on an FPGA chip stay unchanged while the arithmetic operations frequently change during backpropagation. To reuse the hardware resources and minimize the overhead for reconfiguration, it is desirable to design multifunctional hardware modules that support multiple operations without runtime reconfiguration.
2. Challenge of hardware adaptability. The size of the network and hardware resources vary greatly in different applications. A proper hardware design should be adaptive to all reasonable network sizes and hardware platforms.
3. Challenge of complex control logic. When a multifunctional hardware module supports more operations, the control logic becomes more complicated. In particular, it is challenging to define a set of commands to control the hardware modules to collaborate at different stages in backpropagation.

It is difficult to find off-the-shelf solutions because most existing systems for gradient computation and training run on CPUs and GPUs. The novel aspects covered in this contribution include the following:

1. Reuse of hardware resources for different operations. We extend the multifunctional multiplication block in [278] to allow different operations to share the same set of hardware resources without runtime reconfiguration, which addresses the challenge of diverse arithmetic operations. In addition, the resource usage of the architecture is independent of the network layout, which addresses the challenge of hardware adaptability.
2. Commands to control the hardware. To use the same set of hardware resources for different operations without significant loss of efficiency, it is necessary to find a proper set of commands to control the hardware. We define a small set of commands in this section, addressing the challenge of complex control logic.
3. Empirical evaluation. We conduct experiments to compare an experimental implementation of the hardware on an FPGA platform with the PyTorch deep learning framework running on CPUs and GPUs.

6.1.2 Background and Related Work

This section provides a short introduction to Feed-forward networks and and their hardware-based training approaches.

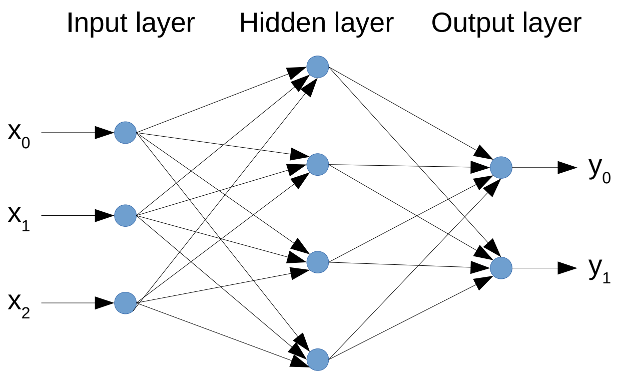


Fig. 6.1: An example feed-forward network.

6.1.2.1 Feed-Forward Networks and Backpropagation

A feed-forward network contains nodes arranged in layers. For instance, Figure 6.1 shows a layout of an FFN with three layers of nodes and two layers of connections. Each node in the input layer feeds a feature of the data point to the network. All nodes in layer l receive signals from all nodes in the previous layer $(l-1)$ and produce an output signal in the form of a vector. The generation of output signal involves two steps. The first step is to calculate the weighted-sum vector:

$$\mathbf{t}_l = W_{l-1}\mathbf{x}_{l-1} \tag{6.1}$$

where $\mathbf{x}_l$ is a vector containing the output of all nodes in layer l; and W_{l-1} is a matrix of real numbers specifying N_l weight vectors. The second step is to produce an output signal x_l with

$$\mathbf{x}_l = f_{\text{act}}(\mathbf{t}_l) \tag{6.2}$$

where $f_{\text{act}}(\cdot)$ is a non-linear function defined on real vectors.

It is necessary to specify the weights $W = \{W_0 \ldots W_{L-1}\}$ before using the network to make predictions. One may find out the weights using data via training. A training algorithm searches for a set of weights that fits a dataset D with respect to an error measure $E(W, D)$. An efficient training algorithm typically updates the parameter set using the gradient

$$\nabla W = \frac{\partial E(W, D)}{\partial W} \tag{6.3}$$

to minimize the error on the training data. The de facto method to compute the gradient ∇W is backpropagation [422].

An episode of backpropagation includes a forward pass and a backward pass of signals. In the forward pass, the network takes a data point and computes a prediction following the direction of the network. In the backward pass, the network propagates an error signal in the opposite direction to compute the gradient. Note that the back-propagation process does not include the optimization algorithm, which uses gradients to update the weights [421].

The backpropagation process is computationally demanding. The sheer amount of network parameters results in problems in the algorithms and hardware. With regard to algorithms, the high dimensionality of the parameter space slows down the convergence of the optimization procedure. As a result, the optimization algorithm needs to invoke a large number of backpropagation episodes before obtaining an accurate network model. With regard to hardware, parameters consume considerable memory space and IO bandwidth during computation because the nodes do not share parameters.

The Graphics Processing Unit (GPU) is arguably the most widely-used hardware platform to implement training algorithms for neural networks. The GPU platform provides high performance with relatively low hardware costs and a short design cycle. However, trends in the development of machine learning suggest that FPGAs may become more promising than GPUs for two reasons. First, more neural networks will be based on customized data types, especially quantized numbers [326]. GPUs can natively support only a limited number of data types. By contrast, FPGAs can support customized data types efficiently. Second, the performance gap between GPUs and FPGAs is narrowing fast [541]. In particular, the size of on-chip memory, clock speed, number of hardware DSP units, memory bandwidth, and the process technology of FPGAs have significantly advanced.

6.1.2.2 Reconfigurable Computing for Neural Network Training

Among the statistical models for classification and regression, neural networks are some of the most popular candidates for reconfigurable acceleration [488]. Because we focus on the training process, we do not cover the hardware engines that only perform inference. Reviews that cover inference engines include [408, 497, 500, 546]. The first type of solutions is the acceleration of the training process of general-purpose neural network architectures. Eldredge and Hutchings [198] divide the backpropagation algorithm into three stages and design hardware for each stage separately. During the training process, the hardware performs a runtime reconfiguration at the beginning of each stage. Paul and Rajopadhye [558] propose a systolic backpropagation engine that avoids the runtime reconfiguration. In their design, all calculations in a complete background procedure are mapped to hardware. Murugan et al. [522] designs a training architecture for a network with five nodes. An implementation on a Xilinx Virtex-E FPGA runs at 5.332 MHz. Li and Pedram [437] propose a coarse-grain architecture mainly to implement the matrix multiplication operations in training. Langhammer and Pasca [417] discuss architectures that evaluate common activation functions with different approximation methods. Kim et al. [374] present the DeepTrain platform to perform energy-efficient training for various types of deep networks. The DeepTrain platform offers tools to generate sequences of operations for the hardware architecture using network descriptions extracted from the TensorFlow deep learning framework. Maeda and Tada [458] propose a training engine for neural networks using the simultaneous perturbation rule [457] to avoid the gradient computation in the training process.

The second type of solutions is the design and optimization of hardware-oriented neural network structures. A popular network structure in this category is the Block-based Neural Network (BbNN). Moon and Kong [502] first propose the BbNN and implement the prediction facility on the FPGA platform. A BbNN connects a collection of neutron blocks. A neutron block carries four numeric I/O ports. Each I/O port may serve as either an input port or an output port. An output port provides an activation signal computed from the input signals on the same neutron block. Jiang et al. [344, 345, 346] study the training process of the BbNN using evolutionary algorithms on the CPU platform. The idea behind their training approach is to encode the topology of the network and the configuration of the neural blocks into a vector so that an evolutionary algorithm may improve the network by manipulating the vector. Merchant and Peterson [488] make it possible to train the block-based neural networks on the FPGA platform. The third type of solutions is the customization of domain-specific or problem-specific neural networks. A representative neural network structure in this category is the convolutional neural network. The convolutional neural network [403] is a feed-forward network structure inspired by the visual cortex of animals. The major application of CNNs is image recognition [403]. The reconfigurable acceleration solutions for CNNs usually take advantages of the unique properties of structured data. Farabet et al. [212] propose an FPGA-based RISC processor that matches basic operations in the CNN. The processor uses a description of a pre-trained CNN in the form of a sequence of instructions to make predictions. A useful observation in this section is that a proper reduction of the precision of image operations results only in a little negative impact on the predictive accuracy. However, the precision reduction may save a considerable amount of hardware resources. Further optimizations [428, 684, 734] have made FPGA devices faster and more energy efficient than CPUs and GPUs. Zhao et al. [744] propose the first stand-alone training engine for CNNs using a streaming data path. The data path contains a collection of parameterized modules. The organization of these modules changes over time with the runtime reconfiguration, which enables the data path to train different layers of a network. In addition to CNNs, FPGA-based reinforcement learning methods have emerged in recent years. Shao and Luk [625] present an architecture trust region policy optimization, which allow robots or agents to efficiently learn policies by interacting with an environment. An implementation on an Intel Stratix-V FPGA achieves up to a 20 times speedup against a 6-thread software reference on an Intel Core i7-5930K CPU running at 3.5 GHz. Gankidi et al. [240] design a Q-learning architecture for a planetary robot. An implementation of the architecture on an Xilinx Virtex-7 FPGA achieves 43 times speedup compared to an Intel 6-gen i5 CPU running at 2.3 GHz.

6.1.3 Architecture for Backpropagation

This section presents the hardware architecture of the backpropagation engine and automated generation of control sequences. During a backpropagation process, two major operations consume the majority of execution time.

1. Linear combination. A node takes the linear combination of the outputs from all nodes in the previous layer. In the forward pass, the operation combines activation signals. In the backward pass, the operation combines error signals and computes gradients.
2. Non-linear function evaluation. A node evaluates a real-valued non-linear function. In the forward pass, the operation evaluates an activation function. In the backward pass, the operation evaluates the derivative of the activation function.

The two major operations are computationally demanding because their time complexity depends on the network layout [421]. Specifically, the time spent on linear combination grows linearly with the number of network parameters. By contrast, the time spent on function evaluation grows linearly with the number of nodes in the network. Other operations in backpropagation are less computationally expensive than the two major operations. For instance, it is necessary to compare the actual label of the training data and the prediction to calculate the initial error signal for the backward pass. However, the comparison runs only once in a backpropagation episode, and the calculation typically has linear complexity regarding the data dimension.

The two major operations are fundamentally different regarding arithmetic operations. A straightforward way to customize hardware architecture is to create separate arithmetic modules for both operations [437]. However, the sequential dependencies between the calculations allow only the execution of one type of operation at the same time. Therefore, when the arithmetic module for one operation is active, the corresponding arithmetic modules work with a full load, but the modules for all other operations are idle. In other words, only a small fraction of logic units work, resulting in wastage of hardware resources. Admittedly, it is possible to prepare separate bitstreams such that each operation takes all hardware resources [198, 744]. However, it is necessary to reconfigure the hardware to switch between operations. Frequent runtime reconfiguration may take a considerable amount of execution time.

We design a hardware block that works for all backpropagation operations. Our objective is to allow different operations to share as many arithmetic facilities as possible. We use a command sequence to dynamically switch between operations by adjusting the behavior of a small subset of arithmetic units without incurring runtime reconfiguration. Figure 6.2 shows the top-level diagram of the architecture, which supports both linear combination and function evaluation. Major components include the buffer crossbar and the arithmetic block.

- The buffer crossbar communicates with two buffers. At any time, the buffer control signal from the command specifies a source buffer and a target buffer. The crossbar

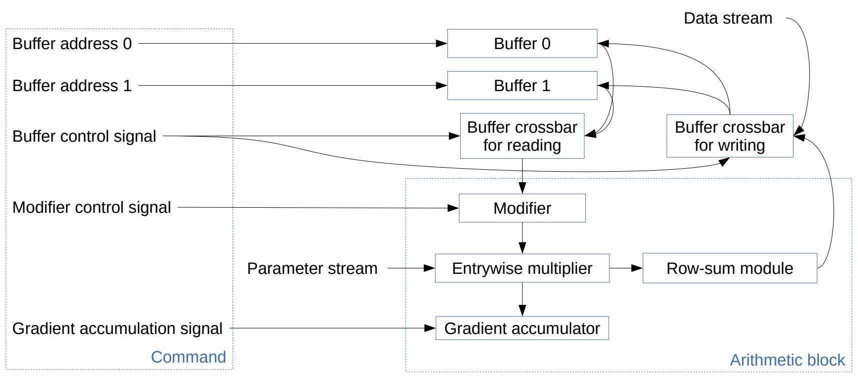


Fig. 6.2: Top-level diagram of the backpropagation engine.

reads a vector from the source vector and passes the vector to the arithmetic block. The crossbar also accumulates the output vector from the arithmetic block to the target buffer.

– The arithmetic block extends the multifunctional multiplication block proposed in [278]. The modifier in the arithmetic block corresponds to the overrider in [278]. This block has two execution modes: the linear mode, which multiplies a matrix by a vector, and the function evaluation mode, which evaluates a non-linear function for all elements in the vector. Our extension includes the gradient accumulator and the row-sum module. A binary signal from the command controls whether the multiplier accumulates the entry-wise products to the gradient. The entrywise multiplier feeds its results to a row-sum module which calculates the sum of each row in the linear mode.

The arithmetic module switches to the linear mode for linear combination. The core calculation for a linear combination operation is to evaluate a vector of weighted sums using a b-dimensional input vector $\mathbf{x}$ and a $b \times b$ weight matrix W. We evaluate an approximate version in the form of a piecewise linear function.

The architecture addresses two challenges in Section 1. First, the components and their connections are independent of the operation. As a result, it is unnecessary to perform a runtime reconfiguration when the operation changes, which addresses the challenge of diverse arithmetic operations. Second, the resource usage is independent of the layout of the network. Therefore, it is possible to scale the architecture for different hardware platforms to control cost or power consumption, which addresses the challenge of hardware adaptability.

One may follow the design flow illustrated in Figure 6.3 to apply the architecture to a backpropagation task. The design flow includes hardware customization and command sequence generation. Hardware customization is the process to set two design parameters. One design parameter is the batch size b, which determines the size of

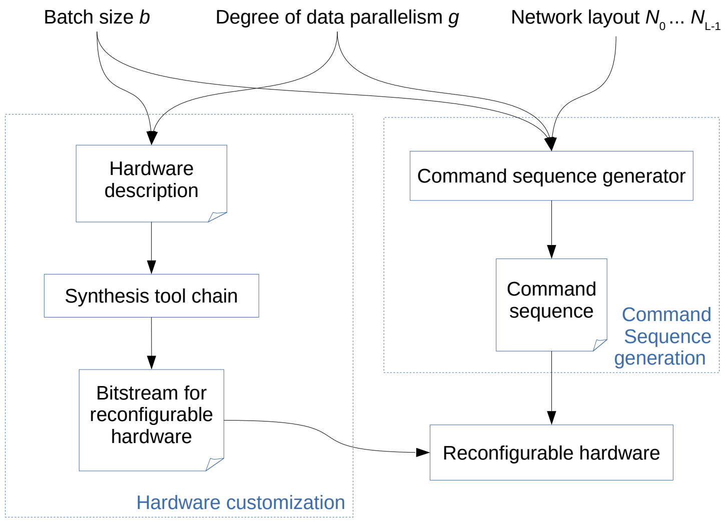

Batch size b
Degree of data parallelism g
Network layout $N_0 \dots N_{L-1}$
Hardware description
Synthesis tool chain
Bitstream for reconfigurable hardware
Hardware customization
Command sequence generator
Command sequence
Command Sequence generation
Reconfigurable hardware

Fig. 6.3: Design flow.

Tab. 6.1: Memory traffic (number of data entries per command).

Operation	**ME**	**Inward**	**Outward**
Data load	0	bg	0
Linear combination for forward pass	0	b^2	0
Function evaluation for forward pass: batch 0..(U – 2)	1	b^2	0
Function evaluation for forward pass: batch (U – 1)	1	b^2	b
Linear combination for backward pass: node error	0	b^2	0
Linear combination for backward pass: gradient output	0	b	b^2
Function evaluation for backward pass	0	0	0

the entrywise multiplier. A larger b allows the entrywise multiplier to process more multiplications in parallel for a single data point. The other parameter is the degree of data parallelism g, which determines the number of data points processed in parallel. After customization, the architecture has g arithmetic blocks. Each arithmetic block contains a modifier, an entrywise multiplier, and a row-sum module. Each entrywise multiplier includes $b \times b$ scalar multipliers. After filling the design parameters to the hardware description, it is possible to generate a bitstream to program the reconfigurable hardware using the synthesis toolchain. Command sequences generation is the process that produces a sequence of commands from the layout of the network.

The memory bandwidth usage of the hardware depends on the operation and the hardware parameters. For ease of discussion, we assume that all data entries in the feature matrix, network parameters, and gradients have the same width. We may measure the memory traffic by the number of data entries transmitted per command. Table 6.1 summarizes the memory traffic for different operations.

6.1.4 Collaboration of Components

In this section, we explain how the components collaborate to execute different operations in backpropagation. The modifier in the arithmetic block determines whether the system performs linear combination or non-linear function evaluation.

- The modifier switches to the linear mode for linear combination. The core calculation for a linear combination operation is straightforward. In particular, the arithmetic block evaluates a vector of weighted sums using a b-dimensional input vector $\mathbf{x}$ and a $b \times b$ weight.
- The modifier switches to the function evaluation mode for non-linear function evaluation. Rather than evaluating an activation function following its mathematical definition, the arithmetic block evaluates an approximate version in the form of a piecewise linear function.

Before running the hardware, it is necessary to define a list of commands to control the hardware architecture. We briefly discuss a set of commands that can be used to perform backpropagation in a straightforward manner. This command set addresses the challenge of complex control logic discussed in Section 1. We first describe two commands that read and write the same buffer including data load and memory reset. We then present the commands for linear combination and function evaluation where the arithmetic block read and write different buffers.

The architecture supports two commands that operate on a single buffer. The first command sets the addressed location in the target buffer to zero. As the arithmetic block always accumulates to the target buffer, it is necessary to initialize N_l entries in the target buffer to zero to ensure correct calculation. A command to reset a memory location needs to set the source buffer to be the same as the target buffer and point both addresses to the location to reset. The parameter memory provides a $b \times b$ negative identity matrix. With these settings, the output of the row-sum module is the opposite of the original value $-\mathbf{x}_t$. The accumulation of the value back to the target memory location resets the content to zero. The second command loads d dimensions from g data points to the target location. The memory crossbar directly reads a data point from the data stream, ignoring the output of the arithmetic block.

The other two commands operate on two buffers. The first command is for the linear combination operation. In each batch, the arithmetic block takes b signals as the input and begins to propagate the signals to b nodes in the adjacent layer in parallel. The second command is for the function evaluation operation. Each modifier takes a copy of the variable and evaluates the piecewise linear function that approximates the activation function.

Tab. 6.2: Resource usage.

Resource	Total	Used	Percentage
Logic utilization	1 182 240	382 256	32.33 %
DSP blocks	6840	4105	60.01 %
Block memory (BRAM18)	4320	1292	29.91 %
Block memory (URAM)	960	150	15.63 %

6.1.5 Evaluation

We empirically evaluate the architecture in this section by comparing an FPGA implementation of the architecture and the PyTorch machine learning platform on a dual-CPU workstation.

6.1.5.1 Experiment Settings

We compare our architecture running on an FPGA-based acceleration card with a CPU implementation running on a multicore CPU. The architecture runs on a Xilinx UltraScale+ VU9P FPGA with 16 nm technology. We run the FPGA chip at 120 MHz. The architecture executes command sequence to $g = 16$ data instances in parallel with dimensional batch size $b = 8$. Table 6.2 shows the resource usage of the implementation. The software implementation runs on the PyTorch 1.0 machine learning platform running on a workstation with two Intel Xeon E5-2643 v4 CPUs and 128 GB DDR4 memory. The process technology of the CPUs is 14 nm, which is slightly more advanced than the FPGA. The workstation has 12 physical cores supporting 24 threads in total. The base frequency of the CPU cores is 3.4 GHz, and the maximum turbo frequency is 3.8 GHz.

We consider two representative types of network layouts in the experiments. We call them *bucket-shaped* networks and *cone-shaped* networks respectively for ease of discussion. In a “bucket-shaped” network, all layers contain an identical number of nodes. These networks usually appear in stand-alone classifiers, generative models, and reinforcement learning. In a “cone-shaped” network, a hidden layer has no more nodes than its previous layer. These networks learn compressed features and representation as each layer introduces information loss in a controlled manner.

Table 6.3 shows the test cases we designed using network layouts similar to those in real-world applications. We test two activation functions–rectifier linear function (relu) and the hyperbolic tangent function (tanh)–for each layout. Due to the alignment requirement of our hardware platform, we round the size of each layer to the next multiple of 32. We also produce challenging test cases for each application by linearly scaling the size of all layers. Specifically, the design of test cases is as follows.

– The experiments with “bucket-shaped” networks include 12 test cases. Test cases B0 and B1 correspond to the network structure for reinforcement learning in [276]. The network has 2 hidden layers with 200 nodes in each layer. Test cases B2 and B3 correspond to a study of traffic-flow prediction [454]. The network with the largest layer size contains 3 layers of hidden nodes with 400 nodes in each layer. Test cases B4 and B5 correspond to the network in the generative adversarial networks in [23]. The network contains 4 hidden layers with 512 nodes in each layer. Test cases B6–B11 are challenging versions for B0–B5, where the layer size of each case is 8 times that of the original version.

– The experiments with "cone-shaped" networks include 8 test cases. Test cases C0 and C1 correspond to the stacked autoencoder in [713]. The network has two hidden layers containing 400 and 225 hidden nodes, respectively. Test cases C2 and C3 correspond to the denoising autoencoder for speech data recognition in [221]. The network contains two hidden layers, one with 1000 nodes and another with 500. Test cases C4–C7 are challenging versions for C0–C3, where the layer size of each case is 4 times that of the original version.

We use randomly generated data and network parameters to test the efficiency of the system. Assuming that the function evaluation procedure takes the same time for different inputs, the total execution time for each backpropagation process is independent of the data distribution and the network parameters. In other words, given the same data size, the total execution time should stay unchanged regardless of the data source. As a result, using randomly generated data and network parameters facilitates experiments with various data sizes without affecting observations. The number of data instances for each test case is 2^{20}. In each test case, we calculate the gradient with respect to 100 sets of weights.

6.1.5.2 Results and Discussion

Table 6.3 records the experimental results. In this table, the benchmark column records the numbers of data instances; the 'CPU-1T', 'CPU-4T', and 'CPU-8T' columns contain the execution times in seconds for the corresponding implementation. The 'SU' columns give the speedup of the FPGA implementation over the CPU with 1 thread, 4 threads, and 8 threads, respectively.

The architecture discussed in this contribution is faster than the CPU system in all but one test case. In the tests with bucket-shaped networks, the architecture achieves up to a 9.4, 5.4, and 5.2 times speedup compared with the software reference on 1, 4, and 8 threads. In the tests with cone-shaped networks, the architecture achieves up to a 7.8, 4.6, and 4.7 times speedup compared with the software reference on 1, 4, and 8 threads. In addition to the overall speed advantage of the architecture, we have the following additional observations:

1. The software implementation scales poorly with the number of threads. The software running 4 threads achieves only around 2 times speedup against a single thread. The speed advantage on 8 threads over 4 threads is insignificant. In test

Tab. 6.3: Execution time (seconds) and speedup.

ID	Layers	Activation	CPU-1T	CPU-4T	CPU-8T	FPGA	SU-1T	SU-4T	SU-8T
B0	224x2	tanh	0.285	0.219	0.155	0.071	4.0	3.1	2.2
B1	224x2	relu	0.181	0.104	0.090	0.071	2.5	1.5	1.3
B2	416x3	tanh	0.731	0.356	0.274	0.105	7.0	3.4	2.6
B3	416x3	relu	0.521	0.202	0.162	0.104	5.0	1.9	1.6
B4	512x4	tanh	1.020	0.504	0.430	0.138	7.4	3.7	3.1
B5	512x4	relu	0.786	0.320	0.235	0.137	5.7	2.3	1.7
B6	1792x2	tanh	4.102	2.493	2.109	0.615	6.7	4.1	3.4
B7	1792x2	relu	3.715	2.163	1.820	0.615	6.0	3.5	3.0
B8	3328x3	tanh	21.758	13.120	13.176	2.576	8.4	5.1	5.1
B9	3328x3	relu	20.827	13.783	12.400	2.574	8.1	5.4	4.8
B10	4096x4	tanh	45.501	24.847	25.286	4.822	9.4	5.2	5.2
B11	4096x4	relu	44.755	24.036	24.320	4.817	9.3	5.0	5.0
C0	416,256	tanh	0.179	0.125	0.167	0.089	2.0	1.4	1.9
C1	416,256	relu	0.137	0.086	0.102	0.087	1.6	1.0	1.2
C2	1024,512	tanh	0.577	0.339	0.421	0.169	3.4	2.0	2.5
C3	1024,512	relu	0.494	0.266	0.299	0.169	2.9	1.6	1.8
C4	1664,1024	tanh	2.092	1.189	1.259	0.375	5.6	3.2	3.4
C5	1664,1024	relu	2.121	1.104	1.021	0.375	5.7	2.9	2.7
C6	4096,2048	tanh	13.467	7.997	8.147	1.732	7.8	4.6	4.7
C7	4096,2048	relu	13.207	7.643	7.636	1.725	7.7	4.4	4.4

cases B10 and B11, the software running on 8 threads is even slower than when running on 4 threads.

2. The software is more sensitive to the selection of the activation function than the architecture. With the same network layout, the software tends to be faster with the rectifier linear function than with the hyperbolic tangent function. The speed advantage is especially significant when the software runs on 4 and 8 threads. This observation confirms the speed advantage of the rectifier linear function on CPUs [255]. By contrast, the architecture on FPGA evaluates any activation using the same number of commands. Therefore, the execution time of the architecture for a given layout is independent of the activation function.
3. The experimental implementation achieves slightly higher speedup with larger networks that carry more network parameters. For instance, in the experiments with bucket-shaped networks with the 'tanh' function, the speedup over 8 threads rises from 2.2 to 5.2 while the layer size expands from 224 to 4096. An exception of the observation is that architecture achieves high speedup against a single CPU thread in test case B0. A possible reason for the exception is that artifacts such as memory initialization for both software and hardware take significant time when the network is small. In this case, the comparison is less reliable than it is in other tests.

Besides the observations above, we have two conjectures based on the hardware design and the experimental results. First, the speed of the architecture will grow if more DSP blocks are available. The arithmetic block contains b^2g scalar multipliers in parallel. Our synthesis tool implements these multipliers mainly with DSP blocks. Therefore, DSP blocks become the critical resource for the design, as shown in Table 6.2. The number of data points processed in parallel grows linearly with the number of multipliers. As a result, when more DSP blocks are available, we may set a larger g to deploy more multipliers to improve the speed. Second, given the same set of hardware resources, our architecture can process networks with more nodes and parameters than some existing solutions such as [558] and [522] for two reasons. One reason is that the number of multipliers is independent of the network layout. The other reason is that the on-chip memory only needs to keep the activation and error signals for two adjacent layers.

6.1.6 Conclusions

We presented a hardware architecture to perform backpropagation for training multilayer perceptrons. The key to acceleration is to reuse the same set of hardware resources to process different operations involved in backpropagation. Our architecture does not incur runtime reconfiguration when switching between operations. The hardware resource usage is independent of the network layout. A prototype implementation of the architecture on a Xilinx UltraScale+ VU9P FPGA achieves up to 5.2 times speedup over PyTorch running on 8 threads on a workstation with two Intel Xeon E5-2643 v4 CPUs.

6.2 Processor-Specific Code Transformation

Henning Funke
Jens Teubner

Abstract: During the last decade, the compilation of database queries to machine code has emerged as a very efficient alternative to classical, interpretation-based query processing modes [529]. Compiled code can better utilize advanced features of modern CPU instruction sets; avoid interpretation overhead; and—most importantly—minimize data I/O (*e.g.*, to main memory).

This success story raises the hope that compilation strategies can be lifted to non-standard architectures, such as GPUs or other accelerators, as well as to support other data-intensive processing tasks. However, as we shall see in this section, the data-parallel nature of the devices is at odds with established techniques in query compilation, resulting in massive resource under-utilization if compilation strategies are applied too naively.

As a remedy, we propose two novel mechanisms that re-establish compute efficiency of compiled code on data-parallel hardware: *Lane Refill* and *Push-Down Parallelism* are "virtual operators" that participate in optimization and code generation just like true query operators (making our approach seamlessly integrate with existing systems). At runtime, they compensate for lurking resource under-utilization by adapting parallelization strategies on-the-go. The outcome is a resource utilization that is close to the hardware's maximum, while causing negligible overhead even in unfavorable situations.

Lane Refill and *Push-Down Parallelism* are part of our compiler platform *DogQC*, which leverages modern graphics processors for efficient database query processing.

6.2.1 Data-Parallel Processing Models

Data-parallel processing models are a particularly promising way to max out the achievable compute performance within the constraints of hardware technology (power and heat dissipation). Instead of dedicating chip resources to control flow management, data-parallel architectures target throughput. For instance, executing an instruction for 32 fields at a time can reduce the control flow management work by a factor of 32, when compared with a scalar execution.

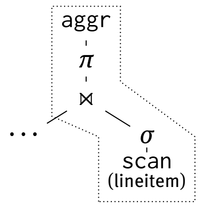


Fig. 6.4: Plan excerpt.

6.2.1.1 Divergence in Data-Parallel Architectures

GPUs are a popular incarnation of this idea, and spectacular performance results have been reported in various application domains. However, actually leveraging the available hardware resources in a beneficial way can be challenging. *Divergence effects*, which may arise whenever data is not perfectly regular, may compromise the benefits.

In this section, we will look at mechanisms to combat performance penalties that may result from divergence effects. To understand the divergence problem, let us consider the execution of a database query, as illustrated here in Figure 6.4 for Query Q10 from the TPC-H benchmark set. A query compiler will attempt to compile the plan region ⬚ into a straight-line sequence of code, *i.e.*, a *pipeline*. The motivation to do so is to propagate tuples within registers rather than spilling data to (slow) memory.

During execution, not all lineitem tuples will actually traverse the full pipeline. Some tuples might instead be *eliminated* by operators such as filter σ or join $\bowtie$. If this happens, a sequential processor will immediately abort the pipeline, continue with the next input item, and hence keep CPU efficiency at peak.

Data-parallel execution back-ends, by contrast, do not have the option of aborting a pipeline early, unless *all* tuples in the same batch of work are eliminated.

Figure 6.5 illustrates this effect for a GPU-based back-end (assuming a batch—or "*warp*"—size of eight for illustration purposes). In some warp iteration, only *warp lanes* 1, 5, and 7 might have passed the filter σ, leaving the five remaining warp lanes *inactive* (indicated as dashed arrows $\dashrightarrow$). The following join de-activates another two warp lanes, bringing GPU efficiency down to 1/8 in this example.

The resulting GPU under-utilization is even worse in real settings. To scan a lineitem table with 150 million rows, actual GPUs will require 5 million *warp iterations*, each consisting of 32 warp lanes. Although σ filters out about 2/3 of all rows, it is extremely unlikely that all lanes within a warp become inactive. Therefore, (almost) all 5 million warp iterations proceed into the join operator $\bowtie$. Only 1 % of the remaining rows find a match during the join. In an actual dataset, 2.9 million rows remain after the join, but they are spread across 1.1 million warp iterations. Ideally, the projection π and aggregation aggr operators could have been processed by only 2.9 M/32 = 90 K warp iterations. In other words, state-of-the-art query compilation techniques will leave 92 % of the GPU's processing capacity unused.

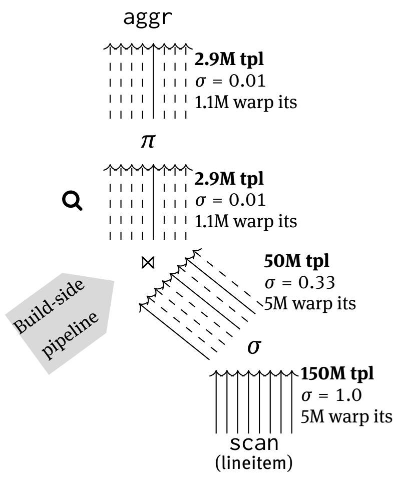


Fig. 6.5: GPU under-utilization due to filter divergence.

6.2.1.2 *DogQC*: A Database Query Compiler for GPUs

GPU code generated by our query compiler *DogQC*[1] leverages *Lane Refill* and *Push-Down Parallelism* techniques to counter divergence effects like the ones illustrated above. In the rest of this section, we will give a high-level idea of the *Lane Refill* and *Push-Down Parallelism* techniques (Sections 6.2.2 and 6.2.3), then report on experimental results for DogQC (Section 6.2.4), and wrap up in Section 6.2.5. More details on the *Lane Refill* and *Push-Down Parallelism* mechanisms can be found in the respective full paper [237].

6.2.2 *Lane Refill* Technique

Divergence effects (here: *filter divergence*) are a consequence of the SIMT ("single instruction, multiple threads") execution paradigm embodied in all modern graphics processors. A number of threads (or *lanes*, typically 32 of them) are grouped into a *warp*. During execution, *all* lanes within a warp execute the *same* GPU instruction.

The SIMT model encounters a problem whenever some lanes or data elements need a different amount or kind of processing than others. In such situations, control flows will *diverge*. Since all lanes within a warp *still* execute the same instruction, lanes will be turned *inactive* and their computation result will be discarded. As illustrated above, this can result in resource under-utilization.

To illustrate the severity of this effect, we instrumented the query plan shown earlier (Figure 6.5) to monitor warp utilization at the plan point marked with a magnifying glass 🔍. Figure 6.6 shows a histogram on the number of warps that have passed this

1 https://github.com/Henning1/dogqc.

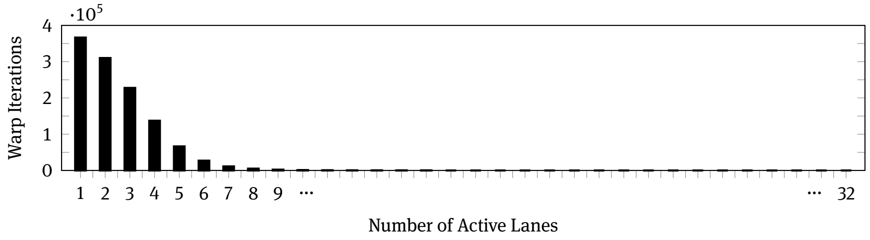


Fig. 6.6: Lane activity profile with filter divergence.

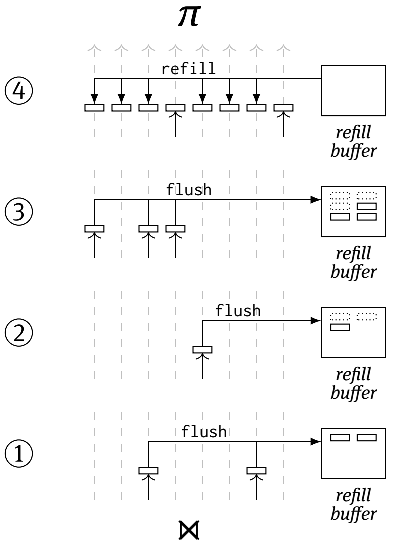


Fig. 6.7: *Lane Refill*: tuples from three low-activity iterations are suspended to the *refill buffer* and resumed for full lane activity in the fourth iteration.

plan stage with a warp utilization of 1, …, 32 active lanes. It is easy to see that only a fraction of the available compute capacity is used; in most warps, only one or two out of 32 warp lanes performed actual work.

6.2.2.1 Balance Operators and Refill Buffers

To combat the situation, DogQC injects *balance operators* into the relational query plan. Code generated for these operators detects warp under-utilization at runtime. Whenever utilization drops below a configured threshold, the state of all remaining active lanes is suspended to a *refill buffer* and the pipeline starts over with a fresh set of input tuples.

Figure 6.7 illustrates this for three successive warp iterations ➀ through ➂. Since only 2, 1, and 3 lanes remained active in these iterations (respectively), their state is

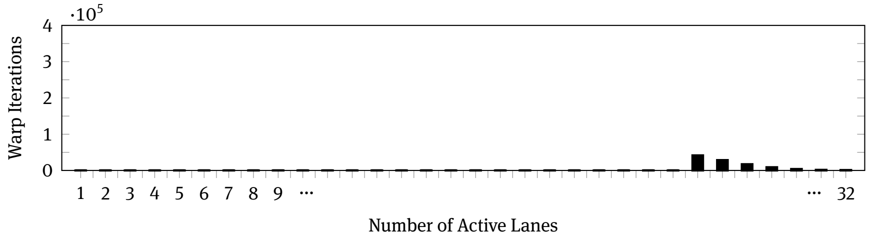


Fig. 6.8: Lane activity profile with lane refill buffer to consolidate filter divergence.

flushed to the refill buffer. After flushing, each of those warp iterations is terminated and processing starts over with the next set of input tuples.

6.2.2.2 Refilling

As soon as a sufficient number of lane states have been stored to the refill buffer, the buffer can be used to *refill* lanes that have become inactive. This time, the under-utilized warp iteration is not terminated but continues processing with full utilization after refilling. This is visualized in Step ④ of Figure 6.7. Here, only two out of eight warp lanes remained active after the downstream join operator. Using the refill buffer, the remaining six warp lanes can be filled with useful work, resulting in full warp utilization upstream.

Implementationwise, flushing and refilling are backed up in DogQC by CUDA's `__ballot_sync`, `__popc` ("population count"), and shuffling primitives. These primitives are highly efficient; balance operators will cause little overhead even when only a few warps go below the utilization threshold.

6.2.2.3 Effect of *Lane Refill*

Lane Refill brings warp utilization back to a high compute efficiency. Following the balancing operator, all executed warps (except for the last warp in each grid block) are *guaranteed* to have a warp utilization above the configured threshold.

In Figure 6.8, this is illustrated with a histogram for the same plan point that we profiled earlier (Figure 6.6), but this time with a balance operator applied. The histogram confirms that *(a)* (almost) no warps exist with a utilization below 26 lanes (the threshold we configured); and *(b)* the total number of executed warps has dropped by a factor of about 10. In terms of overall execution performance, *lane refill* will improve execution times by about 2 – 3× for the example plan shown in Figure 6.5.

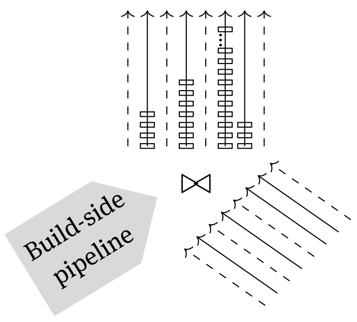


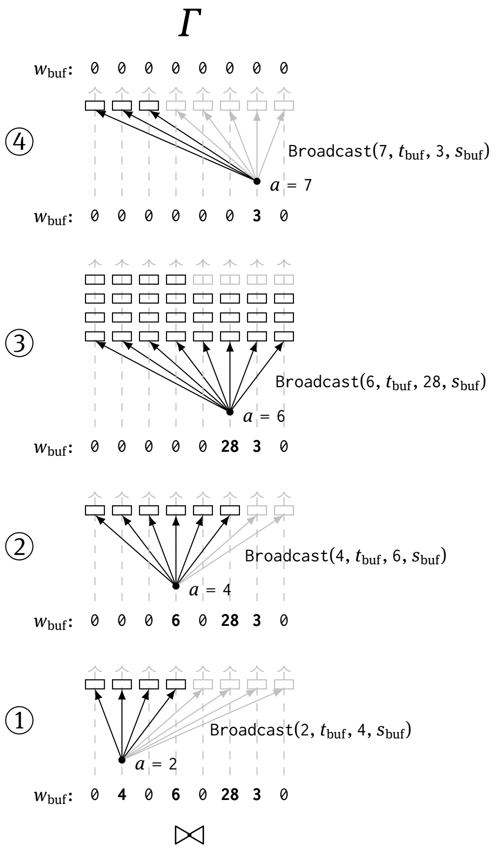
Fig. 6.9: Expansion divergence. Here, some rows in the probe-side relation will find more join partners on the probe side than on the other side.

6.2.3 *Push-Down Parallelism* Technique

DogQC's *Push-Down Parallelism* technique addresses another flavor of divergence that may arise orthogonally to the aforementioned filter divergence. *Expansion divergence* is the effect when a different amount of work is needed to process each of the items within a warp. Database *join operations* are a common situation where this effect arises. Figure 6.9 on the right illustrates the effect. Probe side tuples coming from the right may find a different number of join partners each. Specifically, in the example, lane 6 will have significantly more tuples to process than the remaining warp lanes. In such a situation, existing query compilers will process all matches of a single probe-side tuple *within* the same warp lane. In the example, execution times would be dominated by the sequential processing of all matches for lane 6.

Push-Down Parallelism mitigates the situation by parallelizing the processing of the matches of a single probe-side tuple *across* the available warp lanes. To this end, the execution state of probe-side lanes is *broadcast* over lanes, as illustrated in Figure 6.10. Build-side matches are *partitioned* across. Again, we leverage efficient CUDA primitives, such as `__ballot_sync` and `__shfl_sync` ("shuffle sync"). Please refer to [237] for details.

As illustrated in Figures 6.11 and 6.12, *Push-Down Parallelism* improves lane utilization and reduces the overall number of iterations needed to complete the query. *Lane Refill* and *Push-Down Parallelism* complement one another, and Figure 6.9 shows an example where both flavors of divergence co-exist. Another typical occurrence of expansion divergence is the processing of *variable-length data*, strings in particular. If possible, DogQC will parallelize the processing of strings across warp lanes to improve resource utilization.



Γ
w_{buf}: 0 0 0 0 0 0 0 0
④
Broadcast(7, t_{buf}, 3, s_{buf})
a = 7
w_{buf}: 0 0 0 0 0 0 3 0
③
Broadcast(6, t_{buf}, 28, s_{buf})
a = 6
w_{buf}: 0 0 0 0 0 28 3 0
②
Broadcast(4, t_{buf}, 6, s_{buf})
a = 4
w_{buf}: 0 0 0 6 0 28 3 0
①
Broadcast(2, t_{buf}, 4, s_{buf})
a = 2
w_{buf}: 0 4 0 6 0 28 3 0
$\bowtie$

Fig. 6.10: Illustration of push-down parallelism that expands the join matches of four warp lanes.

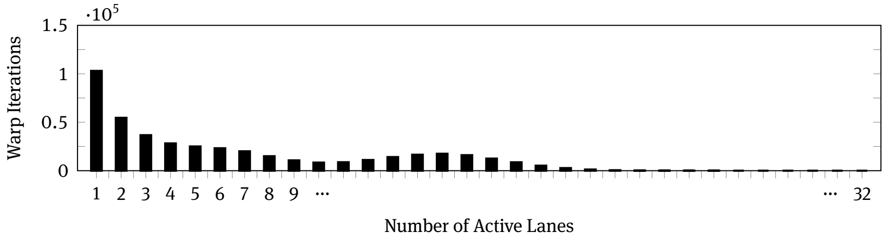


Fig. 6.11: Lane activity with expansion divergence.

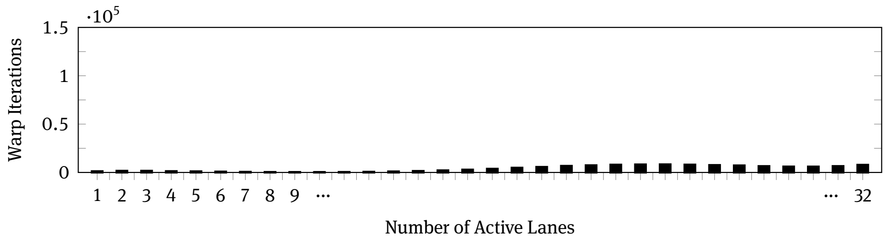


Fig. 6.12: Lane activity profile with push-down parallelism to consolidate expansion divergence.

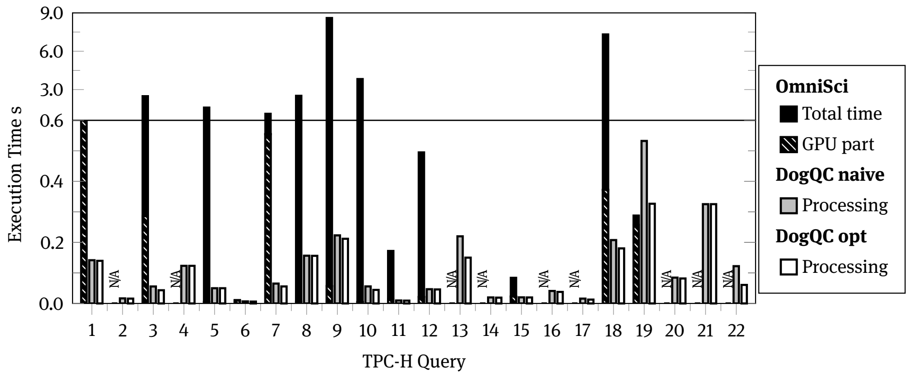


Fig. 6.13: Execution times of DogQC for TPC-H benchmark queries (scale factor 25). The divergence optimizations improve query performance.

6.2.4 Evaluation

With *DogQC*, we provide a query compiler with a wide range of SQL functionality, sufficient to support all queries from the TPC-H benchmark set. Here we use DogQC and the database domain to illustrate the aforementioned anti-divergence mechanisms, which could equally be applied to other data-intensive tasks, including those related to machine learning.

6.2.4.1 TPC-H Performance

To assess the benefits of measures to contain divergence, we performed a series of measurements with the TPC-H benchmark set. Our measurements were based on an NVIDIA RTX2080 GPU with 46 Streaming Multiprocessors and 8 GB GPU memory, installed in a host system with an Intel i7-9800X GPU and 32 GB of main memory. As a reference, we compared DogQC with the hybrid CPU/GPU system *OmniSci* [545].

Our benchmark results are depicted in Figure 6.13. For each of the 22 TPC-H queries, the bars indicate the query execution time assuming that the dataset is resident in GPU memory.

For OmniSci, we report the total wall clock time needed to execute the query as well as the amount of time spent on GPU processing. OmniSci is a hybrid execution engine in which both CPU and GPU will be used to jointly answer the query. As can be seen in Figure 6.13, several queries can, in fact, not benefit much from GPU in OmniSci. Also note that OmniSci could successfully execute only 13 of the 22 TPC-H benchmark queries. DogQC, by contrast, can run all 22 TPC-H queries entirely on the GPU, with execution times that are up to 86× faster than those of OmniSci.

6.2.5 Summary

In this research, we put the processing capabilities of data-parallel co-processors for non-uniform, data-intensive workloads to the test. DogQC introduces techniques that allow us to gracefully align parallel processing units with work items, even when problems are heavily skewed. We observe that *Lane Refill* and *Push-Down Parallelism* are able to increase processing efficiency for these non-uniform workloads, sometimes with dramatic effects on processing throughput.

Existing query coprocessors typically avoid imbalances by working on a uniform surrogate (such as dictionary keys or materialization barriers). This has led to the perception that GPUs have limited capabilities of processing irregular problems. DogQC avoids the overhead of maintaining such additional data structures and instead restores balance during non-uniform processing.

Here we showcase *Lane Refill* and *Push-Down Parallelism* based on an application to database query processing. Compared with state-of-the-art platforms, our prototype DogQC achieves better resource utilization, a bigger functionality range, and better runtime performance on realistic benchmarks. Looking ahead, our anti-divergence measures could be applicable to many machine learning scenarios, especially when the problems involved are heavily skewed and/or depend on non-linear computations.

6.3 Extreme Multicore Classification

Erik Schultheis
Rohit Babbar

Abstract: There are classification problems, such as assigning categories to a Wikipedia article, where the possible set of labels is very large, numbering in the millions. Somewhat surprisingly, these so-called Extreme-Multilabel Classification (XMC) problems can be solved quite successfully by applying a linear classifier to each label individually. This decomposition into binary problems is called a one-vs-rest reduction. As these problems are completely independent, the reduced task is embarrassingly parallel and can be trivially spread across multiple cores and nodes. After training, the model can be sparsified by culling small weights to only require a fraction of the memory and computational power for prediction on new samples.

6.3.1 Introduction to Extreme Multilabel Classification

Extreme Multi-label Classification (XMC) refers to supervised learning with a large target label set where each training/test instance is labeled with a small subset of relevant labels. Machine learning problems consisting of hundreds of thousands of labels are common in various domains such as annotating web-scale encyclopedias [585], hashtag suggestion in social media [171], and image-classification [168]. For instance, all Wikipedia pages are tagged with a small set of relevant labels that are chosen from more than a million possible tags in the collection. It has been demonstrated that, in addition to automatic labelling, the framework of XMC can be leveraged to effectively address learning problems arising in recommendation systems, ranking, and web-advertising [9, 585].

Notation and Setup Let the training data $D := \left\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\right\}$ consist of input feature vectors $\mathbf{x}^{(i)} \in \mathcal{X} \subseteq \mathbb{R}^d$ and respective output vectors $\mathbf{y}^{(i)} \in \mathcal{Y} := \{0, 1\}^m$ such that $y_l^{(i)} = 1$ iff the l-th label belongs to the training instance $\mathbf{x}^{(i)}$. The feature vectors form the rows of the feature matrix $\mathbf{X}$. In XMC settings, the cardinality m of the set of target labels, the dimension of the input d, and the size of the dataset N can all be of the order of hundreds of thousands or even millions.

For text data, the input can be represented by term-frequency inverse-document-frequency (tf-idf) features. In that case, the dimensionality of the feature space is determined by the size of the vocabulary, and for each text $\mathbf{x} \in \mathcal{X}$ the feature x_j is nonzero only if the corresponding word appears in the text. As a result, the input features are highly sparse. The magnitude of the feature is determined by how often

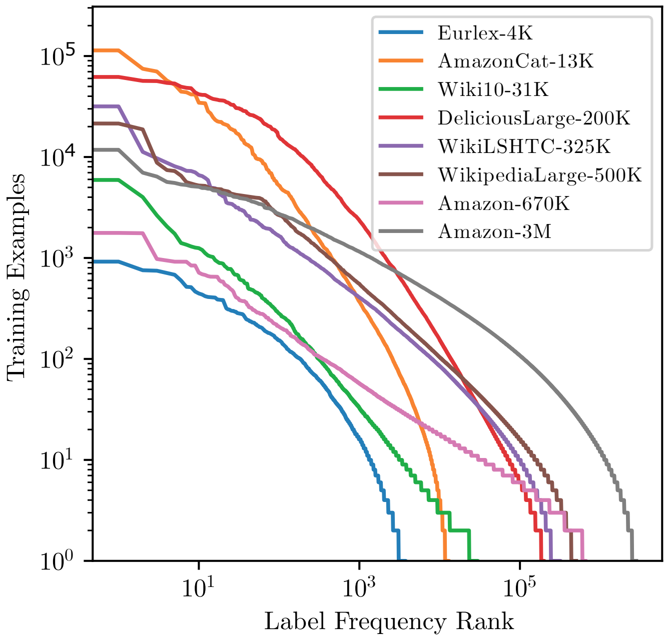


Fig. 6.14: Label frequency in XMLC datasets. X-axis shows the label rank when sorted by the frequency of positive instances and Y-axis gives the number.

the word appears in the document and in the entire corpus. For details on tf-idf, see, e.g., Manning, Raghavan, and Schütze [463].

Similarly, for any given instance $\mathbf{x}^{(i)}$ only a small subset of the labels will be relevant, $\|\mathbf{y}^{(i)}\|_1 \ll m$. Additionally, the number of instances for which a label is relevant is very imbalanced: Few labels will be relevant to many instances, but most labels will apply only to an extremely small fraction. This gives rise to a *long-tailed* label distribution, as shown in Figure 6.14. The labels with very few positives are called *tail labels*. The characteristics of well-known benchmark datasets in XMC are presented in Table 6.4.

In traditional multi-label classification, the goal is to learn a multi-label classifier in the form of a vector-valued output function $h : \mathbb{R}^d \mapsto \{0, 1\}^m$. In XMC, one often wants to restrict the classifier to predict a fixed number of labels because, say, a web interface might have a fixed number of slots in which to suggest related searches. This leads to classification functions $h_k : \mathbb{R}^d \mapsto \mathcal{Y}_k := \{\mathbf{y} \in \mathcal{Y} : \|\mathbf{y}\|_1 = k\}$. Such a function is typically constructed by first learning a score function $r : \mathbb{R}^d \mapsto \mathbb{R}^m$, and then taking the k highest-scoring labels as the prediction.

Evaluation Metrics Due to the extreme sparsity in the label vector, metrics such as accuracy are not informative in the case of XMC. Instead, one typically uses metrics that

Tab. 6.4: Multi-label datasets from XMC repository [50]. APpL and ALpP represent average points per label and average labels per point, respectively.

Dataset	#Training	#Features	#Labels	APpL	ALpP
AmazonCat-13K	1 186 239	203 882	13 330	448.6	5.0
AmazonCat-14K	4 398 050	597 540	14 588	1 330.1	3.5
Amazon-670K	490 449	135 909	670 091	4.0	5.5
Wiki10-31K	14 146	101 938	30 938	8.5	18.6
Delicious-200K	196 606	782 585	205 443	72.3	75.5
WikiLSHTC-325K	1 778 351	1 617 899	325 056	17.5	3.2
Wikipedia-500K	1 813 391	2 381 304	501 070	24.8	4.8

focus on the k predicted labels. Most commonly used are precision at k, denoted P@k, and normalized Discounted Cumulative Gain, denoted nDCG@k [50]. Let $\mathbb{R}^m \ni \hat{\mathbf{y}} = r(\mathbf{x})$ be the predicted scores for an instance with corresponding label vector $\mathbf{y}$. These metrics are defined by

$$\mathrm{P@}k(\mathbf{y}, \hat{\mathbf{y}}) := \frac{1}{k} \sum_{l \in \mathrm{rank}_k(\hat{\mathbf{y}})} y_l \tag{6.4}$$

$$\mathrm{nDCG@}k(\mathbf{y}, \hat{\mathbf{y}}) := \sum_{l \in \mathrm{rank}_k(\hat{\mathbf{y}})} \frac{y_l}{\log(\mathrm{R}_l(\hat{\mathbf{y}}) + 1)} \Big/ \sum_{l=1}^{\min(k, \|\mathbf{y}\|_1)} \frac{1}{\log(l + 1)}, \tag{6.5}$$

where $\mathrm{rank}_k(\mathbf{y})$ returns the k largest indices of $\mathbf{y}$ ranked in descending order, and R_l gives the ordering of the l'th index. Note that unlike P@k, nDCG@k takes into account the ranking of the correctly predicted labels. For instance, if there is only one of the five labels that is correctly predicted, then P@5 gives the same score if the correctly predicted label is at rank 1 or rank 5. By contrast, nDCG@5 gives a higher score if it is predicted at rank 1 and the lowest non-zero score at rank 5.

6.3.2 Parallel Training of Linear One-vs-Rest Models

The P@k (Equation 6.4) and nDCG@k (Equation 6.5) metrics introduced above are non-differentiable and thus not directly usable in typical gradient-based empirical-risk-minimization procedures. However, it can be shown that in order to achieve optimal predictions for precision at k, one only needs to train the scoring function r in such a way that the scores are strictly monotone transformations of the labels' marginals [486]. Therefore, one can train the classifier by independently applying a classification-calibrated[2] loss ℓ_{BC}, such as binary cross entropy or (squared) hinge loss, to each label

2 See e.g. Bartlett, Jordan, and McAuliffe [44]. Intuitively, this means that a classifier that minimizes ℓ_{BC} also minimizes the binary 0-1 loss.

individually. Therefore, the training objective is given by

$$\min_{r} \quad \sum_{i=1}^{N}\sum_{l=1}^{m} \ell_{\mathrm{BC}}\left(y_l^{(i)}, r_l\left(\mathbf{x}^{(i)}\right)\right). \tag{6.6}$$

Such a decomposition is called the *One-vs-Rest* (or One-vs-All) reduction.

Objective Functions for Linear One-vs-Rest This expression becomes particularly favorable if the scoring function r is linear. In that case, the minimization task decomposes into m completely independent subtasks

$$\forall l \in [m] : \min_{\mathbf{w}^{(l)} \in \mathbb{R}^d} \sum_{i=1}^{N} \ell_{\mathrm{BC}}\left(y_l^{(i)}, \mathbf{x}^{(i)\mathsf{T}}\mathbf{w}^{(l)}\right). \tag{6.7}$$

Due to the embarrassingly parallel nature of the training tasks, the computation can easily scale to use thousands of compute cores. A scalable implementation of this method yielding state-of-the-art prediction performance was demonstrated via the DiSMEC algorithm [30], which is a multi-label wrapper around the Liblinear solver [210]. In DiSMEC, the underlying binary loss is the squared hinge loss with an additional l_2 regularization term. Its objective is

$$\forall l \in [m] : \min_{\mathbf{w}^{(l)} \in \mathbb{R}^d} \left(\|\mathbf{w}^{(l)}\|_2^2 + c \sum_{i=1}^{N} \left(\max(0, 1 - s_l^{(i)}\mathbf{x}^{(i)\mathsf{T}}\mathbf{w}^{(l)})\right)^2 \right), \tag{6.8}$$

where $c \in \mathbb{R}_{>0}$ is the parameter to control the trade-off between empirical error and the model complexity and $s_l^{(i)} := 2y_l^{(i)} - 1$ is the label represented as $\{+1, -1\}$.

A similar method is ProXML [29], which switches the l_2 regularization for l_1 regularization in order to induce robustness to l_∞ perturbations in the input samples. This robustness is particularly helpful for tail labels, which have very few positive training instances. The objective of ProXML thus is

$$\forall l \in [m] : \min_{\mathbf{w}^{(l)} \in \mathbb{R}^d} \left(\|\mathbf{w}^{(l)}\|_1 + c \sum_{i=1}^{N} \left(\max(0, 1 - s_l^{(i)}\mathbf{x}^{(i)\mathsf{T}}\mathbf{w}^{(l)})\right)^2 \right). \tag{6.9}$$

Suppose for now that we have a method $\mathcal{A} : (\mathbf{X}, \mathbf{s}) \mapsto \mathbf{w}_\star^{(l)}$ available to solve these individual problems efficiently in a single thread. (This will be discussed in Section 6.3.3). Then the following framework can be used to scale the training process to multiple cores and nodes:

Two-Level Parallelization The distributed training for the optimization problems defined by equations (6.8) and (6.9) is implemented using a two-layer parallelization architecture. At the top level, labels are separated into batches of, say, $M = 1000$, which can be processed independently in parallel on available compute nodes, or sequentially if the number of batches exceeds the number of nodes. On each node, training of a batch of M labels is parallelized using multiple threads, which forms the second layer of parallelization.

After each $\mathbf{w}_{\star}^{(l)}$ is trained, weights of small magnitude are pruned to reduce overall model size drastically, often by more than 99 %. Since this can be performed as soon as $\mathbf{w}_{\star}^{(l)}$ has been computed, there is no need to store the complete dense model, even for a single batch, which reduces the RAM requirements for the algorithm. Unfortunately, in typical sparse matrix formats such as compressed sparse row/column matrices, insertion of new values cannot be done by multiple threads in parallel, because it might require reallocation and the shifting of data in other parts of the matrix. For this reason, we represent the sparse weight matrix as an array of independently allocated sparse vectors that can be written to independently.

The two-layer distributed training framework is summarized in Algorithm 4.

Algorithm 4: Framework for hardware-aware embarrassingly parallel training in DiSMEC and ProXML solvers. The iterations of both loops are independent and can thus be run in parallel.

```
Input: Training data $D = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}) \ldots (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\}$ in sparse representation,
       input dimensionality $d$, label set $\{1 \ldots m\}$, batch size $M$
Output: Learnt matrix $\mathbf{W} \in \mathbb{R}^{d \times m}$ in sparse format
// 1st parallelization; independent nodes
1 for $\{b = 0;\ b < \lfloor \frac{m}{M} \rfloor + 1;\ b{+}{+}\}$ do
2     Load single copy of feature matrix $\mathbf{X}$ into main memory
3     Prepare array $\mathbf{W}_b$ of $M$ sparse vectors
      // 2nd parallelization; independent threads
4     for $\{l = b \times M;\ l \leq (b+1) \times M;\ l{+}{+}\}$ do
5         Generate binary sign vector $\mathbf{s}^{(\ell)} = \{+1, -1\}_{i=1}^{N}$
6         train weight vector $\mathbf{w}_{\star}^{(l)}$ on a single core using $\mathcal{A}(\mathbf{X}, \mathbf{s}^{(l)})$,
7         Prune small weights in $\mathbf{w}^{(l)}$
8     return $\mathbf{W}_{d,M}$
9 return $\mathbf{W}$
```

An advantage of the two-level parallelization over just running m instances of an off-the-shelf solver for binary problems is that the feature matrix $\mathbf{X}$ can be shared for all training jobs running on the same node. This allows us to keep the entire dataset, which may be several gigabytes in size, in main memory. However, on modern CPUs with a large number of cores, or on nodes with a two-socket configuration, this might cause problems due to *Non-Uniform Memory Access* (NUMA). In such a system, the overall RAM is partitioned into regions, called *NUMA domains*. Even though all cores in the

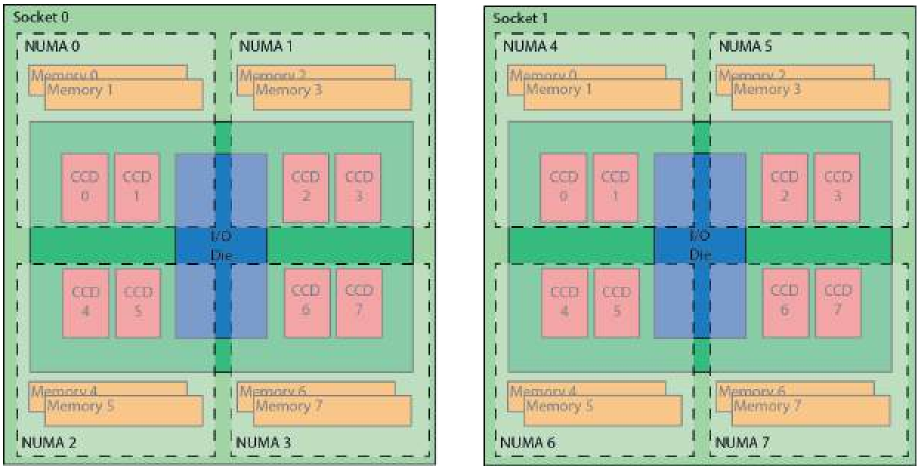


Fig. 6.15: NUMA memory in a dual-socket 64-core AMD Rome 7H12 system. Each CCD contains 8 cores, and each core has fastest access only to the memory within its own NUMA domain, marked with dashed lines. Image by CSC - IT Center for Science under CC-BY-4.0.

system can access the entirety of the memory, access latencies to the different domains vary depending on the distance of the core to the domain. An example of a NUMA setup is given in Figure 6.15.

In such a setup, a single copy of the feature matrix would be accessed by all cores on the system, quickly bottlenecking the memory bus and preventing the program from making efficient use of the available CPU cores. This can be mitigated by pinning threads to their CPU cores and replicating the feature matrix once per NUMA domain. In this way, each thread can read the feature matrix from its local domain, reducing latency, and the memory reads are spread out across different memory modules, improving throughput. In order to achieve this, the outer parallelization layer has to be performed not over physical nodes but over NUMA domains.

6.3.3 Second-Order Optimization Using Conjugate Gradients

The objective Equation 6.8 can be minimized in batch mode using second-order optimization. Compared with the popular (stochastic) gradient descent strategy, second-order optimization can take the curvature of the loss landscape into account and thus converges to the minimum in much fewer iterations. However, the computations for each single iteration are much more involved, as the second-order information is encoded in the potentially very large Hessian matrix. Fortunately, it is possible to implement this procedure without ever actually forming the Hessian, as will be described below.

Dropping the label index, we can write

$$\mathrm{R}_D[\mathbf{w}] = \mathbf{w}^\mathsf{T}\mathbf{w} + c\sum_{i=1}^{N} \ell_{\mathrm{SH}}\left(s_i\mathbf{x}^{(i)\mathsf{T}}\mathbf{w}\right), \tag{6.10}$$

where ℓ_{SH} is the squared hinge loss

$$\ell_{\mathrm{SH}}(r) = \max(0, 1-r)^2. \tag{6.11}$$

Note that this objective function is convex. As a consequence, the optimizer will converge to the global optimum regardless of the starting point.

Determining the Descent Direction The main idea of second-order optimization is to approximate the objective locally using its quadratic Taylor approximation

$$\mathrm{R}_D[\mathbf{w}+\boldsymbol{\delta}] \approx \mathrm{R}_D[\mathbf{w}] + \nabla\mathrm{R}_D[\mathbf{w}]\boldsymbol{\delta} + 0.5\boldsymbol{\delta}^\mathsf{T}\nabla^2\,\mathrm{R}_D[\mathbf{w}]\boldsymbol{\delta}. \tag{6.12}$$

Therefore, the step $\boldsymbol{\delta}_\star$, which is ideal, i.e. which leads to the minimum, in this approximation can be calculated by solving the linear system

$$\nabla^2\,\mathrm{R}_D[\mathbf{w}]\boldsymbol{\delta}_\star = -\nabla\mathrm{R}_D[\mathbf{w}]. \tag{6.13}$$

For Equation 6.8, the gradient and Hessian have a simple structural form [239, 368]

$$\nabla\mathrm{R}_D[\mathbf{w}] = 2\mathbf{w} + c\sum_{i=1}^{N}\ell'_{\mathrm{SH}}\left(s_i\mathbf{x}^{(i)\mathsf{T}}\mathbf{w}\right)s_i\mathbf{x} \tag{6.14}$$

$$\nabla^2\mathrm{R}_D[\mathbf{w}] = 2\mathbf{I} + c\mathbf{X}^\mathsf{T}\mathbf{D}\mathbf{X}, \tag{6.15}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{X} = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}]^\mathsf{T}$ is the data matrix, and $\mathbf{D}$ is diagonal with entries $D_{ii} = \ell''_{\mathrm{SH}}(s_i\mathbf{x}^{(i)\mathsf{T}}\mathbf{w})$.

The Hessian matrix has size $N \times N$, and thus would be far too large to be stored in memory. Fortunately, Equation 6.13 can be solved using a conjugate-gradient procedure, which requires only Hessian-vector products. These can be calculated efficiently through

$$\nabla^2\,\mathrm{R}_D[\mathbf{w}]\boldsymbol{\delta} = 2\boldsymbol{\delta} + c\mathbf{X}^\mathsf{T}\mathbf{D}\mathbf{X}\boldsymbol{\delta}, \tag{6.16}$$

because $\mathbf{X}$ is a very sparse matrix. In practice, Equation 6.13 is only solved approximately, drastically reducing the number of conjugate-gradient iterations, and thus Hessian-vector products, that need to be calculated.

Determining the Step-Size The resulting step vector $\boldsymbol{\delta}$ might be outside the region in which the quadratic approximation (see Equation 6.13) accurately models the true risk landscape $\mathrm{R}_D[\mathbf{w}]$. Therefore, a step-size mechanism is needed usually either by using a trust region or by doing a line search.

Due to the linear nature of the ranking function $r(\mathbf{x}; \mathbf{w}) = \mathbf{x}^\mathsf{T}\mathbf{w}$, the line search can be implemented efficiently by using

$$r(\mathbf{x}; \mathbf{w} + \lambda\boldsymbol{\delta}) = \mathbf{x}^\mathsf{T}\mathbf{w} + \lambda\mathbf{w}^\mathsf{T}\boldsymbol{\delta} \tag{6.17}$$

$$\|\mathbf{w} + \lambda\boldsymbol{\delta}\|_2^2 = \|\mathbf{w}\|_2^2 + 2\lambda\mathbf{w}^\mathsf{T}\boldsymbol{\delta} + \lambda^2\boldsymbol{\delta}^\mathsf{T}\boldsymbol{\delta}. \tag{6.18}$$

By caching the values of $\|\mathbf{w}\|_2^2$, $\mathbf{w}^\mathsf{T}\boldsymbol{\delta}$, $\boldsymbol{\delta}^\mathsf{T}\boldsymbol{\delta}$, $\mathbf{x}^\mathsf{T}\mathbf{w}$ and $\mathbf{x}^\mathsf{T}\boldsymbol{\delta}$, the cost of evaluating the loss for any value of λ after the first evaluation drops to $O(N)$ evaluations of ℓ_{SH} and the corresponding additions and multiplications of the cached values in Equations 6.17 and 6.18.

Implicit Hard-Instance Mining in Hinge Losses When using the squared hinge loss (see Equation 6.11) for ℓ_{SH}, the loss and all its derivatives become zero once the sample is classified correctly with a sufficient margin[3] $s\mathbf{x}^\mathsf{T}\mathbf{w} > 1$. Consequently, the corresponding entries in the diagonal matrix $\mathbf{D}$ in Equation 6.16 become zero. This means that in the product $\mathbf{X}^\mathsf{T}\mathbf{D}\mathbf{X}$, the feature matrix $\mathbf{X}$ can be replaced with a much smaller matrix $\tilde{\mathbf{X}}$ that contains only the examples that are not classified correctly with a margin. Denote with $\mathcal{E} := \{i \in [N] : s_i\mathbf{x}^{(i)\mathsf{T}}\mathbf{w} \le 1\}$ the set of indices of examples with nonzero loss (the *hard* instances), then $\tilde{\mathbf{X}} = [\mathbf{x}^{(i)} : i \in \mathcal{E}]^\mathsf{T}$.

As a consequence, the full feature matrix $\mathbf{X}$ is only needed once per step to determine the gradient $\nabla\,\mathrm{R}_D[\mathbf{w}]$, $\mathbf{D}$, and the hard examples $\mathcal{E}$. Afterwards, each CG iteration only requires the reduced matrix $\tilde{\mathbf{X}}$. This can be interpreted as an implicit hard instance mining step that is performed at the beginning of each step. As the weight vector $\mathbf{w}$ approaches the optimal weights $\mathbf{w}_\star$, most instances will have sufficient margin, and only few hard instances remain, $|\mathcal{E}| \ll N$ (cf. Figure 6.16). Therefore, later iterations require significantly less computation time than earlier ones. In fact, by using an initial vector $\mathbf{w}_0$ for which the hard-example set $\mathcal{A}$ is already small can speed up the overall computation time tremendously, as discussed below.

6.3.4 Further Performance Improvements

Mean-Separating Initialization A simple attempt to improve the initial weight vector is to chose a hyperplane that separates the means of the positive and negative instances for that label. Denote $\mathcal{P} := \{\mathbf{x}^{(i)} : i \in [N], y^{(i)} = 1\}$ and $\bar{\mathbf{x}} := N^{-1}\sum_{i=1}^{N}\mathbf{x}^{(i)}$, then the means are

$$\bar{\mathbf{p}} := \frac{1}{|\mathcal{P}|}\sum_{\mathbf{x}\in\mathcal{P}}\mathbf{x}, \qquad \bar{\mathbf{n}} := \frac{N\bar{\mathbf{x}} - |\mathcal{P}|\bar{\mathbf{p}}}{N - |\mathcal{P}|}. \tag{6.19}$$

3 The margin of an instance denotes how far its score is from the classification boundary. An instance with a margin of 0 is classified correctly, but the slightest perturbation of its features or the classifier's weights could change the classification.

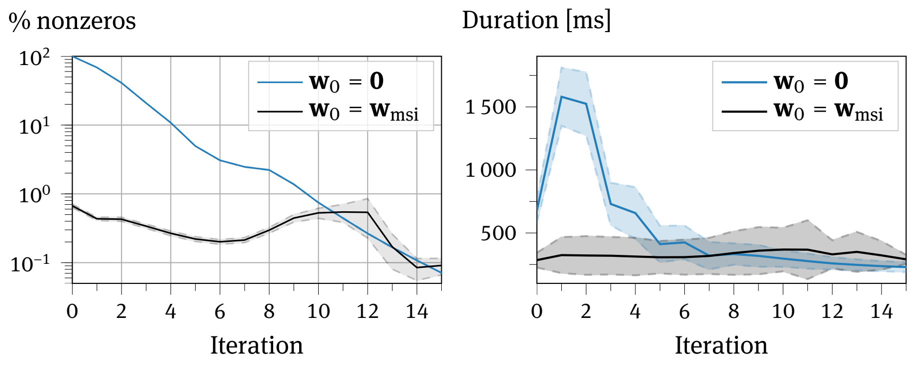


Fig. 6.16: Sparsity of the Hessian calculation $|\mathcal{E}|/N$ (left) and average duration of each optimization iteration (right) over the index of the iteration for zero and mean-separating initialization.

As $\bar{\mathbf{x}}$ only need be calculated once for the dataset, and $\bar{\mathbf{p}}$ can be computed quickly due to the data imbalance $|\mathcal{P}| \ll N$, these values can be computed efficiently.

This procedure can be viewed as an extreme case of data summarization (cf. Chapter 3), in which the entire dataset is reduced to just two instances. The idea is now to solve this very small classification problem, and use its solution as the starting point for solving the full problem. This will be particularly useful if the simple solution already classifies many of the easy negative instances correctly, as the set of hard examples $\mathcal{E}$ will be small in such a case.

For two data points, the classification problem can be solved explicitly, and we call its result the *mean-separating initialization* (`msi`) vector $\mathbf{w}_{\text{msi}}$. This vector is the minimum-norm vector that attains pre-specified margins $\mu_\pm$ for classifying the two data points. As a consequence, it lies in the plane spanned by $\bar{\mathbf{x}}$ and $\bar{\mathbf{p}}$, and is characterized by the following equations:

$$\mathbf{w}_{\text{msi}} = \alpha\bar{\mathbf{x}} + \beta\bar{\mathbf{p}}\,, \tag{6.20}$$

$$\bar{\mathbf{p}}^{\mathsf{T}}\mathbf{w}_{\text{msi}} = \mu_+\,, \qquad \bar{\mathbf{n}}^{\mathsf{T}}\mathbf{w}_{\text{msi}} = \mu_-\,. \tag{6.21}$$

Heuristically, values $\mu_+ = +1$, $\mu_- = -2$ work well, and are based on the rationale that negative samples cover a larger volume in the instance space, and thus the initial decision boundary should be closer to the mean of the positives than to the mean of the negatives.

The efficacy of this method can be seen from Figure 6.16, which evaluates the two initialization strategies on the AmazonCat-13k [478] dataset using an AMD Rome 7H12 CPU.[4] The data shows that

– starting from a zero initial vector, the fraction of nonzeros starts at 100 % and decreases as the training progresses;

4 Computational resources provided by CSC – IT Center for Science, Finland.

- starting from a mean-separating initial vector, the sparsity is already high in the beginning; and
- increased sparsity translates to significant reductions in computation time and corresponding energy savings.

In terms of wall-clock training duration t_{wc}, the speedup that can be achieved by switching from $\mathbf{0}$ to $\mathbf{w}_{\mathrm{msi}}$, defined as $t_{\mathrm{wc}}(\mathbf{0})/t_{\mathrm{wc}}(\mathbf{w}_{\mathrm{msi}})$, lies between 150 % and 500 % as shown in Table 6.5.

Feature-Sorting A large portion of the computation time is spent on calculating the initial margins $\mathbf{X}^{\mathsf{T}}\mathbf{w}$ at the beginning of each iteration. Because $\mathbf{X}$ is a sparse matrix, this computation has low arithmetic density, and because the feature dimension is typically very large, the vector $\mathbf{w}$ does not fit into the L2-cache. These two properties mean that this operation is severely memory-bound.

The caching behavior of $\mathbf{w}$ can be improved significantly by making use of the dataset characteristic – in particular the fact that typical XMC tf-idf datasets have a long-tailed distribution in the feature vector, meaning that some features have a large amount of non-zero entries, but most features have few non-zeros[29]. By sorting the feature indices according to the frequency of their occurrence, the corresponding entries in $\mathbf{w}$ are brought closer together in the address space, thus improving the caching behaviour.

While this has no effect on the scaling of the performance with the thread count, it does induce an absolute speedup, as indicated by the dashed lines in Figure 6.17.

The Memory-Bottleneck On machines with many cores, the performance of the computations presented here is memory-bound. This can be seen in Figure 6.17, where despite the embarrassingly parallel nature of the computations, the performance scales sublinearly once a certain core count is exceeded.

In the specific case of running the computations on a 2-socket AMD Rome 7H12 (64 cores per CPU, cf. Figure 6.15) machine, Figure 6.17 shows almost perfect scaling from 8 threads, corresponding to 1 thread per NUMA node, up to 32 threads, corresponding to one thread per L3 cache. For higher thread counts, the speedup saturates and in some cases more threads may even be disadvantageous to performance.

In addition to the memory bottleneck, there will be a thermal/power bottleneck involved. The used CPU has a base clock of 2.6 GHz, but if only a few cores are used the clock frequency may be increased up to 3.3 GHz.[5] This indicates that even without the memory bottleneck, the expected performance increase of 128 cores over 16 cores would be less than $16\times$. This shows that the sublinear scaling cannot be explained by

5 Given that the computations are memory-bound, even when running with 128 cores the execution ports of the CPU will be idle for a significant amount of time. Only moderate downclocking to ≈ 3.18 GHz occurred in our setup.

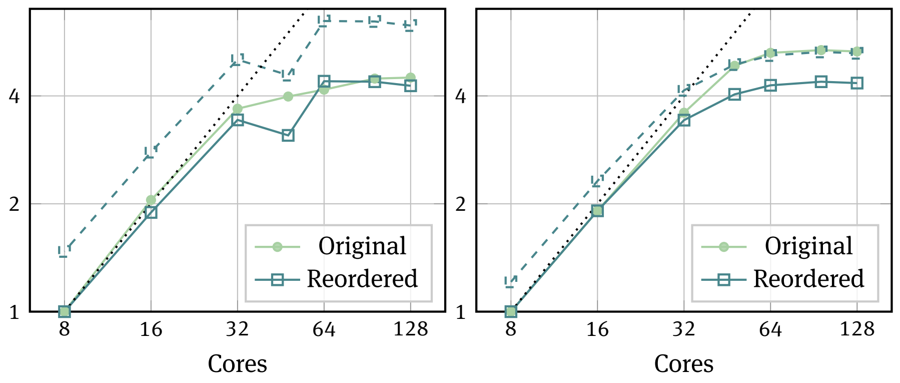


Fig. 6.17: Relative speedup for increasing thread counts using both original and reordered features for the first 10,000 labels of the Wikipedia-500k [50] dataset (left) and for the AmazonCat-13 [478] dataset (right). The dashed line shows the speedup of reordered features, normalized to the computation time with original features. The dotted line indicates perfect scaling. The (non-parallel) portion of the program run-time that is spent parsing the input dataset has been subtracted from the timings presented here.

Tab. 6.5: Training time (in hours) for zero and mean-separating initialization, as well as the number of non-zero weights (NNZ) after pruning (in millions) and their fraction. The experiments were run on a two-socket AMD Rome 7H12 machine.

Dataset	**Zero**	**MSI**	**Speedup**	**NNZ**	**Fraction**
Wiki10-31k	0.05	0.03	164 %	114	3.62 %
Amazoncat-13k	0.33	0.09	353 %	75	2.78 %
Amazoncat-14k	1.31	0.39	339 %	110	1.26 %
WikiTitles-500k	5.34	1.19	451 %	83	0.09 %
Amazon-670k	6.82	1.36	503 %	405	0.44 %
Delicious-200k	7.84	4.73	166 %	1 175	0.73 %
WikiLSHTC-350k	18.52	5.17	358 %	434	0.08 %

Tab. 6.6: Results of DiSMEC in comparison with the state-of-the art results as reported in March 2022 in Bhatia, Dahiya, Jain, Prabhu, and Varma [50], for selected XMC datasets.

Dataset/Metric	DiSMEC	SOTA Method	SOTA
Amazon-670K	—	—	—
P@1	44.7	LightXML	49.1
P@3	39.7	LightXML	43.8
P@5	36.1	LightXML	39.9
AmazonCat-13K	—	—	—
P@1	93.4	LightXML	96.8
P@3	79.1	LightXML	84.0
P@5	64.1	LightXML	68.7
Wikipedia-500K	—	—	—
P@1	70.2	AttentionXML	82.7
P@3	50.6	AttentionXML	63.8
P@5	39.7	AttentionXML	50.4
EURLex-4K	—	—	—
P@1	82.4	APLC-XLNet	87.7
P@3	68.5	APLC-XLNet	74.6
P@5	57.7	APLC-XLNet	62.3

reduced clock frequencies alone; rather, another resource, such as memory bandwidth, is also limiting performance.

6.3.4.1 Comparison With Deep Learning Methods

As shown in Table 6.6, the DiSMEC instantiation of the embarrassingly parallelizable one-vs-rest framework described in Algorithm 4 can be a competitive baseline. Its performance is not significantly worse in comparison to the state-of-the-art deep learning methods, which typically employ transformers encoders [174]. Unlike the deep learning models that require careful hyper-parameter tuning, the linear binary classification underlying DiSMEC is more readily interpretable and well-understood from a theoretical viewpoint. For sparse tf-idf data representation, linear XMC classifiers also work at par with tree-based approaches [371, 584] and those involving dense low-dimensional label embeddings [51, 279].

6.3.5 Summary and Outlook

In this section we presented linear classification algorithms for extreme multi-label classification. The linear model makes the training parallelize perfectly across different labels, though in practice the scaling levels out with too many cores in a single node. This is because even though the training itself does not require any communication or synchronization between the threads, the different cores within a machine still compete for resources such as memory access. By placing a copy of the feature matrix in each NUMA domain, it can be ensured that each CPU can read its data from a part of the memory that it has fastest access to, and the load on the memory interface is spread out across the different NUMA domains. Additionally, by reordering the columns in the sparse feature matrix, data locality and, accordingly, cache efficacy can be improved. An implementation that combines techniques can be found at https://doi.org/10.5281/zenodo.6699587. For further discussion on the interaction between machine learning and the memory hierarchy, see Chapter 7.

The amount of work required to train the linear classifier for the highly imbalanced data typical for XMC can be drastically reduced by starting the weights from a good initialization. One way to find such a weight vector is to reduce the dataset to just two training instances, the centers of masses of the positive and negative training points in the original dataset, which can be calculated efficiently. By initializing the full training procedure with a weight vector that separates this summarized data, one can capitalize on the speedup of the conjugate-gradient optimizer due to implicit hard-instance mining of the hinge loss. This procedure can be seen as a variation of the *sketch-and-solve* principle introduced in Section 3.2. The main difference is that here the solution based on the sketch is used to initialize the full training, and thus no compromise in accuracy is made.

The presentation in the book is focused on the computational and implementation challenges of XMC problems. However, the scale of the label space also leads to interesting statistical consequences such as a long-tailed label distribution and corresponding data-scarcity for tail labels, as well as incomplete training data with missing labels. For a discussion of these issues, see the works of Babbar and Schölkopf [29], Jain, Prabhu, and Varma [336], and Qaraei, Schultheis, Gupta, and Babbar [587].

6.4 Optimization of ML on Modern Multicore Systems

Helena Kotthaus
Peter Marwedel

Abstract: This section demonstrates how the integration of knowledge about underlying hardware platforms and learning algorithms can provide results that would not be feasible by using the knowledge of only one type. In particular, this section presents the optimization of ML algorithms on multicore systems, and in this way addresses the same type of architectures as in Section 6.3. The optimization is based on resource-aware scheduling strategies for parallel machine learning algorithms. The focus is on Model-Based Optimization (MBO), also known as Bayesian optimization, which is an ML algorithm with huge resource demands, including a large number of computational jobs. Execution times of these jobs are estimated in order to enable their scheduling on parallel processors. The section demonstrates that this scheduling enables the processing of larger problem sizes within a given time budget and reduces the end-to-end wall-clock time for a constant problem size.

6.4.1 Motivation

The notion of resource-constrained systems is typically associated with small, integrated, and special-purpose devices exhibiting limitations with respect to, say, computational power, size, or battery life in embedded and cyber-physical systems. However, reducing the understanding of resource restriction to systems of this kind is not sensible. In fact, even high-performance computers and clusters can suffer from resource constraints when solving highly challenging problems that require massive amounts of resources [166, 666]. Therefore, it makes sense to consider resource constraints also for applications typically executed on larger systems.

Here, this is shown for the case of *parallel* MBO. MBO is a state-of-the-art global optimization method for black-box functions that are expensive to evaluate. To reduce the number of necessary evaluations of the black-box function, conventional MBO uses an iteratively refined regression model on a set of already evaluated configurations to approximate the objective function. However, such approaches neglect the heterogeneous resource requirements for evaluating different configurations in the model space, which often leads to inefficient resource utilization. This calls for new resource-aware scheduling strategies to efficiently map configurations to the underlying parallel architecture in accordance with their resource demands. In contrast to classical scheduling problems, the scheduling for MBO needs to interact with the configuration proposal mechanism to select configurations with suitable resource demands for parallel evaluation.

The fundamentals and related approaches of parallel MBO are presented in Section 6.4.2. An overview of the RAMBO (Resource-Aware MBO) framework including the resource-aware scheduling strategies, as well as the corresponding evaluation results on homogeneous multiprocessor cluster systems, is given in Section 6.4.3. Section 6.4.4 proposes a concept for resource-aware scheduling strategies on heterogeneous embedded systems. The results are shown in Section 6.4.5.

6.4.2 Fundamentals and State of the Art for Parallel MBO

In machine learning, selecting the best algorithms for a given optimization problem and simultaneously tuning the corresponding hyperparameters of these algorithms can be very computationally intensive. Many strategies for hyperparameter optimization have been developed. (For an overview see, e.g., [55]). Hyperparameter optimization refers to finding the best configuration θ of a model, e.g., for a prediction problem a model with high predictive performance on an independent test set. When the evaluation of a single configuration already requires high resources, e.g., a very long runtime, then very wasteful optimization methods like evolutionary algorithms are not applicable. A popular approach for algorithm selection is F-racing [448], where a population of configurations is racing against each other and underperforming candidates are iteratively eliminated. This approach also requires many evaluations, at least in the early stage of the algorithm.

An established alternative in the situation of expensive time constraints is Model-Based Optimization (MBO), also known as Bayesian optimization, a state-of-the-art technique for expensive black-box optimization. In this optimization process, an unknown function, say, a machine learning algorithm, is evaluated to find the parameter configuration with the highest quality of the output measured by a given performance criterion within a limited time budget. This process is computationally challenging due to the huge parameter space that needs to be contemplated, and can result in extremely long response times. For this reason it is desirable to reduce the optimization time while maintaining the prediction quality, i.e., $\theta^\star := argmin_{\theta \in \Theta} f(\theta)$ for a search space Θ and an evaluation $f(\theta)$ of the black-box with input $\theta \in \Theta$ [348]. Aiming to reduce the number of evaluations of f required to find the best configuration $\theta^\star$, we used an iteratively refined and updated regression model (surrogate model), which attempts to approximate the black-box function by predicting $f(\theta)$ based on previous evaluations of f. During each iteration, a so-called *infill criterion* (acquisition function) proposes new promising configurations for evaluation.

In its original formulation, the MBO algorithm operates purely sequentially, proposing one configuration to be evaluated after the other [348]. For applications such as hyperparameter tuning for machine learning algorithms or computer simulations, the parallelization of MBO has become of an increasingly interesting approach to reduce the overall execution time [288]. In order to propose multiple points (configurations) simultaneously in a parallel MBO setting, several modifications to the infill criteria or the general technique have been suggested. The modifications result in multiple configurations being proposed in each iteration [54, 254, 328]. The number of simultaneously proposed configurations is typically chosen to match the number of available CPU cores. However, these modifications in general neglect the heterogeneous resource requirements for evaluating different configurations in parallel. Depending on the parameter configuration of the applied machine learning algorithm, resource requirements such as CPU utilization or memory footprint usage can vary heavily [666].

The most important parallel extensions of MBO update the regression model either synchronously or asynchronously. Both variants are based on different infill criteria and have different advantages and drawbacks.

Synchronous Execution To allow for parallelization with a synchronous model update, infill criteria and techniques that propose multiple configurations in each iteration (constant liar, Kriging believer, qEI [254], qLCB [328], MOI-MBO [54]) have been suggested. *Multi-point proposals* are able to derive q configuration proposals $\boldsymbol{x}_1^\star, \dots, \boldsymbol{x}_q^\star$ simultaneously instead of only proposing one configuration $\boldsymbol{x}^\star$ from a *surrogate model*. Here, the model is updated after all evaluations within one iteration are finished. Hutter et al. [328] introduced the qLCB criterion, which is an extension of the single-point LCB criterion using an exponentially distributed random variable to generate q different candidate proposals by drawing random values of $\lambda_j \sim \mathrm{Exp}(\lambda)$ ($j = 1, \dots, q$) from the exponential distribution:

$$\mathrm{qLCB}(\boldsymbol{x}, \lambda_j) = \hat{\mu}(\boldsymbol{x}) - \lambda_j \hat{s}(\boldsymbol{x}) \text{ with } \lambda_j \sim \mathrm{Exp}(\lambda). \tag{6.22}$$

The λ variable guides the exploration-exploitation trade-off. Sampling multiple different λ_j might result in different “good” configurations by varying the impact of the standard deviation term.

Another popular multi-point infill criterion is the qEI criterion [254], which directly optimizes the single-point EI criterion over q points. As the computation of EI uses Monte Carlo sampling, it is quite expensive [136]. Therefore, a less expensive alternative, the *Kriging believer* approach [254], is often chosen. Here, the first configuration is proposed based on the standard single-point EI criterion. Its posterior mean value is treated as a real value of f to refit the surrogate, penalizing the surrounding region with a lower standard deviation for the next point proposal using EI again. This is repeated until q proposals are generated.

The above mentioned multi-point infill criteria can cause inefficient resource utilization when the parallel executed evaluations have heterogeneous execution times. Before new configurations are proposed, the results of all evaluations within one iteration are gathered to update the model. Thus the slowest evaluation becomes the bottleneck, and all other parallel worker processes idle after finishing their evaluation before a new MBO iteration can start. However, performing the model updates only once per MBO iteration also leads to less computation overhead. Varying the execution times of parallel evaluations have already been addressed by Snoek et al. [635], who suggest that these be modelled with an additional surrogate, leading to an “expected improvement per second” favoring less expensive configurations. The resource-aware scheduling strategies for parallel MBO presented in this section also use regression models to estimate resource requirements, but instead of adapting the infill criterion, they use them to guide the scheduling of parallel evaluations. The goal is to guide MBO to interesting regions in a faster and resource-efficient way without directly favoring less expensive configurations.

Asynchronous Execution To avoid CPU idling, asynchronous execution replaces the evaluation of multiple configurations in batches, and the synchronous refitting of the model by refitting the model after each evaluation. Here, the number of worker processes equals the number of available CPU cores, but each worker proposes the next point for evaluation independently, even if configurations $\boldsymbol{x}_{\text{busy}}$ are currently under evaluation on other CPU cores. The main challenge is to avoid evaluations of very similar configurations by modifying the infill criterion to deal with points that are currently under evaluation. The fast Kriging believer approach [254], which is based on EI (also used for multi-point proposals), can be applied to block these regions.

Another approach assessing pending values is the *Expected* EI (EEI) [253, 339, 635]. Here, the unknown value of $f(\boldsymbol{x}_{\text{busy}})$ is integrated out by calculating the expected value of y_{busy} via Monte Carlo sampling, which is, similar to qEI, computationally demanding. For each Monte Carlo iteration, values $y_{1,\text{busy}}, \ldots, y_{\mu,\text{busy}}$ are drawn from the posterior distribution of the surrogate regression model at $\boldsymbol{x}_{1,\text{busy}}, \ldots, \boldsymbol{x}_{\mu,\text{busy}}$, with μ denoting the number of pending evaluations. These values are combined with the set of already known evaluations and used to fit the surrogate model. The EEI can then simply be calculated by averaging the individual expected improvement values, which are formed by each Monte Carlo sample (n_{sim} denotes the number of Monte Carlo iterations):

$$\widehat{\text{EEI}(\boldsymbol{x})} = \frac{1}{n_{\text{sim}}} \sum_{i=1}^{n_{\text{sim}}} \text{EI}_i(\boldsymbol{x}) \tag{6.23}$$

Besides the advantage of an increased CPU utilization, asynchronous execution can also cause additional runtime overhead due to the higher number of model updates and the computational costs for new point proposals, especially when the number of available CPU cores increases. Furthermore, the heterogeneous execution times of job configurations can lead to very similar point proposals due to model updates that are based on similar histories. Instead of using asynchronous execution to efficiently utilize parallel computer architectures, the new approach presented in this section uses the synchronous execution combined with resource-aware scheduling. The next section includes a comparison of this approach (RAMBO) [385, 389, 390, 666] with the synchronous and asynchronous parallel variants of MBO described above.

6.4.3 Resource-Aware Scheduling Strategies

To enable the interaction between resource-aware scheduling strategies and the general MBO process, the RAMBO framework is proposed. Its foundations are based on the `mlrMBO` library [53]. The framework shown in Figure 6.18 aims to resource-efficiently reduce the end-to-end wall clock time needed by parallel MBO, and thus converge to the optimal configuration more rapidly. RAMBO consists of three main steps:

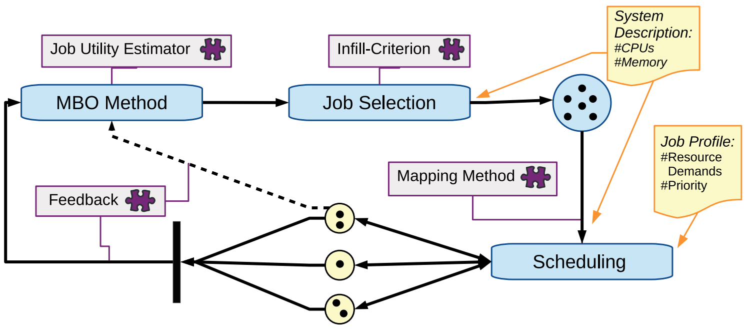


Fig. 6.18: Key steps (shown in blue) in the Resource-Aware Model-Based Optimization Framework [385]: building a regression model, selection of evaluation jobs, and job scheduling. Asynchronous execution (dashed line) and synchronous execution (solid lines) are possible.

First, a previously initialized regression model is built by the *MBO method*. Simultaneously, a *job utility estimator* creates profiles for the evaluations of configurations *(jobs)* by means of an additional regression model. These *job profiles* include runtime estimates, which are used as an input for the respective scheduling strategy later on. In conventional synchronous MBO approaches, such runtime estimates are not available. Hence, the slowest evaluation becomes a bottleneck within one MBO iteration, and the already finished parallel worker processes remain idle. As a consequence, the feedback of all idling processes and hence the model update is delayed.

The second step, i.e., the *job selection*, follows the MBO principles for configuration proposals. Typically, an *infill criterion* such as qLCB in equation (6.22) is used to propose configurations offering a proper compromise between the predicted outputs *(exploit)* and the uncertainty about the search space region *(explore)*, i.e., that have a high potential to optimize the quality of the regression model. To this end, RAMBO provides mechanisms to interact with the job proposal mechanism by postponing or skipping suggested configurations that are deemed to be insufficiently promising or exhibit unsuitable job profiles. As part of this process, a knapsack-based heuristic can be applied to select the most promising and suitable configurations.

Finally, a configurable *scheduling strategy* allocates the jobs to the available resources (system description) according to their particular resource demands. In addition, an execution priority based on the infill criterion is required to ensure that the model is updated i) with the most promising configurations and ii) as soon as possible. This model update follows the synchronous approach, i.e., it is performed when the results of all jobs executed within one MBO iteration are gathered. In a nutshell, the regression model is iteratively updated based on the results of all previous iterations until the runtime budget is exhausted.

Priorities for Job Selection To model the usefulness of a candidate for the objective function, Kriging is used as a surrogate regression model, and qLCB (6.22) is used as a multi-point infill criterion to generate a set of job proposals. Compared with the multi-point proposal qEI [254], the qLCB criterion is more suitable since it is able to propose a set of independent candidates. qLCB can simultaneously generate q candidates by drawing q random values of $\lambda_j \sim \text{Exp}(\lambda)$ $(j = 1, \dots, q)$ from the exponential distribution. Each λ_j results in a different trade-off between exploitation ($\lambda_j \downarrow$) and exploration ($\lambda_j \uparrow$), and thus leads to a different optimal configuration $\boldsymbol{x}_j^\star$ after solving

$$\mathbf{x}_j^\star := \underset{\mathbf{x}}{\operatorname{argmin}} \left[\text{LCB}(\mathbf{x}, \lambda_j)\right] = \underset{\mathbf{x}}{\operatorname{argmin}} \left[\hat{y}(\mathbf{x}) - \lambda_j \hat{s}(\mathbf{x})\right] , \qquad (6.24)$$

where $\hat{y}(\mathbf{x})$ denotes the posterior mean and $\hat{s}(\mathbf{x})$ denotes the root of the posterior standard deviation of the surrogate model at point $\mathbf{x}$.

Since the set of proposed candidates $\boldsymbol{x}_j^\star$ cannot be directly ordered by how promising a candidate is, an additional order is introduced to guide the search for the best candidate towards more promising areas. Therefore, the highest priority is given to the candidate $\boldsymbol{x}_j$ that was proposed using the smallest value of λ_j and is thus closest to the optimum (exploitation). The priority for each job is defined as $p_j := -\lambda_j$.

However, qLCB does not include a penalty for the proximity of selected configurations, which might become a problem if the number of parallel evaluations is high. Therefore, the Euclidean distance is used to reprioritize p_j to $\tilde{p}_j$, encouraging the selection of configurations that are more scattered in the domain space.

First, a set of $q > m$ configurations is sampled from the qLCB criterion. These configurations are then hierarchically clustered by their distance in the domain space of the objective function using the complete linkage method. The procedure starts with the configuration that has previously been assigned the highest priority and assigns it to the first position in the list of selected jobs $\tilde{J}$. For each following step $i \geq 2$, all candidates are split into i clusters according to the hierarchical clustering. Of these i clusters the $i - 1$ clusters that already contain candidates with assigned positions are discarded, leaving one cluster. The position i in $\tilde{J}$ is assigned to the job with the highest priority within this cluster. This goes on until all q candidates have assigned positions.

Thereby an ordering following the hierarchy induced by the clustering is generated. Finally, new priorities $\tilde{p}_j$ are assigned based on the order of $\tilde{J}$, i.e. the first job in $\tilde{J}$ gets the highest priority q and the last job gets the priority 1.

As a result, the set of candidates contains batches of jobs with similar priority, which are spread in the domain space. The priorities serve as input for the scheduling, which groups the q jobs to m CPU cores using the runtime estimates $\hat{t}$.

Resource Utility Estimation The runtime estimates of the set of jobs proposed in each MBO iteration are needed for the scheduling to avoid the execution of jobs with high runtime variances and thus to reduce idling worker processes. This is accomplished by using an additional regression model. As for the MBO algorithm itself, the runtime of a job is predicted in each iteration based on the runtimes of all previously evaluated jobs to build the runtime model of the black-box function. For the model, Kriging is used for homogeneous CPU systems since the runtime is expected to be a continuous function. For parallel architectures with heterogeneous CPUs, Random Forest is used for the model instead. Here, the runtime of a job is estimated for different CPU types (as described in Section 6.4.4). The accuracy of the runtime estimation also influences the scheduling decision. Therefore the runtime estimation quality is also included in the evaluation results.

Knapsack-Based Scheduling Strategy The goal of the knapsack-based scheduling strategy is also to reduce the CPU idle time on the workers while acquiring the feedback of the workers in the shortest possible time to avoid model update delay. Here the qLCB multi-point infill criterion is used to form a set of jobs $J = \{1, \ldots, q\}$ that should be executed on the available CPU cores $K = \{1, \ldots, m\}$. The estimated runtime is given by $\hat{t}_j$ and the corresponding priority by p_j for each job proposal. The time bound for each MBO iteration (deadline) is defined by the runtime of the highest prioritized job. The goal is to maximize the profit, given by the priorities, of parallel job executions within each MBO iteration. To solve this problem, we apply the 0 – 1 multiple knapsack algorithm for global optimization routines [62]. Here, the knapsacks are the available CPU cores and their capacity is the maximally allowed computing time, defined by the runtime of the job with the highest priority. The items are the jobs J, their weights are the estimated runtimes $\hat{t}_j$, and their values are the priorities p_j. Accordingsly, the capacity for each CPU core is $\hat{t}_{j^\star}$, with $j^\star := \operatorname{argmax}_j p_j$. To select the best subset of jobs, the algorithm maximizes the profit Q:

$$Q = \sum_{j \in J} \sum_{k \in K} p_j c_{kj}, \tag{6.25}$$

It is the sum of priorities of the selected jobs, under the restriction of the capacity

$$\hat{t}_{j^\star} \geq \sum_{j \in J} \hat{t}_j c_{kj} \; \forall k \in K \tag{6.26}$$

per CPU. The restriction with the decision variable $c_{kj} \in \{0, 1\}$ s.t.

$$1 \geq \sum_{k \in K} c_{kj} \; \forall j \in J, c_{kj} \in \{0, 1\} \tag{6.27}$$

ensures that a job j is at most mapped to one CPU.

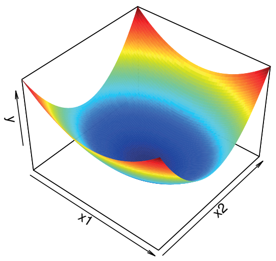
The job with the highest priority defines the time bound (deadline) $\hat{t}_{j^*}$ and is thus mapped to the first CPU core $k = 1$ exclusively, while single jobs with higher execution times are directly discarded (discarded jobs will be proposed again in the next MBO iteration if they are promising enough). Then, the knapsack algorithm is applied to assign the remaining candidates in J to the remaining $m - 1$ CPU cores. This leads to the best subset of J that can be run in parallel, minimizing the delay of the model update. If a CPU core is left without a job, the regression model can be optionally queried for a job with an estimated runtime smaller or equal to $\hat{t}_{j^*}$ to fill the gaps. Jobs having an estimated runtime *shorter* than the deadline, however, can lead to idle times if no other job can be executed within the time remaining until the next model update. The idle time resulting from suboptimal resource usage can be additionally exploited by enabling preemption and migration. More precisely, allowing jobs to be preempted and migrated to other cores provides the opportunity to fill unused time slots within an MBO iteration with high-priority jobs that would be skipped otherwise. Thus a larger set of jobs can be executed. The details of RAMBO's flexible migration mechanisms are described in [389].

Evaluation To evaluate the resource-aware MBO scheduling strategies included in the RAMBO framework, a comparison with different synchronous and asynchronous parallel MBO approaches was performed. The comparison included two asynchronously executed MBO strategies [253, 338] aiming to use all available CPU time to solve the optimization problem in parallel. Both of them used Kriging as a surrogate, with the EEI criterion 6.23 [339] and the Kriging believer [254] criterion. In Kotthaus et al. [390], RAMBO was also compared with a third asynchronous execution strategy, which is included in the SMAC (Sequential Model-based Algorithm Configuration) tool [329], using a random forest surrogate. The results showed that RAMBO and the two other asynchronous execution strategies always converged faster to the optimum compared to SMAC, which is why SMAC is not included in the following presentation. Besides the comparison with the asynchronous strategies, the following presentation also includes two synchronously executed MBO approaches. One of them used the qLCB multi-point infill criterion 6.22 and the other used the qEI criterion [254]. All parallel MBO approaches including the new RAMBO approach were evaluated on a set of established continuous synthetic functions combined with simulated execution times to ensure a fair and disturbance-free environment.

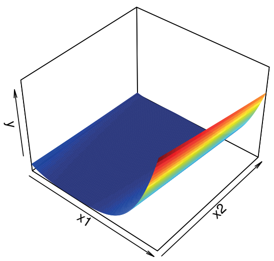
The usage of synthetic functions ruled out technical problems emerging on multi-user systems (swapping, network congestion, CPU cycle stealing, other users occupying fast caches, etc.). Furthermore, synthetic functions eased the evaluation of MBO approaches on different difficulty levels. Two different categories of objective functions (implemented in the R library `smoof` [66]) were considered:

1. Functions with a smooth surface: `rosenbrock`(d) and `bohachevsky`(d) with dimension $d = 2, 5, 10$, which are likely to be fitted well by MBO.
2. Highly multimodal functions: `ackley`(d) and `rastrigin`(d) ($d = 2, 5, 10$), for which MBO is expected to have problems achieving good results.

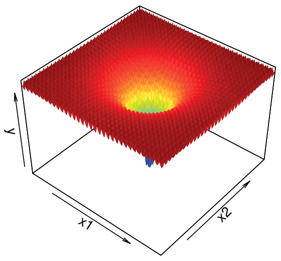
For each objective function, a 2-, 5- and 10-dimensional version were used, each of which was optimized using 4 and 16 CPU cores in parallel to investigate scalability. Figure 6.19 visualizes the synthetic test functions for $d = 2$ [385].



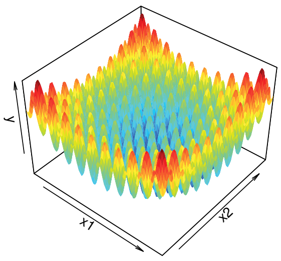
(a) Bohachevsky



(b) Rosenbrock



(c) Ackley



(d) Rastrigin

Fig. 6.19: Synthetic test functions used for the evaluation for $d = 2$. (a) and (b) show a smooth surface; (c) and (d) are highly multimodal [385].

Since synthetic functions are illustrative test functions, they have no significant runtime. Therefore, these functions were also used to simulate different runtime behaviors. For each benchmark two different synthetic functions were combined. One determines the number of seconds it would take to calculate the objective value of the other function. For example, for the combination `rastrigin`(2).`rosenbrock`(2) it would require `rosenbrock`(2)($\boldsymbol{x}$) seconds to retrieve the desired objective value `rastrigin`(2)($\boldsymbol{x}$) for an arbitrary proposed configuration $\boldsymbol{x}$. Technically, the benchmark sleeps `rosenbrock`(2)($\boldsymbol{x}$) seconds before returning the objective value. The runtime was simulated with either `rosenbrock`(d) or `rastrigin`(d), and all combinations of the four objective functions were analyzed except where the objective and the time function were identical. For the unification of the input space, values from the input space of the objective function were mapped to the input space of the function that simulated the runtime behavior. The output of the runtime functions were scaled to return values between 5 minutes to 60 minutes.

To examine how fast the parallel approaches converge to the optima of the benchmark functions within a limited time budget, the distance between the best found configuration at time t and a predefined target value (optimal configuration) was measured. This measurement reflects the accuracy of the receptive MBO approach within the given time budget. To make this measurement comparable for all objective functions, the function values were scaled to [0, 1]. Here, 0 is the target value, defined as the best configuration y reached by any optimization approach within the given time budget. The upper bound 1 is the best y found in the initial set of already evaluated configurations, and is identical for all approaches per given benchmark. Both values were averaged over 10 repetitions. If an optimization needs 2 hours to reach an accuracy of 0.5, this means that within 2 hours half of the way to the best configuration 0 has been accomplished, after starting at 1. The differences between the approaches were compared at the three accuracy levels 0.5, 0.1, and 0.01. The optimizations were repeated 10 times and conducted on $m = 4$ and $m = 16$ CPUs to examine scalability. Time budgets were 4 hours for 4 CPU cores and 2 hours for 16 CPU cores in total, including all computational overhead and CPU idling. All experiments were executed on a Docker Swarm cluster using the R library `batchtools` [415]. The initial set was generated by latin hypercube sampling [481] with $n = 4 \cdot d$ configurations, and all of the following optimizations start with the same initial set in all 10 repetitions:

- `rs`: Random search, served as a base-line.
- `qLCB`: Synchronously executed MBO using qLCB where in each MBO iteration $q = m$ configurations were proposed.
- `ei.bel`: Synchronously executed approach using Kriging believer where in each MBO iteration m configuration were proposed.
- `asyn.ei.bel`: Asynchronously executed MBO using Kriging believer.
- `asyn.eei`: Asynchronously executed MBO using EEI (100 Monte Carlo iterations).
- `rambo`: New synchronously executed MBO using qLCB with job priority refinement and the knapsack-based resource-aware scheduling strategy, $q = 8 \cdot m$ candidates proposed in each iteration.

Optimizations `qLCB` and `ei.bel` are implemented in the R library `mlrMBO` [53]. Optimizations `asyn.eei`, `asyn.ei.bel` and `rambo` are also based on `mlrMBO`. For all MBO approaches, a Kriging model was used from the library `DiceKriging` [605] with a Matern $\frac{5}{2}$-kernel [474] and a nugget effect of $10^{-8} \cdot \mathrm{Var}(\boldsymbol{y})$, where $\boldsymbol{y}$ denotes the vector of all observed function outcomes.

The quality of resource-aware scheduling depends on the accuracy of the resource estimation. Without reliable runtime predictions, the scheduler is unable to optimize for efficient utilization. The runtime for all benchmarks was simulated with either `rosenbrock`(d) or `rastrigin`(d). Figure 6.20 shows an example where the runtime estimation for the `rosenbrock`(5) time function works well (left part). Here, the residual values for the runtime estimation of the evaluated configurations decrease over time. However, the runtime prediction for `rastrigin`(5) (right part) is imprecise. For the 2- and 10-dimensional versions, the results are similar.

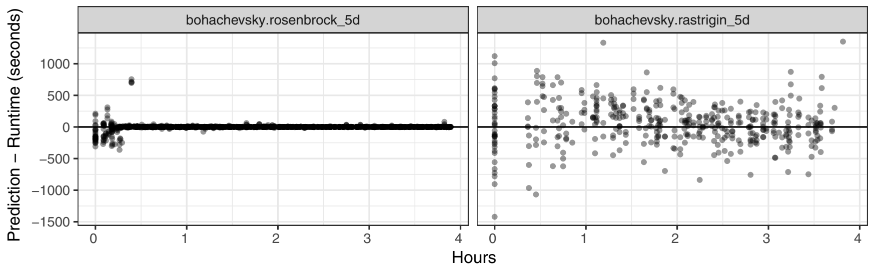


Fig. 6.20: Residuals of the runtime estimation over time for the `rosenbrock`(5) and `rastrigin`(5) time functions on 4 CPU cores combined with `bohachevsky`(5) as the objective function. Positive values indicate an overestimated runtime and negative values indicate an underestimation [385].

This encourages us to consider separate scenarios where runtime estimation has a high quality (`rosenbrock`(·)), and scenarios where runtime estimation is usually error-prone (`rastrigin`(·))s. In the following, we will focus on the scenario with high resource estimation quality. The evaluation results of the scenario with low runtime estimation quality can be found in [385] and are further optimized by a flexible scheduling mechanism [389].

Box plots for the time required to reach the three different accuracy levels in 10 repetitions within a budget of 4 hours on 4 CPU cores are shown in Figure 6.21, and within a budget of 2 hours on 16 CPU cores in Figure 6.22. The faster an approach reaches the desired accuracy level, the lower the box and the better the approach. If an approach was unable to reach an accuracy level within the given time budget, the respective time budget plus a penalty of 1 000 s is entered. Table 6.7 lists the aggregated ranks over all objective functions, grouped by approach, accuracy level, and number of CPU cores. For this computation, the approaches are ranked with regard to their performance for each repetition and benchmark before they are aggregated with the mean. If there are ties in Figures 6.21 and 6.21 (e.g., if an accuracy level was not reached), all values are assigned the worst possible rank. The benchmarks indicate an overall advantage of the new resource-aware MBO algorithm `rambo`. On average, `rambo` is always fastest. `rambo`

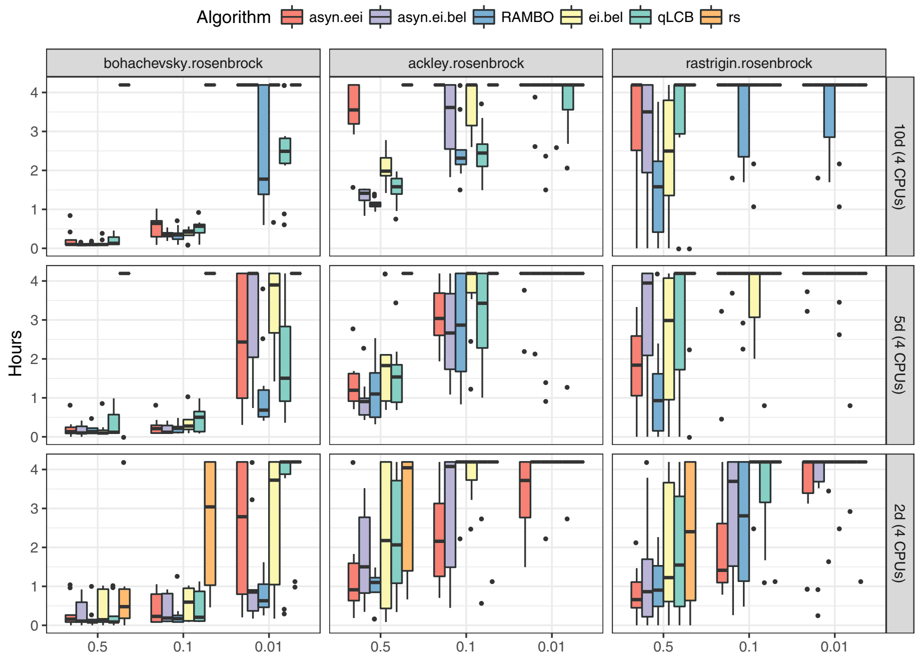


Fig. 6.21: Execution times on 4 cores as a function of the accuracy level for different objective functions using the time function `rosenbrock`(·) [385]. Execution times are low for moderate accuracy levels and favourable for RAMBO (shown in blue).

is closely followed by the asynchronous MBO variant `asyn.ei.bel` for accuracy levels 0.5 and 0.1 on 4 CPU cores but the lead becomes more clear on 16 CPU cores, especially for the highest accuracy level 0.01.

In comparison with the conventional synchronous MBO approaches `ei.bel` and `qLCB`, `rambo`, `asyn.eei`, and `asyn.ei.bel` reach the given accuracy levels in shorter time on 16 CPU cores. This is especially true for objective functions that are highly multimodal and thus hard to model (`ackley`(·), `rastrigin`(·)) by the surrogate, as seen in Figure 6.22.

Table 6.7 shows that the less expensive `asyn.ei.bel` approach performs better than the computationally demanding `asyn.eei` on 16 CPUs. On 4 CPUs the synchronous `qLCB` approach is faster than the asynchronous approaches for the highest accuracy level 0.01. This result is influenced by the good performance of `qLCB` on functions with a smooth surface, as can be seen in Figure 6.21 in the 5– and 10-dimensional version of the `bohachevsky`(·) benchmark. When comparing the performance of the approaches for the 2-dimensional versus the 10-dimensional versions of the benchmarks, Figure 6.22 shows that the `rambo` approach outperforms all other approaches at higher dimensional problems compared with lower dimensions.

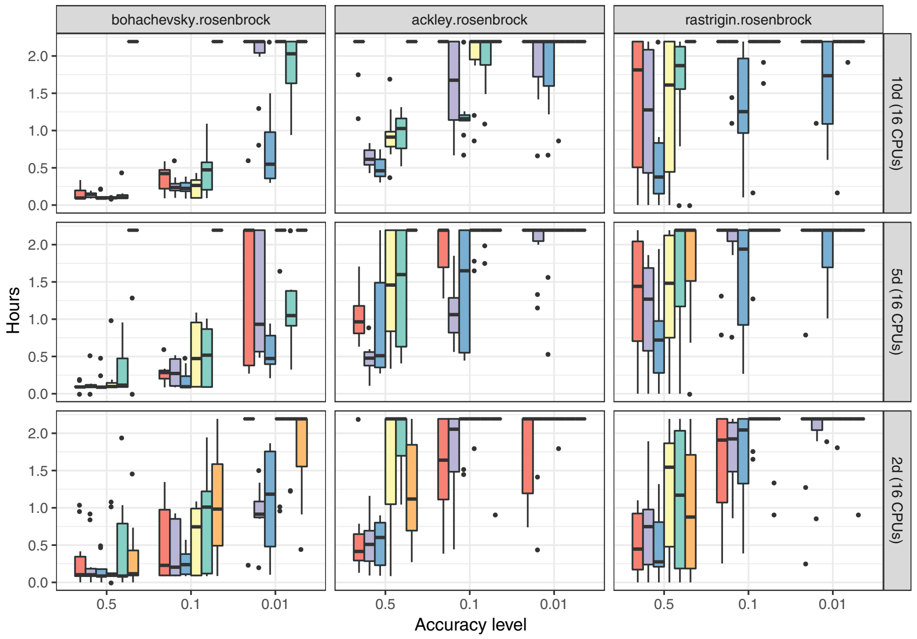


Fig. 6.22: Execution times on 16 cores as a function of the accuracy level for different objective functions using the time function `rosenbrock`(·) [385].

Figure 6.23 exemplarily visualizes the mapping of the parallel configuration evaluations (jobs) for all MBO approaches on 16 CPU cores for the $5d$ versions of the benchmarks. Each gray box represents the execution time of a job on the respective CPU. The gaps represent CPU idle time. For the synchronously executed MBO approaches `rambo`, `qLCB`, and `ei.bel`, the vertical lines represent the end of an MBO iteration. Red boxes indicate that the CPU is busy with a point proposal.

The necessity of a resource estimation for jobs with varying runtimes is obvious because the synchronous variants `qLCB` and `ei.bel` can cause long idle times by queuing jobs together with large runtime differences. The scheduling in `rambo` manages to reduce this idle time. This effect of efficient resource utilization increases with the number of CPUs. `rambo` reaches nearly the same effective resource utilization as the asynchronous approaches and at the same time reaches the accuracy level fastest. The Monte Carlo approach `asyn.eei` generates a high computational overhead as indicated by the red boxes, which reduces the effective number of evaluations. Here, the overhead for a new point proposal sometimes needs the same amount of time as the job evaluation. Idling occurs because the calculation of the EEI is encouraged to wait for ongoing EEI calculations to include their proposals. This overhead also increases with the number of evaluated points. By contrast, `asyn.ei.bel` has comparably low overhead and thus basically no idle time. This seems to be an advantage for `asyn.ei.bel` on 16 CPU cores,

Tab. 6.7: Execution times for accuracy levels 0.5, 0.1, 0.01 averaged over all benchmarks with the rosenbrock(·) time function on 4 and 16 CPU cores with a time budget of 4 hours and 2 hours, respectively [385]. Relative ranks within a column are included in parentheses.

	4 CPUs			**16 CPUs**		
Algorithm	**0.5**	**0.1**	**0.01**	**0.5**	**0.1**	**0.01**
asyn.eei	3.53 (3)	3.91 (3)	4.91 (3)	3.64 (3)	4.30 (3)	5.30 (3)
asyn.ei.bel	3.21 (2)	3.66 (2)	5.04 (4)	2.93 (2)	3.31 (2)	4.48 (2)
rambo	**2.47 (1)**	**3.40 (1)**	**4.23 (1)**	**2.54 (1)**	**2.98 (1)**	**3.72 (1)**
ei.bel	3.64 (4)	4.36 (5)	5.31 (5)	3.81 (4)	4.57 (4)	5.70 (5)
qLCB	4.02 (5)	4.24 (4)	4.83 (2)	4.27 (5)	5.04 (5)	5.40 (4)
rs	5.57 (6)	5.89 (6)	5.89 (6)	5.17 (6)	5.71 (6)	5.82 (6)

where on average it performs better on all accuracy levels than the computationally demanding asyn.eei, especially for higher dimensional problems.

Observations rambo outperforms the conventional synchronous MBO. The resource utilization obtained by the scheduling in rambo leads to faster and better results, especially when it comes to increasing problem dimensions (configurable parameters) and increasing numbers of available CPU cores. On average, rambo converges faster to the optimum than all considered asynchronous approaches. This indicates that the resource utilization obtained by the RAMBO approach improves MBO, especially when the number of available CPU cores increases. Predictable runtimes can be assumed for real applications like hyperparameter optimization for machine learning methods, even if the runtime estimation quality is difficult to determine in advance. The results also suggest that, on some setups, the choice of the infill criterion determines the parallelization strategy for better performance.

6.4.4 Scheduling Strategies for Heterogeneous Architectures

As described in Section 6.4.3, the resource-aware scheduling for MBO uses two inputs: the estimated resource utilization and the priority of the proposed candidates. While the priority of a candidate is computed as described above, the estimation of the resource utilization needs to be enhanced for heterogeneous systems.

Resource Estimation for Heterogeneous Systems The regression model used to estimate the execution times of the candidates was previously based on Kriging; now Random Forest is applied instead. Random Forest is more suitable for heterogeneous systems since the job execution times build up a discontinuous model due to the additional categorical variable that represents the processor type. The regression model now needs to estimate the runtime $\hat{t}_j$ for each candidate in the proposed set of jobs $J =$

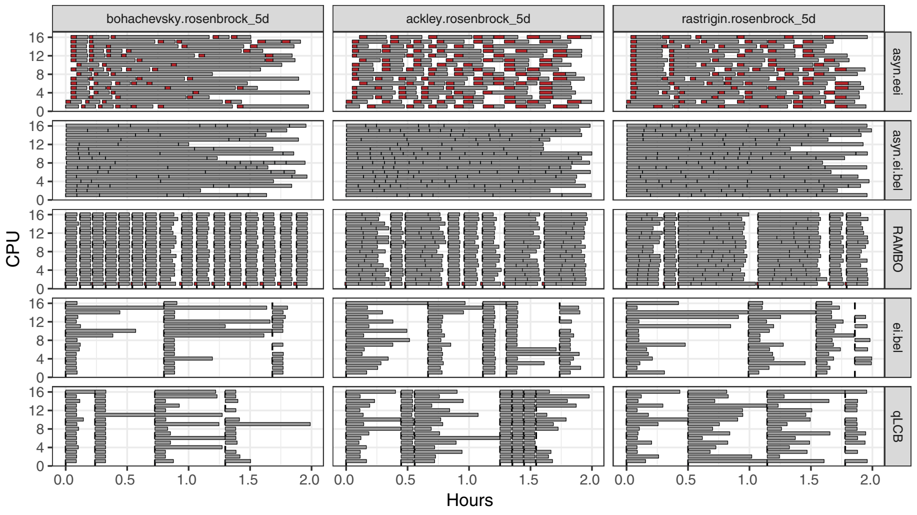


Fig. 6.23: Scheduling of MBO algorithms: Time shown on x-axis and mapping of candidates to $m = 16$ CPU cores on y-axis. Each gray box is a job. Each red box represents the overhead of the point proposal. The gaps represent CPU idle time [385].

$\{1, \ldots, q\}$ and for each available CPU core $K = \{1, \ldots, m\}$. This is required since the execution of a job is processor-dependent. If the underlying heterogeneous architecture is known, the number of runtime estimates per job can be reduced to the number of different processor types. Thus the runtime of a job $j \in J$ is predicted for each available processor type $k \in K$ in each MBO iteration based on the runtime of all previously evaluated jobs to build the runtime model of the black-box function and is therefore denoted as $\hat{t}_{kj}$.

Knapsack-Based Scheduling To apply the $0-1$ multiple knapsack algorithm for scheduling on heterogeneous architectures, the original formulation from Section 6.4.3 needs to be extended. Now the items representing the jobs J have different weights, represented by the different runtime estimates $\hat{t}_{jk}$ per processor type k. Since the capacity of the CPU cores is now heterogeneous, a reformulation is needed. For this purpose, a ratio variable representing an approximated ratio of the runtime differences produced by the different processor types is introduced.

To minimize the delay of the model update with the results of the most promising candidate, the job with the highest priority $j^{\star} := \text{argmax}_j \, p_j$ is now always placed on the CPU core $k^{\star} := \text{argmin}_k \, \hat{t}_{kj^{\star}}$ leading to the shortest estimated runtime for $j^{\star}$. The capacity for the remaining CPUs, and thus the time bound for each MBO iteration, is accordingly defined by the shortest estimated runtime of the highest prioritized job $\hat{t}_{k^{\star}j^{\star}}$. We introduce the ratio variable $\hat{t}_{k^{\star}j^{\star}}/\hat{t}_{kj^{\star}}$ representing the runtime difference of the highest prioritized job on the remaining k CPU cores.

The assumption that runtimes on different CPU types differ by a constant factor goes back to the uniform processor model described by Pinedo, which is a simplified model of real hardware [579]. For example, one CPU might offer vector instructions that some jobs benefit from, whereas others make no use of them. Instead of relying on statically precomputed ratios (such as those derived from the ratio of CPU frequencies), the selected job $j^\star$ is used as the "benchmark" for comparing CPU speeds in a given MBO iteration, under the assumption that in this iteration the speed on CPU k differs from $k^\star$ by a factor of $\hat{t}_{k^\star j^\star}/\hat{t}_{kj^\star}$. The formulation of the restriction of the capacities for the remaining CPU cores is thus as follows, while the rest of the knapsack algorithm remains as described in Section 6.4.3:

$$\hat{t}_{k^\star j^\star}\frac{\hat{t}_{k^\star j^\star}}{\hat{t}_{kj^\star}} \geq \sum_{j\in J}\hat{t}_{k^\star j}c_{kj}\ \forall k \in K. \tag{6.28}$$

Here, the estimated execution times of the remaining candidates on the fastest CPU core $\hat{t}_{k^\star j}$ on the right-hand side of equation 6.28 is expected to be approximately similar to the estimated runtime of a job on the remaining CPU cores $\hat{t}_{kj}$, multiplied by the ratio variable:

$$\hat{t}_{k^\star j} \approx \hat{t}_{kj}\frac{\hat{t}_{k^\star j^\star}}{\hat{t}_{kj^\star}}\ \forall k \in K, \forall j \in J. \tag{6.29}$$

This formulation is needed to reduce the number of weights (number of runtime estimates per CPU type) per item j to a single weight $\hat{t}_{k^\star j}$ in order to apply the original knapsack algorithm.

Evaluation The effectiveness of the heterogeneous RAMBO approach is evaluated by targeting the ARM big.LITTLE architecture[6] of the Odroid-XU3 platform,[7] which is also commonly found in mobile devices. This platform is equipped with four "big" Cortex A15 CPUs (quad-core) with a frequency that can be scaled up to 2.0 GHz and four "little" Cortex A7 CPUs that have about half the processor speed (1.4 GHz). The Odroid-XU3 platform also includes a Mali-T628 GPU (not considered here) and 2 GB of main memory. For the evaluation of RAMBO on heterogeneous processing architectures, not only the runtime that is needed to find the best possible configuration is examined but also the energy consumption. This is accomplished by reading from the power measurement sensors INA231 offered by the Odroid-XU3 platform, which report energy consumption for both processor types as well as for the RAM and the GPU. To measure the energy and power consumption of the resource-aware scheduling strategy and its competing MBO approaches, a so-called Relay Reader [526] is used to read out the sensor data in regular

6 ARM big.LITTLE Technology: https://developer.arm.com/technologies/big-little (accessed Feb. 22nd, 2022).

7 Odroid-XU3: https://developer.arm.com/graphics/development-platforms/odroid-xu3 (accessed Feb. 22nd, 2022).

intervals of approximately one second via threads for both CPU types. These threads are executed on separate CPUs and do not influence the runtime measurements.

The experimental setup consists of a subset of the setup described in Section 6.4.3. RAMBO is compared with the conventional synchronous MBO approach using the qLCB multi-point infill criterion and with the asynchronous MBO approach, which aims to exploit all available CPU time to solve the optimization problem in parallel and using the Kriging believer criterion [254]. All MBO approaches are evaluated on the 2-dimensional versions of the synthetic functions and executed on 4 CPU cores. The runtime of the objective functions was previously simulated by sleeping for a given time, determined via an additional synthetic function that represented the runtime behavior of the respective objective function. For the power consumption measurements, a real computation is needed. This is accomplished by repeatedly executing a function that draws random numbers. The runtime of this real computation is still controlled via an additional synthetic function that defines the number of repetitions and simulates the time that is needed for calculating the objective value. For the synthetic function that simulates the runtime of the objective functions, the `rosenbrock`(d) function is used, since it delivers a more reliable runtime estimation than `rastrigin`(d) (see Figure 6.20). The output of the `rosenbrock`(2) function is scaled to return values from 5 min to 50 min. The MBO approaches run for 2 hours on $m = 4$ CPU cores, and include all computation overhead and CPU idling. The initial set is generated as with the homogeneous experiments by using the latin hypercube sampling [481] with $n = 4 * d$ configurations. All approaches start with the same initial set in all 10 repetitions.

Tab. 6.8: Ranking for accuracy levels 0.5, 0.1, 0.01 averaged over all problems with `rosenbrock`(2) time function on 4 CPU cores with a time budget of 2 hours [385].

Algorithm	**0.5**	**0.1**	**0.01**
`rambo`	**1.90 (1)**	**1.77 (1)**	**1.90 (1)**
`asyn.ei.bel`	2.07 (2)	2.43 (2)	2.63 (2)
`qLCB`	2.67 (3)	2.63 (3)	2.70 (3)

Table 6.8 lists the aggregated ranks over all 2-dimensional objective functions, grouped by accuracy level. As described in Section 6.4.3, the approaches are ranked with regard to their performance for each of the 10 repetitions and for each benchmark before they are aggregated into the mean. Figure 6.24 shows the corresponding box plots for the time required to reach the three different accuracy levels, as described in Section 6.4.3. The faster an approach reaches the desired accuracy level, the lower the displayed box and the better the approach.

The benchmarks indicate an overall advantage of the new knapsack-based algorithm for heterogeneous systems, especially for the highest accuracy level 0.01. On

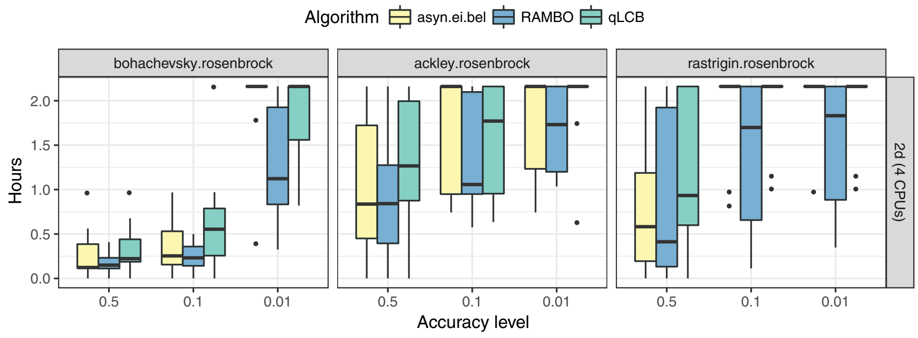


Fig. 6.24: Execution time as a function of the accuracy level for the 2-dimensional objective functions using time function `rosenbrock`(2) (lower is better) [385].

average, `rambo` is always fastest in reaching each of the three accuracy levels, and thus converges faster to the optimum in the time budget of 2 hours. In comparison with `rambo`, the conventional synchronous MBO approach `qLCB` is unable to reach the accuracy level 0.01 for the `rastrigin`(2) and `ackley`(2) functions in all 10 repetitions (see Figure 6.24). The same can be said about the asynchronous MBO approach `asy.ei.bel` for the `bohachevsky`(2) and `rastrigin`(2) functions.

Figure 6.25 shows the box plots for the energy consumption over all 10 repetitions for each benchmark on each CPU type (upper part, Cortex `A7`, and Cortex `A15`) and over all CPUs (lower part, `combined`). Low boxes indicate a small energy consumption. The results indicate that `rambo` consumes more energy than the default `qLCB` approach on the "slow" Cortex `A7` CPU cores, while it consumes less energy on the "fast" Cortex `A15` CPU cores. In comparison with the `asy.ei.bel` approach, `rambo` manages to consume less energy on the "slow" Cortex `A7` CPU cores. The reason for the higher energy consumption of `rambo` compared with the synchronous `qLCB` approach on the "slow" Cortex `A7` cores (see upper part of Figure 6.25) lies in the resource-aware scheduling strategy, which is able to utilize the less energy consuming `A7` CPU cores more efficiently by mapping jobs to specific cores. Furthermore, only jobs with a runtime smaller or equal to the job with the highest priority are executed within one MBO iteration. Accordingly, longer running jobs with a lower optimization potential are discarded and more MBO iterations can be performed in the given time budget. By contrast, `qLCB` is not able to map jobs to specific CPU cores; it starts four jobs on the 4 available CPU cores that were proposed by the infill criterion in each MBO iteration, without respect to the heterogeneity of the underlying architecture and the job execution times.

Another contributing factor to the higher energy consumption of the `qLCB` approach is that it executes more jobs on the more energy-consuming `A15` CPU cores due to the OS scheduling. Within one MBO iteration, the OS scheduler migrates jobs from a "slow" `A7` CPU to a "fast" `A15` CPU for cases where a job on a fast CPU finishes earlier than a job on a slow CPU. This speeds up computation and thus executes more MBO-

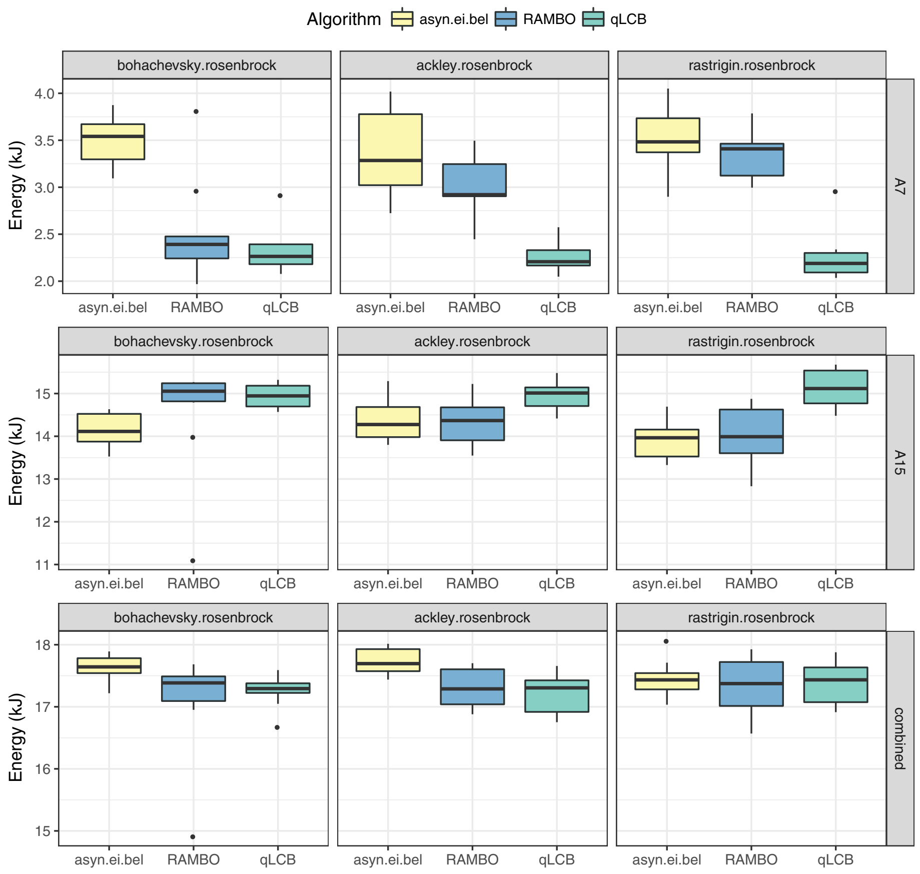


Fig. 6.25: Energy consumption in kJ on the two A15 CPUs (2.0 GHz), the two A7 CPUs (1.4 GHz), and combined consumption on both CPU types across all 10 repetitions for each objective function, with `rosenbrock`(2) time function and a time budget of 2 hours (lower is better) [385].

iterations. Hence, `qLCB` has nearly no idle time on the `A15` CPU cores. However, the conventional synchronous approach only performs approximately half as many MBO iterations as `rambo`. In general, `rambo` executed more job evaluations in the given time than both competing MBO approaches. However, the `combined` energy consumption on all four CPU cores depicted in the lower part of Figure 6.25 shows that `rambo` consumes approximately the same amount of energy as `qLCB`, while it consumes less energy than `asy.ei.bel` for the `bohachevsky`(2) and `ackley`(2) benchmark functions.

The asynchronous `asy.ei.bel` approach in most cases consumes more energy than `rambo` since it has nearly no CPU idle time. However, it still converges more slowly to the optimum. The reason for this is that `rambo` selects more promising candidates with shorter runtimes since it executes only jobs with a runtime shorter than or equal to the most promising candidate, and thus aims to find the cheapest way of evaluations through the model.

Overall, the results show that the resource utilization obtained by the scheduling for heterogeneous architectures in `rambo` enables MBO to converge faster to the optimum without consuming more energy resources than the competing approaches.

6.4.5 Summary: Resource-Aware Scheduling for ML on Multicores

We presented resource-aware scheduling strategies for parallel machine learning algorithms on multicore systems. The resource-aware model-based optimization framework RAMBO was introduced and evaluated. RAMBO can fully use the potential of parallel architectures. This was accomplished with an estimation model for the runtimes of each evaluation of a black-box function to guide the scheduling of configurations to available resources. In addition, an execution priority reflecting the estimated profit of a black-box evaluation was used to guide MBO to interesting regions in a faster, resource-efficient way without directly favoring less expensive configurations. The evaluation results showed that RAMBO converged faster to the optimum than the existing parallel approaches. RAMBO was especially efficient for complex high-dimensional problems, and strongly improved upon the existing approaches in terms of scalability when the number of available CPU cores was increased. Overall, it was shown that the integration of knowledge from the theory of using the underlying hardware (like scheduling) with knowledge about machine learning algorithms achieved results that would not have been feasible without crossing the boundaries of traditional knowledge areas.

6.4.6 Conclusion

The advantage of linking information on underlying hardware platforms with algorithmic knowledge is assumed to exist not only in this particular case. Lowering the walls between disciplines is likely to provide benefits in other cases as well.

7 Memory Awareness

Due to the involvement of massive data and the growing size of trained models, most machine learning techniques are memory intensive. As one of essential components in the von Neumann architectures widely used nowadays, memory is a well-known bottleneck on the execution time, particularly due to the “Memory Wall” problem. That is to say, the access time of memory is way larger than the processor cycle time. In addition, the energy and power consumption required by the memory are known to be significant in the overall system. On embedded systems, which is the focus of this book, such design constraints are amplified and impose great challenges for machine learning techniques. Although the emerging non-volatile memories appear to be promising because of their attractive features, e.g., low leakage power, high density, and low unit costs, they also bring up new design constraints like higher error rates, which might degrade the performance of machine learning techniques. To this end, several optimization and architecture-aware approaches have been proposed to improve the usage of memory and enhance the reliability of learning algorithms.

In this chapter, several techniques are briefly introduced to tackle some of the aforementioned issues related to memory. By leveraging the application-specific knowledge, we demonstrate that the memory footprint can be effectively reduced (see Section 7.1). Given learning models, we can further optimize the memory layout proactively in the model implementation to favor the underlying cache memories with a probabilistic perspective (see Section 7.3). Last but not the least, learning models can be reliable with unreliable memories if we take bit errors into account during the training phase (see Section 7.2). Overall, this chapter tends to suggest that the design constraints of underlying memory can be handled in a post-optimization fashion, within the implementation of learning models, or even earlier at the training phase. The insights presented in this chapter should remain highly relevant in upcoming years, and become more important for future applications along with emerging memory technologies and their new design constraints.

https://doi.org/10.1515/9783110785944-007

7.1 Efficient Memory Footprint Reduction

Helena Kotthaus
Peter Marwedel

Abstract: This section discusses optimization approaches for the efficient memory footprint reduction of machine learning algorithms that are written in the GNU R programming language. The presented optimization strategies target the memory management layer between the R interpreter and the operating system and reduce the memory overhead for large data structures by ensuring that memory will only be allocated for memory pages that are definitely required. The proposed approaches use additional information from the runtime environment, e.g., the short-term usage pattern of a memory block, to guide optimization. The evaluation is based on statistical machine learning algorithms. When the memory consumption hits the point that the OS starts to swap out memory, optimization strategies are able to speed up computation by several orders of magnitude.

7.1.1 Motivation

In order to execute machine learning algorithms on resource-constrained devices, it is important to make efficient use of the available resources. These resources include processors (including runtime), memories, communication bandwidth, and energy. This book includes sample optimization algorithms aiming to achieve resource efficiency. In particular, Chapters 6 to 9 present such sample optimizations. The current section demonstrates the optimization potential memories as resources. Ideally, memories have an infinite capacity, but their size can have a relevant impact on the applicability of certain techniques. This is especially true for resource-constrained embedded systems. The current section focuses on the efficient use of memories for machine learning algorithms written in the R language. The R language is used for many machine learning applications and, therefore, it is considered here. As shown in [387, 503], the R environment has several drawbacks leading to slow and memory-inefficient program execution. In R programs, most data structures are vectors. When the size of a vector is above a certain threshold, the R interpreter allocates a large vector. For each large vector, a dedicated block of memory is allocated, potentially spanning multiple pages. These pages, even when unused, take up memory. When the amount of memory required for computations exceeds the physical memory available to the application, the execution is drastically slowed by frequent page swaps that require I/O, a phenomenon also known as “thrashing”. The performance penalty due to thrashing might render complex computations entirely infeasible.

The current contribution is based on the work of Kotthaus et al. [383, 385, 386]. Section 7.1.2 provides a survey of related work and explains the fundamentals of R's memory management. Section 7.1.3 discusses the page-sharing strategies for efficient memory utilization of R machine learning algorithms. Finally, Section 7.1.4 presents the evaluation results and concludes with a summary.

7.1.2 Related Work and Fundamentals: Memory Footprint Reduction and the R Environment

Related Work - Memory Footprint Reduction The memory optimizations presented in Section 7.1.3 work on a layer between the R interpreter environment and the OS. This enables the optimization of arbitrary R applications, especially memory-hungry machine learning applications, with only small modifications to the R interpreter and without requiring application changes. Thus in the following, the related system-level approaches for reducing memory utilization will be discussed.

In general, related work on utilizing main memory more efficiently can be categorized into two groups: memory compression approaches, often influenced by embedded systems resource constraints, and memory deduplication, which is mostly used in virtualization.

Memory compression tries to reduce the swapping activity of a system by compressing memory contents instead of swapping pages to the secondary storage. Compression approaches share the drawback that a significant amount of processor time is spent on compressing and decompressing memory contents.

By contrast, memory deduplication reduces the memory overhead by mapping virtual pages with identical contents to a single physical page. This is often beneficial in virtualized environments where large amounts of read-only memory, such as shared libraries, are used in multiple virtual machines [626]. However, deduplication can introduce significant computational overhead, since the contents of pages have to be scanned periodically in order to identify pages with identical content. An often used implementation of deduplication that has been the subject of multiple improvements is available in Linux as the *Kernel Samepage Merging* (KSM) [22]. KSM has also been optimized in [133] by introducing a classification scheme based on access characteristics, comparing only pages within the same class to reduce the overhead of page scanning. A memory trace-based evaluation of different deduplication and compression approaches is presented by Deng et al. [169], showing that deduplication yields better results than memory compression.

Sharing memory pages within a single process appears to be a rarely-used concept: on Linux, it is automatically used to map a set of newly allocated virtual pages to a single physical page filled with null bytes. This can cause performance issues in high-performance environments since each write to any newly allocated page will trigger a page fault. Here, an enhancement by Valat et al. [678] was proposed that avoids unnecessary page removal when the application knows that it will overwrite a page in the near future. A language-level version of this *copy-on-write* technique, operating on objects instead of memory pages, is sometimes implemented using reference counters [665]. The R language also implements a copy-on-write scheme. Here, the complete object (potentially spanning multiple pages) is copied when it is modified, resulting in page duplications for partial modifications.

OS level optimizations lack knowledge about the specific memory behavior of the runtime environment. Although some information can be used to improve the time needed to detect duplicates, the application-specific knowledge of why the data was copied in the first place is ignored. By contrast, the memory optimization presented in Section 7.1.3 employs specific knowledge about the interpreter state to reduce the number of pages that need to be scanned for identical content and *proactively* avoids the main sources of identical-content pages from object allocation and duplication by optimizing the copy-on-write mechanism for partial object modification.

Fundamentals – The R Environment The *lifecycle of an object*, (e.g., a vector data structure) in the R runtime environment starts with its allocation. In R, vectors are assumed to consist of a contiguous block of (virtual) memory. Depending on the size of the object, the R interpreter uses a system of multiple memory pools for vector objects with a data size of up to 128 B. For larger vectors, memory is allocated directly via the malloc C library function instead of pooling the allocations. This reduces the memory fragmentation when many small objects are created and some of them are released. The R language does not require the programmer to explicitly manage memory; a garbage collection is needed to automatically free memory. The garbage collector in R is a mark-and-sweep, non-moving, generational collector. It can be manually triggered, but it also starts automatically when the interpreter is in danger of running out of heap space.

The R interpreter ensures that an allocated object is always initialized—either by explicit initialization or implicitly by writing the results of a computation to it. After the object is allocated and initialized, it can be used as input for various R functions such as the plus operator. The fact that functions can modify an object, in conjunction with R implementing *call-by-value* semantics, means that objects need to be copied when being passed to a function. However, at this point a copy-on-write optimization is triggered: copying an object is done by merely sharing the reference; the actual copy is delayed until the object is modified (if at all). The interpreter now has two references to the same object, which may be modified later. When this modification happens, the copy process is triggered and a full copy of the affected object, potentially spanning multiple pages, is created using the interpreter-internal *duplicate* function. This is illustrated in Figure 7.1.

On the left-hand side, a large R vector object consisting of a header *H* and four pages *A* to *D* is shown both in *virtual memory* on the top (marked with dotted lines) and its corresponding allocated *physical memory* on the bottom (solid lines). On the right-hand

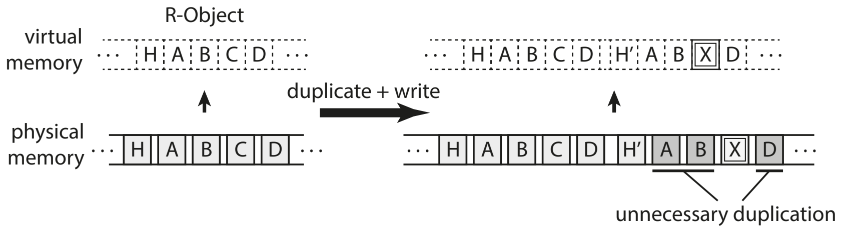


Fig. 7.1: Example of the copy-on-write mechanism in the R interpreter. R copies (duplicates) at object level instead of page level granularity [385].

side, the situation after a duplication that was triggered by a write access is shown. Now there are two R objects, shown in the virtual memory on top and their corresponding physical memory on the bottom. In one of the copies, page *C* was modified and is now marked as *X*, and the copy has its own header *H'*. Although unmodified, the R interpreter needs to use additional memory to create duplicates of pages *A*, *B*, and *D* (marked in gray) since it assumes that objects are organized as contiguous blocks of memory and thus it has to duplicate at *object-level granularity*.

The memory optimization presented in this contribution has the goal of reducing this memory overhead by copying only parts of the object, sharing the same memory pages between multiple objects as long as they are not modified. This scheme is transparent to the interpreter's memory management including the garbage collection, requiring only small changes in memory allocation and freeing, as well as in the duplicate function. This optimization will be presented in the next section.

7.1.3 Memory Footprint Reduction via Page Sharing Strategies

Different optimization strategies are combined for the efficient memory footprint reduction of machine learning algorithms implemented in the R language. The first strategy that proactively *avoids the duplication* of memory pages is based on optimizing the allocation and duplication mechanisms of the R interpreter. This approach is further refined by a second strategy using *static annotations* to reduce the optimization overhead and by *dynamic refinement* using a page content analysis for page deduplication to increase the amount of shared memory.

Page Duplication Avoidance As shown in the previous section, the R interpreter can only allocate complete objects that potentially span multiple pages. The first part of the memory optimization is based on the object allocation mechanism of R. To enable the allocation and thus the sharing of memory at *page-level granularity* instead of object granularity, a custom memory allocator is employed when a large vector has to be allocated, as shown in Figure 7.2. When the internal function of the R interpreter *allocVector* is called to allocate a large vector, the optimized interpreter selects between the *custom allocator* to share memory on page granularity or the *default malloc* function if this is not required. In both cases, the allocated memory is accessible within the address space of the R interpreter. The custom allocator uses a memory management scheme

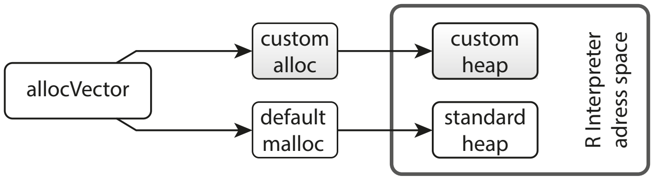


Fig. 7.2: Memory allocation scheme for dynamic page sharing [385].

similar to standard virtual memory schemes commonly used in Operating System (OS) kernels. For ease of implementation, it is completely implemented in the user space. The downside of such a user-space implementation is that it needs to replicate certain data structures that are already present in the OS (e.g., for mapping virtual to physical memory) because those OS kernel data structures are not sufficiently exposed to user space. This replication could be avoided by implementing the optimization in the Operating system kernel (cf. [383]), but this is significantly more invasive and not applicable in many environments where the user has no control over the Operating system kernel. Since the user space has no direct access to physical memory, a single file located on a RAM disk (see *custom heap* in Figure 7.2) is used.

The allocation of physical memory from this file is realized via a simple free-bitmap based allocator. The file can be dynamically enlarged if needed. Mapping physical pages into the virtual address space of the R interpreter can be accomplished by utilizing the mmap Unix system call. For changing the access permissions of these physical pages, the mprotect system call that modifies the settings of the memory management unit of the processor is employed. Besides these system calls, an additional page table is needed to enable the mapping from a virtual address to a physical address. For simplicity reasons a hierarchical page table with the same four-level structure as used by the processor is implemented. To enable the sharing of pages, the user space memory management system needs to map the same physical page to multiple locations in virtual memory. Therefore, a reference counter is required for each physical page. A reference counter greater than 1 indicates that the page is shared between multiple objects or multiple times within one object.

To avoid the zero-initialization of allocated large vector objects, a *global shared zeroed page* is utilized. This also ensures that memory is only allocated for pages that are actually written to at a later time. Figure 7.3 illustrates an example for this optimized R object allocation. Here, the custom memory allocator was asked to allocate an object with a total size of five pages. While the object has the requested size of five pages in

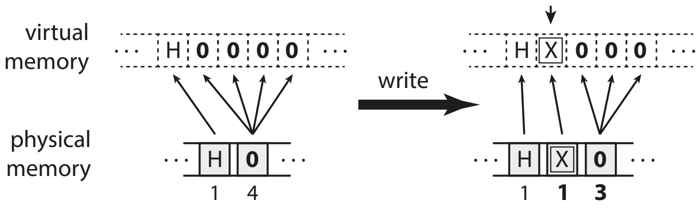


Fig. 7.3: Optimized object allocation via sharing a global zeroed page [385].

virtual memory (dotted, left upper part), physically it only consists of two pages (left lower part). Those two pages are a single non-shared page, marked with *H* for header in the beginning, followed by a shared page, marked with ***0***, called the global zeroed page. The numbers in small print below the physical pages are the *reference counters*. The zeroed page has a reference counter of 4 since it is shared four times within the allocated object (mapped four times into virtual memory). The shared zeroed page is filled with zero-bytes. The concept of prepared zeroed pages is already implemented in OS kernels. However, the standard R interpreter does not benefit from this concept since it immediately writes to all memory that it allocates for initialization. The non-shared initial page *H* is required as it will contain not just data but also the object header. The R interpreter writes this object header to the front of the allocation area. Since it will be updated frequently (e.g., during garbage collection), it is not shared between multiple objects. Since the header page *H* is mapped only once, its reference count is 1.

The R interpreter now has the illusion that it has allocated five pages of memory, even though only two pages are allocated physically. To sustain this illusion, the optimized allocation mechanism has to ensure that any write access to a virtual page that points to a shared physical page can be detected and handled. If such a write access is not handled correctly, it affects not only the intended virtual page but also all virtual addresses where the same physical page is shared. Therefore, all pages with a reference counter greater than 1 are marked as *read-only*, ensuring that a write access triggers a segmentation fault. This fault is caught by a *signal handler* that performs the unsharing of the affected page. To determine the affected physical page the handler uses the virtual address of the write access. It then allocates a new page, copies the contents of the original page to it, and replaces the page that caused the segmentation fault with the new one. The resulting situation is shown on the right side of Figure 7.3: one of the instances of the zeroed page that was written to was replaced with a new page marked with *X*. This updates the reference count of both the zeroed page and the newly allocated page. Since the new page is only mapped once, it can now be marked as read-write so that further access no longer requires special handling.

As noted, the R interpreter can only copy on the object level. Thus, if an object consists of multiple pages, parts of the copy may end up with the same content as the original (see Figure 7.1). To avoid this, the *duplicate mechanism* of the interpreter is optimized by improving the granularity of the copy from object level to page level.

While the allocation optimization avoids the immediate allocation of pages by using the global zeroed page, the duplicate optimization allows the reuse of already-allocated pages of the original object instead of allocating new pages. An example of the duplicate optimization is shown in Figure 7.4.

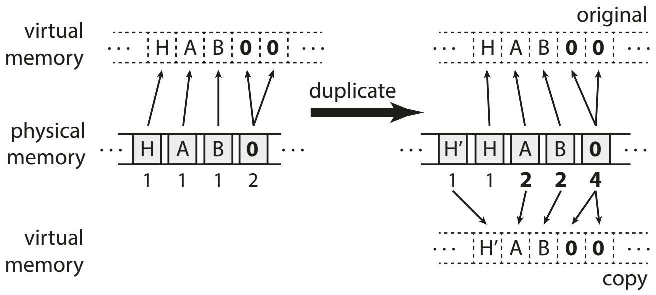


Fig. 7.4: Optimized copy mechanism on page-level instead of object-level granularity via page sharing [385].

The left side shows the situation before the duplication is shown: an object occupies five virtual pages, two of which reference the global zeroed page. Unlike the original R interpreter that would need to allocate five new pages for the copy of this object, the optimized version reduces this to a single allocated physical page. This is shown on the right side with the original object at the top and its copy at the bottom. Here, a *virtual-only copy* of the first page that contains the object header is not created, since the header of the copy is updated immediately by the R interpreter after the duplication. This would otherwise trigger an unsharing of this page. To avoid the overhead of this event, the optimized duplication immediately creates a physical copy of the header page. Most of the pages of the original object are now mapped twice in virtual memory and their reference counters are updated. Both the original and copy are marked as read-only to allow for unsharing on write access.

Overall, the finer copy granularity of the optimization enables storing both the original and copied objects from the example in just five pages of memory. By contrast, the original R interpreter would need ten pages of memory to store the same objects. Although the mechanisms of sharing pages during allocation and duplication described above always result in a valid view on memory for the interpreter, there are cases where additional overhead is caused that can be avoided by further refinements described in the next subsection.

Static Refinement via Annotations To reduce the runtime overhead caused by proactively avoiding page duplications, a static refinement consisting of two kinds of annotations is applied. The first annotation is based on the expected utilization of an object immediately after allocation and the second annotation is based on the size of the allocated object.

The optimized memory allocation (see Figure 7.3) reduces the memory footprint by using a global zeroed page, assuming that not all pages of the allocated object will be written to immediately. However, this assumption is not always valid. For instance, (built-in) vector arithmetic functions in the R interpreter allocate a new object and immediately write to all pages of it to store their results. This would cause a segmentation fault for the first write of every page, triggering the memory allocation for all pages of the object. These segmentation faults cause runtime overhead that would not occur when allocating an object with non-shared pages.

To avoid this overhead, *annotations* are placed in the C source code of the R interpreter built-in functions where newly allocated memory is completely overwritten directly after allocation. Here, the custom allocator returns an object where every virtual page references a new physical page, so no segmentation faults will be triggered by write accesses. Although these R objects do not save memory on allocation, they still have the opportunity for later optimizations, e.g., when they are duplicated. Currently, the annotations for these *"full-overwrite"* functions need to be manually placed in the R interpreter's C source code by locating calls to *allocVector*, followed by loop structures that write to every element of the newly-allocated object. Those manually placed annotations could also be automated by a static code analysis checking for allocation calls followed by loops writing to the newly-allocated object.

The second annotation for reducing the runtime overhead incurred by the optimization relates to the size of the allocated object. The R interpreter can allocate objects with a variety of sizes, not all of which span multiple pages. The optimized custom allocator is therefore enabled only for object sizes that indicate a potential for page sharing. Here, the potential is limited for smaller objects. The first page of an object stores not just data but also the frequently modified object header that is therefore never shared. Thus R objects smaller than two pages of memory are passed to the standard, non-sharing memory allocator. This size limit could also be used as a tunable parameter to select a trade-off between memory savings and runtime overhead.

Dynamic Refinement via Page Contents In addition to the above-described static refinements, an additional dynamic refinement for increasing the number of shared pages is applied. During the execution of an R program, allocated objects are updated with the results of calculations. Those updates can result in multiple distinct pages with the same contents, which opens up the opportunity for sharing those pages. The general idea of locating identical objects in a system and saving memory footprint by reducing them to a single object is known as deduplication.

The memory optimization employs a restricted version of locating identical contents. Here, the content scan only checks for pages identical to the already existing global zeroed page. The *deduplication of zeroed pages* is illustrated in Figure 7.5. On the

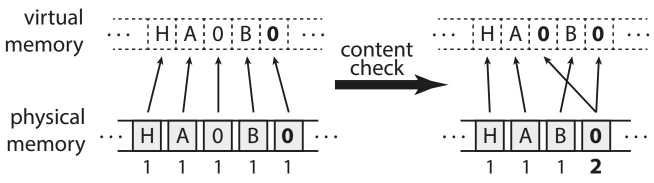


Fig. 7.5: Deduplication optimization for zeroed pages [385].

left side, the situation before the page content scan is shown where an object contains two identical zero pages. One of those pages is already mapped to the global zeroed page (shown in bold), while the other uses a separate physical page. On the right side, the situation after deduplication is shown. Here, the *content check* has detected the separate copy and mapped its virtual page to the global zeroed page, freeing the memory used for the unnecessary duplicate.

Although a general scan that is able to detect duplicated pages with arbitrary content could be applied, such a scan would incur a significant runtime overhead (e.g., due to the calculation of hash values) compared to scanning just for zeroed pages. While a scan for zeroed pages can use an early abort condition as soon as a non-zero element is found, a scan for arbitrary content would need to check the full content of all pages in the system. The overhead incurred by deduplication of zeroed pages is influenced by the frequency of the content check and by the number of pages that need to be scanned. To reduce this overhead, the scan is only activated after the completion of a garbage collection in the interpreter. This avoids scanning the pages that would soon be discarded and also provides a natural regulation mechanism for the frequency of content checks, as the frequency of garbage collection depends on the memory requirements of the executed program.

With the deduplication optimization, pages that were previously excluded from sharing the global zeroed page, in arithmetic vector operations, say, can now be dynamically shared. Thus, both the static and the dynamic refinements of the memory optimization complement each other. Details on the interaction of the refinement strategies and the general page duplication avoidance strategy can be found in a separate publication [384].

7.1.4 Evaluation: Memory Footprint Reduction Strategies

The results obtained by applying the proposed memory optimization strategies for R to real-world machine learning benchmarks are presented in this section. Both, the evaluation results related to the memory consumption and the runtime effects of the page sharing optimization strategies will be discussed.

Experimental Setup For the following experiments, a computer equipped with a 2.67 GHz Intel Core i5 M480 CPU and 6 GB of RAM, using a 64-bit version of Debian Linux 7.0 as the operating system is used. On this system, memory pages have a size of 4096 bytes. Although a larger page size than the system page size could be used for the memory optimization, the same size was chosen as it is expected to maximize the amount of memory that can be shared (using a smaller page size than the system size is inefficient since the optimization relies on the hardware Memory Management Unit (MMU) for efficient page access protection). To evaluate the proposed memory optimization approach, the memory usage and runtime of the R interpreter including the described optimizations is compared to the standard GNU R interpreter. Both the standard as well as the optimized interpreter are compiled using GCC version 4.7.2 with the default flags (-O2) selected by the build system of R version 3.1.0.

The standard memory measurement functions for user space functions in Linux measure only the virtual memory of a process. Since the optimization approach maps the same physical page multiple times into virtual memory, these functions are not sufficient for the evaluation. They are not able to measure the amount of physical memory saved since they only count every virtual instance of a shared physical page. Therefore, a separate memory measurement function was created. To measure the amount of memory allocated by the default allocator, the standard allocation functions such as malloc are overwritten with versions that track the current total amount of memory allocated and the original functions are called afterwards. For the optimized custom allocator, the number of physical pages that need to be reserved is directly tracked along with the size of the memory management data structures. With these mechanisms, the allocated physical memory can be measured accurately.

For the evaluation of the optimization, two different benchmark sets are applied. The first set is a shorter-running set of benchmarks, selected from the R benchmark 2.5 suite [274], which was originally developed to measure the performance of various configurations of the R interpreter (in the following denoted by GU) plus one additional benchmark, as listed in Table 7.1. The R benchmark 2.5 suite was also applied in other optimization approaches that focus on dynamic compilation for R [353]. To analyze if the memory optimization is also beneficial for algorithms that already try to reduce the memory footprint by using application-specific knowledge, the additional benchmark *glmnet* is included. This benchmark utilizes an existing sparse matrix optimization implemented as an R package. For accurate measurements, the iteration counts for the outer loop of each benchmark were scaled to result in a runtime of approximately 1 minute with the standard R interpreter.

The second set of benchmarks is based on a set of publicly available long-running real-world machine learning benchmarks [384], listed in Table 7.2. The choice of these classification algorithms is based on the method's popularity and the availability of its implementation. The default parameters or, if available, the implementation's internal auto-tuning process was used to configure the algorithm parameters. The input dataset is a 2-class classification problem and has a sufficiently large number of observations

Tab. 7.1: Misc Benchmark Set.

Benchmark	Description
GU/08a-1	Linear regression over a 3000 × 3000 matrix
GU/08a-2	FFT of 2 400 000 random values
GU/08a-3	Inverse of a 1600 × 1600 random matrix
GU/08a-4	Greatest common divisors of 400 000 pairs (recursive)
glmnet	Regression using glmnet on a sparse 20 000 × 1000 matrix

Tab. 7.2: Machine learning benchmark set.

Benchmark	Description
ada	Boosting of classification trees
gbm	Gradient boosting machine
kknn	*k*-nearest neighbour classification
lda	Linear discriminant analysis
logreg	Logistic regression (binary classification decision derived using a probability cutpoint of 0.5)
lssvm	Least-squares support vector machine
naiveBayes	Naive Bayes classification
randomForest	Random classification forest
rda	Regularized discriminant analysis
rpart	Recursive partitioning for classification trees

to achieve accurate results. The machine learning benchmarks were configured to use a 20-fold cross-validation. The size of the input dataset (15 000 samples with 200 numeric features) was chosen to ensure that the runtime of the fastest algorithms is approximately one minute on the standard interpreter. To allow for a better comparison of the memory requirements, the same dataset was applied to all machine learning algorithms.

Each benchmark was executed 10 times with the standard and the optimized version of the R interpreter. The results are given as the arithmetic mean of these 10 executions. To make the results reproducible, the random number seed is selected as a fixed value placed as the first statement in each of the benchmarks. Each repetition was started in a fresh interpreter process; hence initialization costs are included in the measurements (an expected overhead on the order of one second or less). The R interpreter does not use adaptive optimizations. All system services that might interfere with the measurements were disabled. Both runtime and memory usage were measured simultaneously. For these measurements, we calculated a 95 % confidence interval and the ratio of the means using the percentile bootstrap method. We use geometric means here to reduce the influence of outliers.

Memory Consumption To analyze the benefits of the page sharing optimization techniques with regard to the memory consumption we evaluate the global peak memory usage and the average memory usage of each benchmarks. The *Peak usage* represents the maximum amount of memory that was consumed during execution of a benchmark. However, the peak memory consumption does not represent information about changing memory usage over time, since the peak memory usage may occur only for an instant of time depending on the benchmark. To gain a complete view of the memory consumption the short-term peak usage is measured in intervals of 1 second, resulting in a memory-over-time profile. The *Average usage* of memory is calculated as the arithmetic mean of these 1 second measurements and used as a second indicator to allow easier comparisons of the memory behavior.

Figure 7.6 shows the peak (*Peak usage*) and average (*Average usage*) memory consumption of each benchmark running with the page-sharing optimization. The 100 % baseline represents the standard R interpreter without optimizations. Values below this baseline indicate relative memory savings realized by the page sharing strategies. Error bars have been omitted as the confidence intervals were smaller than 0.5 % for all values. The detailed values are presented in Table 7.3, including the number of pages identified as shareable by the content check. They indicate the optimization potential of the dynamic refinement (deduplication of zero pages).

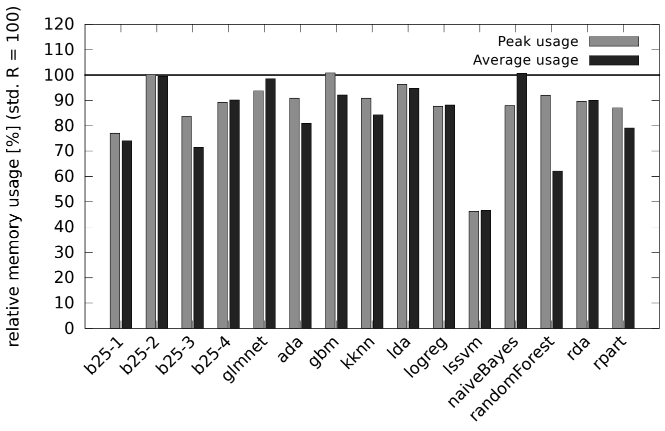


Fig. 7.6: Relative memory usage with page-sharing optimization compared with standard R (lower is better). The 100 % baseline represents the standard R interpreter without optimizations. Geometric means for the memory savings are 13.6 % for peak and 18.0 % for average memory usage [385].

The gain for reducing the peak memory usage (*GainP*) of the standard R interpreter (*StdPeak*) ranges from −0.9 % for gbm to 53.8 % for lssvm. However, the negative values in the columns *GainP* and *GainA* of Table 7.3 indicate that the page-sharing optimizations do not gain memory savings for three of the benchmarks. Here, the peak memory

Tab. 7.3: Memory Optimization Results: StdPeak – peak memory usage by the standard R interpreter; OptPeak – peak memory usage by optimized interpreter; GainP – relative peak memory reduction achieved by optimization; StdAvg – average memory usage by the standard interpreter; OptAvg – average memory usage by optimized interpreter; GainA – relative average memory reduction achieved by optimization; ZPG – number of zero pages found by the content check [385].

Benchmark	StdPeak [MB]	OptPeak [MB]	GainP [%]	StdAvg [MB]	OptAvg [MB]	GainA [%]	ZPG [#]
GU/08a-1	296.2	228.1	23.0	259.6	192.2	25.9	13
GU/08a-2	131.1	131.4	-0.2	128.8	128.0	0.6	13
GU/08a-3	197.2	164.8	16.4	157.7	112.6	28.6	37 919
GU/08a-4	134.2	119.7	10.8	127.2	114.6	9.9	194 892
glmnet	354.9	332.8	6.2	249.5	246.0	1.4	46 877
ada	187.2	170.1	9.1	156.0	126.2	19.1	2 031 992
gbm	191.5	193.2	-0.9	147.7	136.0	7.9	464
kknn	316.5	287.6	9.1	274.0	231.0	15.7	421
lda	216.2	208.2	3.7	184.8	175.1	5.3	20 447
logreg	213.0	186.7	12.3	184.7	162.8	11.9	955
lssvm	1365.1	631.0	53.8	820.2	381.1	53.5	3 972 699
naiveBayes	143.6	126.2	12.1	80.8	81.3	-0.6	78
randomForest	565.5	520.4	8.0	390.8	242.7	37.9	1 130 650
rda	254.1	227.7	10.4	197.0	177.3	10.0	707
rpart	144.5	125.8	12.9	130.7	103.3	20.9	56 214

consumption for two of the benchmarks (`gbm`, `GU/08a-2`) and the average memory consumption for one benchmark (`naiveBayes`) increase slightly. This is caused by the additional data structures that are needed for the internal handling of memory pages.

For `gbm`, a reduction of the average memory usage by 7.9 % (*GainA*) is achieved. For `naiveBayes` the situation is reversed: the optimization saves 12.1 % of its peak memory usage while its average memory usage (−0.6 %) is slightly increased. Since the number of pages recovered by deduplication (see column *ZPG*) is low (78), the savings of the peak memory usage are assumed to be induced by the proactive avoidance of page duplicates via the optimized allocation and duplication strategies. For `GU/08a-2`, the optimization was not able to save memory for peak memory usage and no meaningful amount for the average memory usage was saved (*GainA*). The reason why `GU/08a-2` does not gain from the optimization is that even though it uses large vectors with 2.4 million elements, it allocates a vector that is immediately filled with random numbers. Thus, it does not profit from the optimized allocation and the content check can only recover a low number of zero pages as shown column *ZPG* (13). GU/08a-2 does not use any object duplication. Therefore, the optimized duplication has no potential for saving memory.

Even though the page-sharing optimization results in a slight increase of peak or average memory usage for the three benchmarks described above, all of the twelve other benchmarks benefit from the optimization with savings in both peak and average memory usage. We compute the geometric mean over all 15 benchmarks, thereby avoiding the impact of outliers on the geometric mean. The result is a reduction of peak memory usage by 13.6 % and a reduction of average memory usage by 18.0 %. Here, the highest amount of memory that could be saved occurs in the `lssvm` benchmark with 53.8 % for peak usage and in `randomForest` with 37.9 % for the average memory usage. Both of these benchmarks have a high number of zero pages recovered by the content check. Thus for those benchmarks, the reduction of the memory footprint is not just triggered by the allocation and duplication optimization but also by the dynamic refinement that deduplicates zero pages.

Table 7.3 shows summarized values for the memory consumption over the complete runtimes of all benchmarks. To gain additional insights into the memory consumption behavior, the complete profile of the memory usage over runtime will be also analyzed. The four most interesting memory consumption profiles for the benchmarks (`glmnet`, `gbm`, `randomForest`, and `naiveBayes`) are shown in Figure 7.7. For each benchmark, the run with the execution time closest to the average of its 10 executions is selected. The confidence intervals over all 10 runs of each benchmark are less than 1 %, thus the figure shows only the data from a single run. The x-axis represents the runtime in seconds while the y-axis represents the corresponding memory consumption of the benchmark. Both the profile for the standard R interpreter (yellow curves) and the interpreter including the page-sharing optimizations (green curves) are presented. The straight lines at the top indicate the peak memory usage, while the dotted lines mark the average memory usage.

glmnet As mentioned, the `glmnet` benchmark utilizes an already-existing memory optimization for sparse matrices. It is included in the evaluation to determine if the page-sharing optimization can offer additional memory savings in the presence of an optimization that applies specialized application knowledge. In the top left of Figure 7.7 the memory-over-time behavior of this benchmark is illustrated. While there is only a small improvement for the average memory usage (see dotted green line), 6.2 % of the peak memory consumption is saved (see lines on the top). The memory consumption curves show that at all local memory peaks the optimized version of the R interpreter saves a small amount of memory while the memory consumption during the remaining parts of the execution is largely unaffected. This results in only a minor reduction of the average memory consumption. Still, even in the presence of a very specific optimization for sparse matrices the page sharing optimization can offer additional memory savings. As can be seen from column *ZPG* (Table 7.3), savings are triggered by a large number of pages recovered by deduplication (46 877).

gbm Not all benchmarks benefit from the content checks, though. For example, Table 7.3 shows that in `gbm` only 464 zero pages are recovered. This benchmark benefits more from the optimizations in allocation and duplication. The corresponding memory-over-time behavior is shown in the top right of Figure 7.7. Here, the optimization does not reduce the peaks of the memory consumption, but there is a

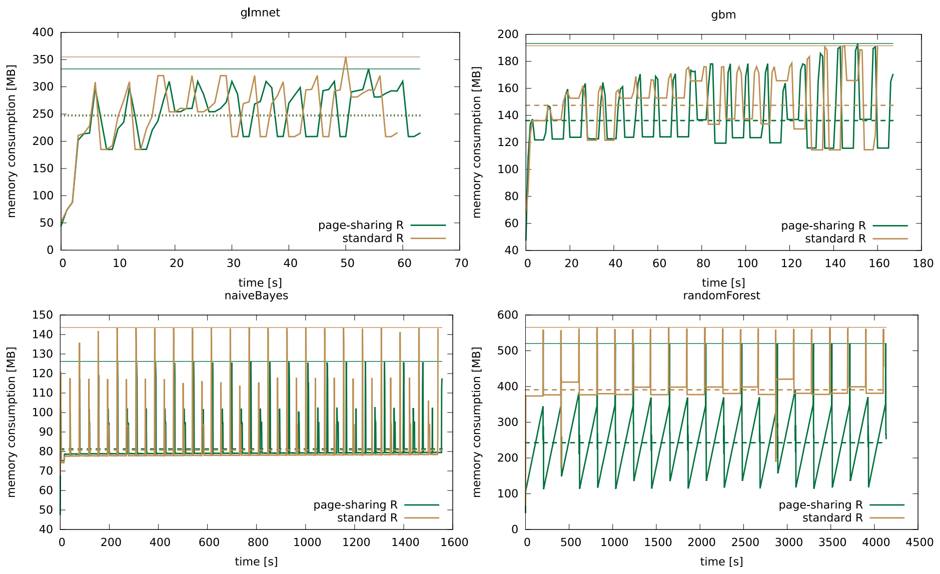


Fig. 7.7: Memory consumption over time profiles for benchmarks with different memory behavior for the standard R interpreter vs. the interpreter with the page-sharing optimization. Lines at the top indicate the peak memory usage; dotted lines mark the average memory usage [385].

marked reduction of memory usage in the valleys between the peaks, reducing the average memory consumption by 7.9 %.

naiveBayes Another benchmark that does not benefit from the content checks is `naiveBayes` with just 78 zeroed pages recovered. Its memory-over-time profile is illustrated in the bottom left of Figure 7.7. In `naiveBayes` only the peak memory usage is reduced by the optimization (large distance between the straight lines at the top), but the average memory usage (small distance between the dotted lines) is not affected. The profile also shows that `naiveBayes` has much smaller peaks compared with `gbm`. Thus, the large reduction of memory usage at those peaks results only in a small effect on the average memory consumption.

randomForest Finally, `randomForest` in the bottom right of Figure 7.7 represents one of the benchmarks where the recovery of zeroed pages triggers high memory savings. Here, the content checking reclaims 1 130 650 pages, which corresponds to slightly more than 4 GB of memory. The `randomForest` profile shows a saw-tooth curve for the optimized interpreter (see green curve). This indicates that the benchmark uses large blocks of memory that are slowly written to. For the page sharing optimizations, this represents an ideal memory usage pattern, as the allocation of memory is delayed until the benchmark writes data to it. This results in a 37.9 % improvement of the average memory consumption (large distance between dotted lines)—the average time during which the benchmark has a high memory consumption is thus reduced.

Looking back at the profile of `glmnet` (top left), the green curve that shows the profile for the optimized interpreter is longer than the yellow curve for the standard interpreter and there is an increasing shift between the peaks of both curves over time. The reason for this lies in the additional CPU time needed to provide the page-sharing optimizations. The runtime overhead induced by the memory optimization will be referred to in the next paragraph.

Runtime Overhead There are multiple reasons for the runtime overhead caused by the optimizations. For the 15 benchmarks shown so far, 4 have a runtime overhead of ≤ 1 %, an additional 6 have an overhead ≤ 5 %, an additional 2 have an overhead around 8 %, and the remaining 3 have an overhead between 13 % and 17 %. More details on the overhead are available in a separate publication [385].

Runtime Reduction In all previous measurements, the RAM available in the system was sufficient to hold all data used by the benchmark. If this is not the case, runtime overhead can become insignificant. This will be illustrated in the following. When the amount of RAM in the system is too small to hold all data required, there are situations where the proposed memory optimization is also able to reduce the runtime of the benchmark instead of adding overhead. This is due to frequent page swaps requiring I/O when the total capacity of RAM is exceeded, also known as “thrashing”. To analyze this situation, two benchmarks are considered. The first one is the lssvm benchmark where the optimization provides a large reduction in memory consumption. The second benchmark is an instance of logreg where the optimization provides only smaller memory gains.

For the analysis, the memory requirements of the benchmarks need to be increased beyond the capacity of the RAM in the system. Instead of increasing the dataset size of both benchmarks, the system is limited to just 1 GB of RAM, since the runtimes of the benchmarks do not scale linearly with the dataset size, leading to excessively high execution times. However, since the logreg benchmark has a much smaller memory consumption than 1 GB, the dataset size for logreg is increased to 70 000 samples with 300 numeric features. This increases the memory requirements of this benchmark to approximately the same level as lssvm. This still results in acceptable execution times for logreg.

Table 7.4 shows the results for the previous 6 GB system configuration and the limited 1 GB RAM configuration for both benchmarks. The logreg benchmark is now shown as logreg-2 because it was executed with the previously described larger dataset. In the 1 GB configuration, the system had to swap for both the standard and optimized interpreters, resulting in a large increase in runtime compared with the 6 GB configuration. The peak memory usage for the interpreters is identical in both configurations while the average memory usage differs because this value is time-dependent and thus influenced by swapping. This swapping also increases the variability in the runtime

Tab. 7.4: Evaluation results with two configurations of RAM; Std – standard R interpreter; Opt – optimized R interpreter; Gain – relative gain; Speedup: runtime speedup factor (Std / Opt). Confidence intervals (C) for runtime are shown; others are ≤ 0.8 % [385].

Benchmark	Std Peak [MB]	Opt Peak [MB]	Gain Peak [%]	Std Avg [MB]	Opt Avg [MB]	Gain Avg [%]
logreg-2, 1 GB	1228.2	1094.8	10.9	965.7	789.6	18.2
logreg-2, 6 GB	1228.2	1094.8	10.9	967.8	823.2	14.9
lssvm, 1 GB	1365.1	631.1	53.8	970.0	381.3	60.7
lssvm, 6 GB	1365.1	631.0	53.8	820.2	381.1	53.5

Benchmark	Std [s]	Opt [s]	Speedup (CI)
logreg-2 1 GB	6395.5	5785.6	$1.105^{1.144}_{1.071}$
logreg-2, 6 GB	579.8	598.5	$0.969^{0.971}_{0.967}$
lssvm, 1 GB	3080.3	593.8	$5.188^{5.350}_{5.029}$
lssvm, 6 GB	530.5	601.2	$0.882^{0.885}_{0.880}$

measurements, thus the confidence intervals for the speedup factors are also included (see lower part of Table 7.4).

Reducing the available memory from 6 GB to 1 GB drastically increases the runtime for both versions, the standard R interpreter (*Std*) and the interpreter including the memory optimization (*Opt*). Still, the reduction in memory consumption for logreg-2 has turned the slowdown (factor 0.969) in its 6 GB configuration into a small speedup (factor 1.105) when the RAM is limited to 1 GB. Depending on the benchmark and its memory usage pattern, a different situation could also happen. In the worst case, the content check of the optimized interpreter touches a large number of pages, forcing them to be swapped in. This additional swap activity can increase the runtime so that the gains from a reduced memory footprint may become irrelevant. The second benchmark `lssvm` shows something closer to the best case for the optimization: Here, the page-sharing optimization manages to save enough memory to avoid swapping. In this case, significant speedups are gained, as shown in the lower part of Table 7.4 for the 1 GB configuration of `lssvm`.

Similar to `logreg-2`, memory usage does not vary much between both configurations (see upper part of Table 7.4). Considering the runtime results, the optimized interpreter *Opt* only needs 593.8 seconds to run the `lssvm` benchmark. This is almost unchanged from the 6 GB configuration (601.2 seconds). By contrast, the standard interpreter *Std* has now increased its runtime to 3080.3 seconds (51.3 min.) when limited to 1 GB of RAM. This makes the overhead of the memory optimization irrelevant because the time gained by avoiding page I/Os is much larger. The page-sharing optimization enables a speed up by a factor of 5.2 for llsvm by reducing the peak memory consumption by 53.8 %. This speed up is also illustrated in Figure 7.8. It shows the

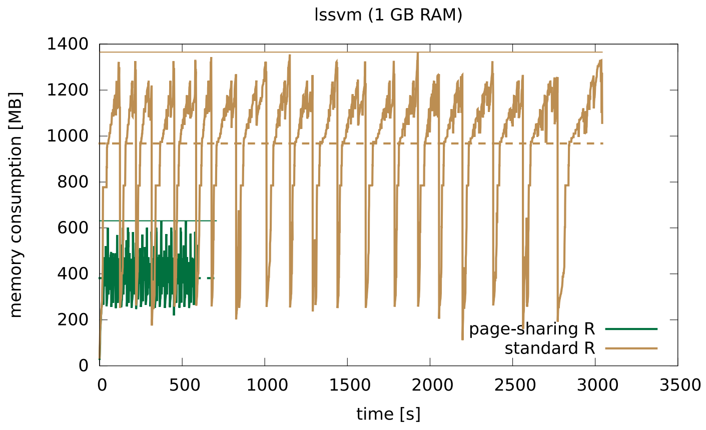


Fig. 7.8: Memory consumption over time profile for the lssvm benchmark. Speed-up reaches a factor of 5.2 on a system with 1GB of RAM. Solid lines indicate the peak memory and dotted lines mark the average memory usage [385].

memory consumption profile for one exemplary execution of the `lssvm` benchmark. This demonstrates that reducing the memory consumption with the page-sharing optimization can significantly improve the runtime for memory-hungry benchmarks if the available RAM is constrained. In turn, this can enable the processing of larger datasets.

7.1.5 Summary

The R interpreter induces a large memory overhead in the machine learning applications, due to wasteful memory allocation [387]. The goal of the presented memory optimizations was to enable efficient memory utilization, especially for memory-hungry R applications like machine learning algorithms. To accomplish this goal, this contribution presented an application-transparent memory optimization employing page sharing at a memory management layer between the R interpreter and the operating system's memory management. The optimization benefits a large number of applications since it preserves compatibility with the available software libraries that most R programs are based on, and covers one of the most important resource bottlenecks of machine learning algorithms. By concentrating on the most rewarding optimizations—the sharing of zero-filled pages and deduplicating at the page level instead of the object level—the overhead of more general OS level memory optimization approaches such as deduplication and compression is avoided. With the proposed optimization, considerable reductions of the memory consumption for a large number of typical real-world benchmarks have been achieved. This is an important step towards processing larger input sizes. It also significantly speeds up the computation in cases where previously pages had to be swapped out due to insufficient main memory.

7.1.6 Conclusion

Designers of machine learning applications should be allowed to focus on the functionality of their algorithms. In order to execute these on resource-constrained embedded systems, possible optimizations of the implementation should be performed. The presented work demonstrates the benefits of such optimizations for the case of memory resources. In addition to the other optimizations in this contribution, we conjecture that more memory-oriented optimizations exist and propose that they should be exploited in order to execute machine learning algorithms in particular on hardware with limited amounts of memory.

7.2 Machine Learning Based on Emerging Memories

Mikail Yayla
Sebastian Buschjäger
Hussam Amrouch

Abstract: Due to the exceptional recent developments in deep learning, many fields have benefited from the application of Artificial Neural Networks (ANNs). One of the biggest challenges in ANNs, however, is the resource demand. To achieve high accuracy, ANNs rely on deep architectures and a massive amount of parameters. Due to this, the memory sub-system is one of the most significant bottlenecks in ANNs.

To overcome the memory bottleneck, recent studies have proposed using approximate memory in which the supply voltage and access latency parameters are tuned for lower energy consumption and for faster access times. However, these approximate memories frequently exhibit bit errors during the read process. Typical software solutions that monitor and correct these errors require a large processing overhead that can negate the performance gains of executing ANNs on these devices. Hence, error-tolerant ANNs that work well under uncorrected errors are required to prevent performance degradation in terms of accuracy and processing speed.

In this contribution, we review the available and emerging memories that can be used with ANNs, with a focus on approximate memories, and then present methods to optimize ANNs for error tolerance. For memories, we survey existing memory technologies such as Static Random-Access Memory (SRAM) and Dynamic Random Access Memory (DRAM), but also present emerging memory technologies such as Ferroelectric FET (FeFET), and explain how the modeling on the device level needs to be performed for error tolerance evaluations with ANNs. Since most approximate memories have similar error models, we assume a general error model and use it for the optimization and evaluation of the error tolerance in ANNs. We use a novel hinge loss based on margins in ANNs for error tolerance optimization and compare it with the traditional flip regularization. We focus on Binarized Neural Networks (BNNs), which are one of the most resource-efficient variants of ANNs.

7.2.1 Introduction

Artificial neural networks have been applied successfully in numerous fields, and are being executed on a variety of systems ranging from large computing clusters to small, battery-driven embedded systems. In most cases, state-of-the-art neural network models rely on a large number of parameters to achieve high performance. This leads to an expensive, slow, and energy-consuming *memory bottleneck*. On neural network acceleratorswith SRAM, the energy consumption of the memory makes up the largest fraction of system energy, while advances in memory bandwidth are significantly slower than processing speed. Hence, improving the memory consumption of ANNs and improving the memory sub-systems is imperative to further push the applications of ANNs. One design paradigm to improve the memory sub-system is to use approximate memory in which resource efficiency is achieved by allowing for bit errors during the read and/or write process. Likewise, reducing the memory consumption of ANNs is an established part of deep learning research. Here, arguably, the most extreme form is to use Binarized Neural Networks (BNNs) that only use binary weights $\{0, 1\}$ leading to a potential 32 times memory reduction as high as 32 times that of their floating-point siblings. Interestingly, it has been shown that BNNs can be trained to tolerate bit errors by bit flip injections during training. However, this method has a large overhead and does not scale well with model size and higher bit error rates .

In this contribution, we first summarize the currently available and emerging memories that are possible to be used with neural network inference systems. Here, we focus on approximate memories, which are unreliable due to bit errors and for which countermeasures are necessary. One of the most promising emerging memory components is the FeFET, which has high speed, and low energy consumption, but faces reliability issues. We explain how FeFET can be used as approximate memory for neural networks despite the bit errors caused by temperature and read voltage. Finally, we present results on how bit error tolerance in ANNs is achieved without bit flip injections based on margin-maximization and compare it to the traditional methods for bit error tolerance optimization of ANNs. This contribution was previously published as a conference paper in [113].

7.2.2 Emerging Memories

Recent studies on efficient ANN-based inference systems have explored the use of approximate memory, which has been realized by reducing the memory supply voltage and tuning latency parameters with the goal of lower power consumption and faster access. If these methods are pushed to the limit, high Bit Error Rates (BERs) can occur. Before discussing bit errors and how to deal with them in more detail we will quickly survey volatile memories (SRAM, DRAM) and other emerging non-volatile memories (FeFET, Resistive Random Access Memory (RRAM), Spin Transfer Torque Random Access Memory (STT-RAM) or Magnetoresistive Random Access Memory (MRAM)) here.

SRAM For ANN inference systems using on-chip SRAM, the works in the literature mainly employ scaling of various device parameters. To reduce energy consumption, the SRAM voltage is scaled in [306, 652]. Yang et al. [717] separately tune the weight and activation values of BNNs to achieve fine-grained control over the energy consumption. Sun et al. [652] propose similar techniques for ternary ANNs. A similar approach is employed by Henwood et al. [306], in which layer-wise the best energy-accuracy trade-off for SRAM is chosen.

DRAM For DRAM, the study by Koppula et al. [381] provides an overview of studies related to ANNs that use different DRAM technologies and proposes a framework for evaluating ANN accuracy when using approximate DRAM in various different settings and inference systems. Specifically, the study shows that DRAM parameters can be tuned such that energy and performance are optimized to achieve significant improvements, whereas the ANN accuracy drop stays negligible due to the ANNs' adaptations in retraining. Other studies, e.g. [532, 672], also optimize the refresh rate of DRAM to achieve energy savings.

RRAM Hirtzlin et al. [316] propose computing BNN operations with RRAM that features in-memory processing capabilities. They set the write energy of RRAM low and show that BNNs can tolerate the resulting errors by error tolerance training. This low-energy setting also increases the RRAM cell lifetime since low-energy writes stress the cells less. The work by Yu et al. [727] also uses RRAM to implement on-chip BNNs. They show that under limited bit yield, BNNs can still operate with satisfying accuracy. Sun et al. [651] propose an RRAM synaptic array to deploy BNNs. They investigate the accuracy impact of errors from sense amplifiers that have offsets due to process variation.

MRAM or STT-RAM Another branch in the literature is about ANNs on STT-RAM or MRAM. Hirtzlin et al. [315] propose deploying BNNs on MRAM with a low-energy programming setting that causes relatively low error rates, and no significant accuracy drop, but decreases write energy by a factor of two. Tzoufras et al. [675] also propose operating BNNs on MRAM with reduced voltage with similar results. They test a wide range of error rates and discuss the implications of BNN bit error tolerance on the lifetime, performance, and density of MRAM. Pan et al. [549] take a different approach for energy reduction and investigate the benefits of multi-level cell MRAM for the in-memory acceleration of BNNs. For more general ANN models, Vincent et al. [686] propose tunable STT-RAM to save resources.

FeFET FeFET is considered to be one of the most promising memory technologies. The reason why FeFET store logic '0' and logic '1' lies in the available dipoles inside the FE. The directions of these dipoles can switch if a sufficiently strong electric field is applied. This state is non-volatile because the dipoles retain their direction when the field is turned off. The logic '0' and logic '1' can be read out from the FeFET based on the intensity of the current returned (e.g. high or low), which can be converted into the digital domain with sensing circuits.
The three main advantages of FeFET over other non-volatile memories are as follows:

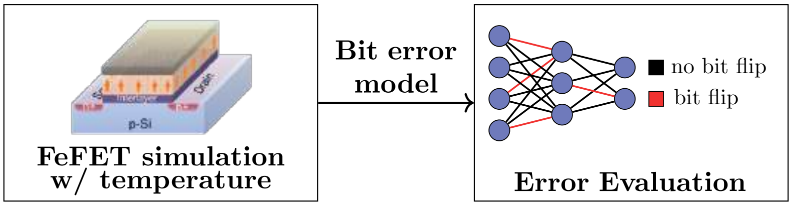


Fig. 7.9: Errors due to temperature, stemming from underlying FeFET devices, are modeled and then injected during the ANN inference [720].

1. FeFET is fully CMOS-compatible, which means that it can be fabricated using current manufacturing processes. This has been demonstrated by GlobalFoundries [668].
2. FeFET-based memories can perform read operations within 1ns latency. This reduces the differences to traditional SRAM technology, while the energy usage of FeFET is significantly lower [668].
3. FeFET memory has the potential to enable extremely low-density memory since the core cell consists merely of a single transistor.

One of the major disadvantages of FeFETs is error susceptibility. Manufacturing variability (i.e. process variation during production) and temperature fluctuations at run-time can cause variations in the FeFET properties. This shrinks available noise margins and may cause errors. To still employ FeFETs despite the errors in, say, on-chip memory for Binarized Neural Networks (BNNs) inference systems, it is necessary to extract the error models for the stored bits. With the error model, the impact of the temperature-induced bit errors on the inference accuracy of BNNs can be evaluated.

In Figure 7.9, the steps for extracting the temperature-dependent error model of FeFET transistors are shown. The entire FeFET device has been implemented and modeled in the Technology CAD (TCAD) framework (Synopsys Sentaurus [656]). The variation in the underlying transistor and the added ferroelectric layer are considered. After incorporating the temperature and variation effects in the calibrated TCAD models, Monte-Carlo simulations for the entire FeFET device are performed. Then the probability of error is extracted for a certain read voltage, i.e. the probability that logic '0' is read as logic '1' and a logic '1' is read as logic '0'. Details on device physics modeling and reliability analysis for FeFET under the effects of temperature variability (runtime) and manufacturing (design-time) variability can be found in [280] and [534], respectively.

7.2.3 Binarized Neural Networks

Traditional neural networks use floating-point (e.g., 32 bits) or integer values (e.g., 8 bits) to represent the ANN parameters (i.e., weights, activations, inputs, etc.) . In such a case, the position of the occurred bit error (i.e., the bit flip in the value) does matter. Specifically, in floating-point ANNs, a one-bit error in one weight can cause the prediction of the ANN to become useless (see e.g. [381]). This typically occurs when a bit flip in the exponent of the floating-point representation occurs leading to an error with an unacceptable magnitude. As mentioned before, BNNs are resource-efficient neural networks that are ideally suited for small devices. Additionally, they can be trained to be resilient against bit errors, which makes them ideal candidates for approximate memories. In BNNs each weight (and possibly each activation) is stored in a single bit $\{0, 1\}$. Hence, a bit error in a binary weight or binary input causes a change of the computation result by merely 1, reducing its overall impact. In addition to the reduced impact of bit errors and reduced memory footprint due to smaller weights the execution of BNNs also becomes simpler. Consider, for example, the output of the fully connected l-layer with activation σ and weights W^l

$$f^l(X) = \sigma(W^l X) \tag{7.1}$$

In regular floating-point neural networks, the execution of this layer requires the repeated computation of matrix-vector products $W^l X$ as well as the application of σ. In a BNN this operation becomes

$$2\texttt{popcount}(\texttt{XNOR}(W^l, X)) - B > T \tag{7.2}$$

where `popcount` counts the number of 1s in the XNOR-result, B is the number of bits in the XNOR operands, and T is a learnable threshold parameter if batch normalization layers are used, whose comparison produces binary values (representing a shifted binarization function) [325, 609].

A common method of training ANNs is to apply stochastic gradient descent (SGD) with mini-batches. Let $\mathcal{D} = \{(x_1, y_1), \ldots, (x_I, y_I)\}$ be the training data with $x_i \in \mathcal{X}$ as the inputs, $y_i \in \mathcal{Y}$ as the labels, and $\ell\colon \mathcal{Y}\times\mathcal{Y} \to \mathbb{R}$ as the loss function. $W = (W^1, \ldots, W^L)$ are the weight tensors of layer $1 \ldots L$ and $f_W(x)$ is the output of the ANN. The goal is to find a solution for the optimization problem

$$\arg\min_W \frac{1}{I} \sum_{(x,y)\in\mathcal{D}} \ell(f_W(x), y) \tag{7.3}$$

with a mini-batch SGD strategy that computes gradients using backpropagation.

To train BNNs, Hubara et al. [325] proposes to deterministically binarize the weights and activations during the forward pass. For backpropagation, the floating-point numbers are used for parameter updates. This leads to training times similar regular ANNs but assumes binary values during the forward pass. More formally, let $b\colon \mathbb{R} \to \{-1, +1\}$ be a binarization function with

$$b(x) = \begin{cases} 1 & x > 0 \\ -1 & \text{else} \end{cases} \tag{7.4}$$

and let $B(W)$ denote the element-wise application of b to a tensor W. Now we simply apply B during the forward pass to each weight tensor. During the backward pass, the authors propose using full floating point precision, whereas during the backward-pass they replace the gradient of b with the straight-through estimator. Consider the forward computation $Y = B(X)$. Let $\nabla_Y \ell$ denote the gradient with respect to Y. The straight-through estimator approximates

$$\nabla_X \ell := \nabla_Y \ell, \tag{7.5}$$

essentially pretending that B is the identity function. Algorithm 5 summarizes this approach.

Algorithm 5: Binarized forward pass for a network with L layers, each with weight tensors W^l performing a generic operation $\circ^l$ (e.g. a convolution).

```
1 for $l \in \{1, ..., L\}$ do
2     $x \leftarrow B(B(W^l) \circ^l x)$
```

7.2.3.1 Flip Regularization

To make BNNs bit error-tolerant, the state-of-the-art method is bit flip injections in the binarized values during the forward pass, as proposed by Hirtzlin et al. [316]. The idea is simple: To make BNNs robust against bit errors, we simulate the errors already during training time. During each forward pass computation, we generate a random bit-flip mask and apply it to the binary weights.

Let M denote a random bit-flip mask with entries ±1 of the same size as W that we multiply component-wise to the binarized weights. We first consider computing the bit-flip operation as $H = (B(W) \cdot M) \circ X$ where $\circ$ denotes the application of the ANN to the input X. Standard backpropagation on a loss ℓ that is a function of H yields the following gradient of ℓ with respect to $B(W)$

$$\nabla_{B(W)} \ell = M \cdot \nabla_{B(W) \cdot M} \ell \tag{7.6}$$

which for fully connected layers amounts to a gradient update

$$\nabla_{B(W)} \ell = M \cdot (\nabla_H \ell \, X^T). \tag{7.7}$$

We see that an update computed this way accounts for the bit-flips that were performed. We propose instead using a special flip-operator with straight-through gradient approximation. We denote by e_p the bit error function that flips its input with probability p and let E_p denote its component-wise version. During training, we change the forward pass such that it computes

$$X^{l+1} := B(E_p(B(W^l)) \circ X^l). \tag{7.8}$$

We replace the gradient of E_p with a straight-through approximation. This way, in the example above we now have $H = E_p(B(W)) \circ X$ with gradient updates $\nabla_{B(W)}\ell = \nabla_{E_p(B(W))}\ell$ which for fully connected layers yields the update

$$\nabla_{B(W)}\ell = \nabla_H \ell \, X^T \tag{7.9}$$

which is unaware of bit flips and just uses the corrupted outputs H.

The original bit-flip regularization proposed in [316] reports extreme overfitting to the flip probability used during training. As we will see later in the experiments, we do not report such an overfitting. We believe that the approach using straight-through gradient approximation is superior and that the extreme overfitting is attributable to the use of the naive gradient.

7.2.3.2 Margin-Maximization for Bit Error Tolerance Optimization

Bit-flip regularization improves the error tolerance of the network by simulating bit errors during the forward pass. This introduces two objectives to the training: Given a set of labeled input data, train a BNN for high accuracy and for high bit error tolerance. Hence, another approach is to combine high accuracy and high bit error tolerance into a single loss function directly so that both objectives are jointly optimized during training. To do so, we now introduce a margin-based neuron-level bit error tolerance metric for BNNs that is then extended to formulate a bit error tolerance metric for the output layer.

In the following, we use a notation describing the properties of neurons in convolutional layers, but our considerations also apply to neurons in fully connected layers. Let n be the index of one neuron in a ANN, and $x \in \mathcal{X}$ an input to the ANN. The output of a neuron in a convolutional layer is a feature map with height U and width V. Let $h_{x,n,u,v} \in \mathbb{Z}$ be the pre-activation value of neuron n at place $(u, v) \in \{0, \dots, U\} \times \{0, \dots, V\}$, *before* applying the activation function. For BNNs, the pre-activation values of a neuron are computed by a weighted sum of inputs and weights that are ±1. Therefore, one bit flip in one weight changes the pre-activation value by 2.

Theorem 25. *Let $n \in \{0, \dots, N\}$ be the index of one neuron. Furthermore, let q be the number of bit flips induced in the weights of neuron n. The pre-activation of neuron n at place (u, v) after induction of these bit flips is in the interval $[h_{x,n,u,v} - 2q, h_{x,n,u,v} + 2q]$.*

The proof can be found in [113].

A detailed analysis of the error tolerance for hidden-layer neurons has been conducted in [114], but the use of Theorem 25 for optimizing bit error tolerance on the neuron-level has been reported to be unsuccessful. We hypothesize that bit flips of neuron outputs can only affect the BNN prediction if the effect of bit flips reaches the output layer and leads to a change in the predicted class. Therefore, we now shift our focus on applying the notion of margin to the output layer, i.e., to neurons with index in N_O.

Each neuron in the output layer has only one output value $h_{x,n,1,1}$ which is one entry in the vector of predictions $\hat{y}$. No activation function is applied to the output value of these neurons. There are as many values in $\hat{y}$ as there are neurons in the last layer. The index of the entry with the maximum value in $\hat{y}$ determines the class prediction, where we assume that ties are broken arbitrarily.

If bit errors modify the output values in the output layer such that another neuron provides the highest output value, then the class prediction changes. Let $h_{x,n',1,1}$ and $h_{x,n'',1,1}$ with $n', n'' \in N_O$ be the highest and the second-highest output value of neurons in the output layer. The following corollary shows that the margin

$$m := h_{x,n',1,1} - h_{x,n'',1,1} \tag{7.10}$$

serves as a bit error tolerance metric for the output layer.

Corollary 26. *If* $m > 0$, *then the output layer of the BNN tolerates* $\max(0, \lfloor \frac{m}{2} \rfloor - 1)$ *bit flips.*

The proof can be found in [113].

We now focus on constructing a loss function based on Corollary 26 and the hinge loss known from Support Vector Machines (SVMs). The hinge loss [602] for maximum margin classification is defined as

$$\ell(y, f) = \max(0, 1 - y \cdot f), \tag{7.11}$$

with the ground truth prediction $y = \pm 1$ and the prediction $f \in \mathbb{R}$. This loss becomes small if the predictions have the same sign as the predicted class and are close to 1 in magnitude. For predicted values larger than 1, the loss becomes 0. The “1” in the loss forces the classifier to maximize the margin between two class predictions.

For BER tolerance of the last layer, the margin m as introduced in Equation 7.10 needs to be large so that the maximum number of bit flips the output layer can tolerate is high. The margin can be directly computed by subtracting the second-highest entry $\hat{y}_{c''}$ of the output vector $\hat{y}$ from the highest entry $\hat{y}_{c'}$, i.e., $m = \hat{y}_{c'} - \hat{y}_{c''}$. However, optimizing with respect to m without considering the other entries $\hat{y}_c$ of $\hat{y}$ may not exhaust the full potential of the margin between $\hat{y}_{c'}$ and the output of the other classes $\hat{y}_c$. The larger the margin between $\hat{y}_{c'}$ and $\hat{y}_c$ of other classes c, i.e. $m_c = \hat{y}_{c'} - \hat{y}_c$, the more bit errors can be tolerated in the neuron that calculates $\hat{y}_c$ without a change in the prediction. To put it concisely, for a bit error tolerant output layer, $\hat{y}_{c'}$ needs to be as large as possible, while the other $\hat{y}_c$ need to be as small as possible.

In the case of BNNs for multi-class problems, however, the version of the hinge loss in Equation 7.11 cannot be directly used. To extend the hinge loss to multiple classes, we define y_{enc} as a one-hot vector with elements in $\{-1, 1\}$, which has a +1 at the index with the ground truth, else −1. y_{enc} has the same number of elements as $\hat{y}$. Then the element-wise product $y_{enc} \cdot \hat{y}$ is computed. In this product, in case of correct predictions, positive predictions in the correct class will stay positive, and negative predictions that

Tab. 7.5: Datasets used for experiments.

Name	# Train	# Test	# Dim	# classes
FashionMNIST	60000	10000	(1,28,28)	10
CIFAR10	50000	10000	(3,32,32)	10

Tab. 7.6: Parameters used for experiments.

Parameter	Range
Fashion FCNN	In → FC 2048 → FC 2048 → 10
Fashion CNN	In → C64 → MP 2 → C64 → MP 2 → FC2048 → 10
CIFAR10 CNN	In → C128 → C128 → MP 2 → C256 → C256 → MP 2 → C256 → C256 → MP 2 → FC 2048 → 10

should be as negative as possible become positive. In case of wrong predictions, i.e. high negative values for the correct class and high positive values for the wrong class, the values become negative. For a high penalty in the wrong case and a small penalty for the correct case, we subtract the product $y_{enc} \cdot \hat{y}$ from a parameter b, and get $(b - y_{enc} \cdot \hat{y})$. Since we do not demand higher prediction values than b, we set negative values to zero with the max function, and denote the Modified Hinge Loss (MHL):

$$\ell_{MHL}(\hat{y}, y_{enc}) = max\{0, (b - y_{enc} \cdot \hat{y})\}. \tag{7.12}$$

7.2.4 Experiments

We evaluate fully connected binarized neural networks (FCBNNs) and convolutional binarized neural networks (CBNNs) in the configurations shown in Table 7.6 for the datasets FashionMNIST and CIFAR10 (see Table 7.5). In all experiments, we run the Adam optimizer for 100 epochs for FashionMNIST and 250 epochs for CIFAR10. We use a batch size of 128 and an initial learning rate of 10^{-3}. To stabilize training, we exponentially decrease the learning rate every 25 epochs by 50 %. In the following, we compare the margin-based methods (MHL) to Flip Regularization (FR). FR uses the Cross-Entropy Loss (CEL) by default. We first compare MHL without FR to FR. In a second step, we compare MHL without FR to MHL in combination with FR.

7.2.4.1 MHL Only vs. FR

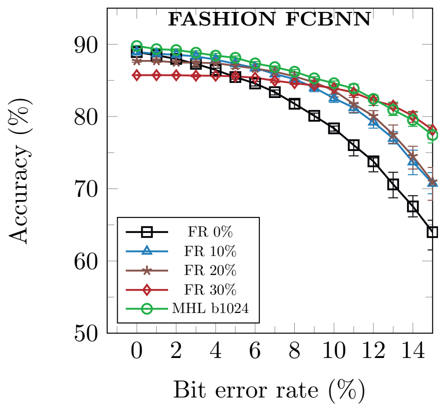
Figure 7.10 presents the experimental results of different BNNs with respect to the accuracy over BER (from 0 % to up to 15 % in Figure 7.10(a) and (b), and from 0 % to up to 5 % in Figure(c)). For each dataset, five BNNs were conducted using MHL without any FR and FR with different BERs for bit-flip injections. Moreover, for all BNNs trained with MHL, we employed a parameter search for b, testing powers of two, up to two times the maximum value the neurons in the output layer can compute (maximum output value of a neuron in the output layer is the number of neurons in the layer before the output layer). Among these configurations of b, the best one was chosen. We observe that BNNs trained with the MHL without FR have better accuracy over BER than the BNNs trained with FR, i.e., in Figure(a) and (b) up to 10 %, and in Figure(c) up to 5 %. The BNNs trained with FR suffer from a significant accuracy drop for lower BERs, when the BER during training is high, e.g., CEL 20 % and/or CEL 30 % at low BER. The BNNs trained with MHL, however, do not suffer from this accuracy drop. Although the BNNs trained with FR 20 % and bit-flip injections have better accuracy for Fashion CBNN in Figure 7.10(b) when the error rate is higher than 10 %, the accuracy of the BNNs drops by a significant amount, which may be unacceptable. Below, we thus present further investigations.

7.2.4.2 MHL Combined With FR

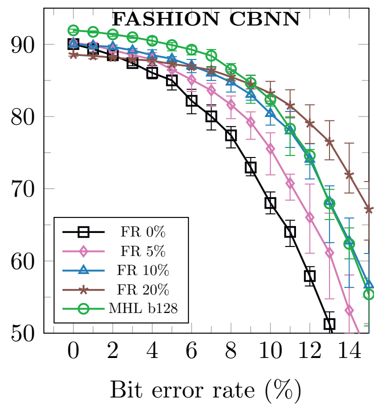
We evaluate BNNs trained with the MHL and FR under different BERs. In addition, the BNNs trained with the MHL without FR (i.e., those BNNs generated using the MHL in Figure 7.11 under 0 % BER) are included here as the baseline in this subsection. For all configurations, we employed the same parameter search for b as in the previous section. Figure 7.11 presents the experimental results of different BNNs with respect to the accuracy over BER (from 0 % to up to 30 % in Figure 7.11(a) and (b) and from 0 % to up to 6 % in (c)). In all experiments, we observe that the accuracy over the BER of the BNNs trained under MHL and FR is significantly higher than that of the baseline trained by only MHL. For example, for Fashion in Figure 7.11, the BER at which the accuracy degrades significantly is extended from 5 % (baseline, green curve) to 20 % and 15 %, respectively, with a small trade-off in the accuracy at 0 % BER. If more accuracy at low error bit rates is traded, the BER at which accuracy degrades steeply can be shifted even further. For CIFAR10 in Figure 7.11, this breaking point can also be increased. However, more accuracy has to be traded compared with previous cases. If b is higher than the ones shown, the accuracy for lower BERs suffers similarly to how it would using CEL with high BERs. If b is lower, there will be no significant change compared with CEL with 0 % BER. We only show the results with the best b.

7.2.5 Conclusion

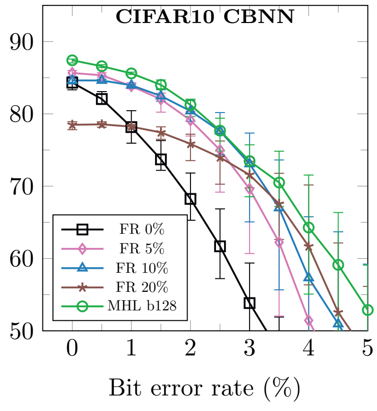
Deep learning is notoriously memory hungry and hence new memory sub-systems must be developed to push the application of ANNs to small devices. Likewise, new ANN architectures can help to reduce memory consumption and offer a more resource-friendly execution of deep networks. Non-volatile memories such as Ferroelectric FET (FeFET) are a promising technology for new memory sub-systems. FeFET enables faster and more energy-efficient read/write operations but it introduces bit errors into the execution. While standard software solutions can monitor and correct bit errors, they negate the advantages of non-volatile memories by introducing further processing overhead. Neural networks that are resilient to random bit errors by design, on the other hand, can retain the advantages of non-volatile memories leading to potentially faster and more energy-efficient solutions. BNNs are a novel class of small, resource-efficient neural nets that are ideally suited for such a setting. In BNNs each weight consists of weights $\{0, 1\}$ so that they require 32 times less memory than their floating-point counterpart while being more resilient to random bit flips. In this contribution, we provided an in-depth discussion of the bit errors in BNNs and derived a novel max-margin optimization from it. Our approach offers a better accuracy across most error rates while preventing the overfitting of the BNN to a specific error rate. Hence, our approach allows the deployment of BNNs on a variety of different devices with unknown and varying error rates.



(a) Fashion FCBNN

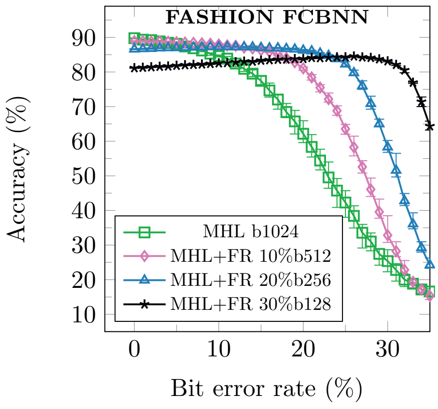


(b) Fashion CBNN

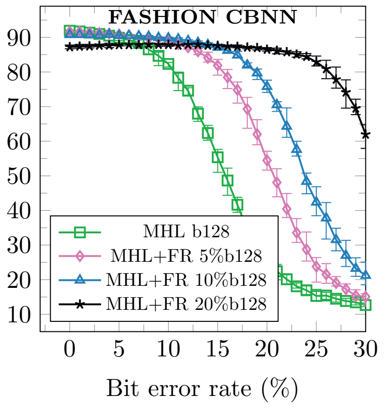


(c) CIFAR10 CBNN

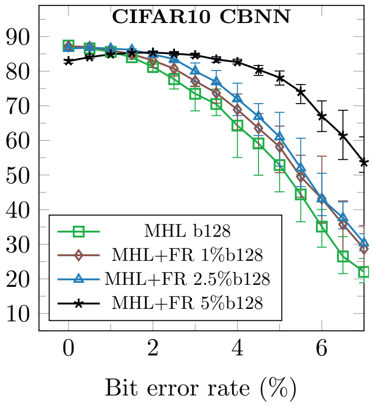
Fig. 7.10: Accuracy over bit error rate for BNNs trained with FR under a given bit flip injection rate (specified in the legend, 0 %, 5 %, 10 %, etc.) and BNNs trained with MHL without FR for a specified *b* in Equation 7.12.



(a) Fashion FCBNN



(b) Fashion CBNN



(c) CIFAR10 CBNN

Fig. 7.11: Accuracy over bit error rate for BNNs trained with MHL and FR (denoted as FR 0 %, 1 %, etc). The number after the *b* is the value to which the parameter *b* in the MHL is set during training (see Equation (7.12)).

7.3 Cache-Friendly Execution of Tree Ensembles

Sebastian Buschjäger
Kuan-Hsun Chen

Abstract: Ensembles of decision trees are among the most used classifiers in machine learning and regularly achieve state-of-the-art performance in many real-world applications, e.g., in the classification of celestial objects in astrophysics, pedestrian detection, etc. Machine learning practitioners are often concerned with model training, re-training different models again and again to achieve the best performance. Nevertheless, once a learned model is trained and validated, the executing cost of its continuous application might become the major concern.

Applying decision trees for inferences is very efficient in run-time, but it requires many memory accesses to retrieve nodes. For example, it is common to train several thousand trees, e.g., each with depth 15 leading to $2^{15} = 32\,768$ nodes per tree. This leads to millions of decision nodes that must be stored in memory and processed. Cache memory is commonly adopted to hide the long latency between the main memory and the processor. However, an improper memory layout might bring up additional cache misses, leading to performance degradation. Thus, designing a suitable memory layout of tree ensembles is of key importance to achieve efficient inference over tree ensembles.

In this contribution, we discuss the deployment of tree ensembles on different hardware architectures. Given a pre-trained decision tree ensemble, we first present different realization techniques commonly used in the literature. Afterwards, we study different layout strategies to optimize the node placement in the memory, focusing on the caches available on different hardware architectures. Finally, we present the evaluation results over different configurations and combine all approaches into a single framework that automatically generates the optimized realization for a target hardware architecture.

7.3.1 Introduction

Efficient learning has always been the focus of research, but the demand for the *efficient application* of learned models has emerged only recently. Consider, for example, self-driving cars. Current prototypes use machine learning (ML) for image recognition and fundamental steering.[1] Thus, the ML model must not only be applied continuously, but it also must react on time. As a second example, consider search engines that utilize ML

1 ,https://towardsdatascience.com/teslas-deep-learning-at-scale-7eed85b235d3.

models such as Gradient Boosted Trees[2] to rank search results. These engines routinely process roughly 12 billion search queries a month worldwide.[3] The 4 480 287 queries per second they process demand fast model application.

While deep learning is excellent for unstructured image data, tree ensembles are often referred to as one of the best black-box methods available for structured data. They offer high accuracy with only a few parameters to tune [120, 223] and frequently place among the top methods in data science competitions.[4] For real-time application, tree ensembles have become important in many domains, e.g., the real-time classification of celestial objects in astrophysics [115], real-time pedestrian detection [466], real-time 3D face analysis [211]), the real-time classification of noise signals [608], nano-particle sensors [439].

However, these trees are usually stored in the main memory and processed directly out of the memory. The runtime of such a memory-intensive application is mainly determined by the use of the various caches of the CPU. Surprisingly, as the line between realizational details and algorithmic contributions becomes blurry on modern computing systems, caching behavior determines the performance of implemented algorithms even more than algorithmic differences [615]. For tree ensembles, we can foresee that an analytical approach to an efficient layout of the memory is desirable. Given a pre-trained tree ensemble, we present several cache-aware approaches to optimize the memory layout (so-called tree-framing), while preserving the original ensembles' accuracy. The proposed approaches are wrapped in a code generator that automatically adapts to underlying architectures to produce optimized code segments. Overall, we present the following contributions:

Cache-aware tree-framing approaches We analyze the source of cache misses on two common tree realizations, i.e., *native* and *if-else* trees, and discuss several approaches at the application level to optimize the memory layout by artificially creating instruction/data locality .

Architecture-aware code generator We present a code generator that exploits the analytical insights for generating optimized realizations of a given tree ensemble. The code generator is publicly available at https://github.com/sbuschjaeger/fastinference.

Empirical evaluation We perform 1800 experiments across three different computer architectures and show that our approaches offer a speed-up factor at 2 – 4 on average without changing the prediction accuracy of the given trained model.

This contribution was previously published as a conference paper in [108] and was later expanded in a dissertation in [107].

2 https://www.seroundtable.com/bing-core-ranking-algorithm-machine-learning-27040.html.

3 Numbers are for 2019, see https://www.statista.com/topics/4294/bing/.

4 https://www.kdnuggets.com/2016/01/anthony-goldbloom-secret-winning-kaggle-competitions.html.

7.3.2 Related Work

Tree ensembles are some of the most used machine learning algorithms and, as such, have been studied extensively in the literature. In the context of model application and fast inference, there are two principled approaches. The first set of methods changes the training procedure for Decision tree (DT) ensembles to produce more resource-friendly models. This can be beneficial to achieve the highest accuracy given the computational resources provided, but often result in longer training times and more evolved training procedures. Common examples for this approach are pre- and post-pruning rules for trees (see, e.g. [43]) or the pruning of entire ensemble members [347, 449, 589, 739].

The second set of methods studies the realization of a given DT ensemble and its execution. This approach uses the ensemble as-is and, as such, does not affect the training. We will focus on this methodology in this contribution. Note that both methods can also be combined. For example, Van Essen et al. present in [679] a comprehensive study of different architectures for implementing Random Forests (RFs) on CPUs, FPGAs, and GPUs. Based on the CATE algorithm [586], the authors train an RF with DTs constrained by a fixed height. By fixing the tree-depth, the authors show a practical pipelining approach for executing DTs on CPUs, FPGAs, and GPUs.

Asadi et al. introduce different realization schemes of tree-based models in the context of learning-to-rank tasks [26]. They introduce two different realization schemes, which will be discussed in more detail later: the first one uses a while-loop to iterate over individual nodes of the tree, whereas the second approach decomposes each tree into its if-else structure. For the first realization, the authors also consider a continuous data layout (i.e., an array of *structs*) to increase data locality but do not directly optimize each realization. Also note that the authors mainly consider gradient-boosted trees. There, the individual trees are usually “weak” in a sense, that they are comparably small, as opposed to larger trees in RFs.

Also in the context of ranking models, Lucchese et al. present the QuickScorer algorithm for gradient boosted trees [162, 450]. In this approach, the authors discard the tree structure and decompose each tree into its comparisons. Then, they sort the comparisons of the entire ensemble according to the feature value and perform them one after another instead of traversing trees in a classical sense. To do so, they introduce a 2^{Δ}-dimensional bit vector, where Δ is the height of a tree in which the most significant bit (MSB) signifies the prediction leaf node of that tree. This way, the algorithm can reuse comparisons across all ensemble members while minimizing cache misses. In [452] the authors further enhance their method by adding vectorization over multiple examples for more efficient batch-processing. To mitigate the limitations of a fixed height, Ye et al. propose in [721] using an encoding scheme called epitome that decodes the bitvectors on the fly while preserving vectorization. We note that, while these methods usually offer a tremendous speed-up, they execute *all* possible comparisons in the entire ensemble in the worst case. Thus, they are especially effective for large ensembles of smaller trees commonly produced by gradient boosting algorithms.

Kim et al. present in [373] a realization for binary search trees using vectorization units on Intel CPUs and compare their realization against a GPU realization. The authors provide insight on how to tailor the realization to Intel CPUs by taking into account register sizes, cache sizes, and page sizes. Their work is specialized for Intel CPUs, and thus, it is not directly applicable for different CPU architectures. Lucchese and colleagues have already noticed, that many nodes are seldom visited [450]. Buschjäger and Morik formalize this observation in [110] by estimating the probabilities of specific paths during tree traversal. Based on this probabilistic view of model execution or inference, the authors consider different realization schemes for tree traversal and theoretically analyze their runtime. Note, however, that this model of computation remains at the software level and does not include the memory layout. Buschjäger et al. enhance this model in [108] by including the memory layout in their model. They show how to minimize cache misses and how different realizations affect the instruction and data cache differently for executing ensembles of large trees commonly found in RFs. We will now discuss this paper in more detail.

7.3.3 A Probabilistic View of DT Execution

We consider supervised learning problems, in which we infer a model $f : \mathbb{R}^d \to \mathcal{Y}$ from labeled training data $\{(\mathbf{x}_i, y_i) | i = 1, \ldots, N\}$ to predict the value $f(\mathbf{x})$ of new, unseen observations. For $\mathcal{Y} = \mathbb{R}$, we have a regression problem, for $\mathcal{Y} = \{0, 1, \ldots\}$, we have a classification problem.

Tree ensembles train a set of individual trees and combine their predictions to establish a joint model. In the classical Random Forest (RF) approach by Breiman [72], K DTs are trained using different samples of input features. Other RFs variations have been explored, such as those that train trees on samples of data (bagging) [71] or those that randomly generate splits for training [250]. Boosting [610] also frequently uses decision trees as their weak base learners, but trains them sequentially to correct each other.

A decision tree is a simple, tree-structured model that consists of inner nodes with two children and leaf nodes. Each inner node compares the feature value x_f of the current sample $\mathbf{x}$ against a threshold t where f and t are computed during tree training. Depending on the outcome of this comparison, either the left or the right child of this node is used until a leaf node is found. The leaf node stores a constant prediction value (e.g. the estimated class probabilities that fall into the leaf) which is then returned.

Our goal is to analyze the probability of performing a certain comparison while traversing a DT. Based on this analysis, we can decide for each tree, which realization and which data layout is best. Our notation is the following: each node receives a unique identifier (e.g., in breath-first order) i. We denote the left child of i with $l(i)$ and the right child with $r(i)$. Note that every observation takes exactly one path $\pi(\mathbf{x})$ from the root node to one leaf. To lighten the notation, we drop the argument $\mathbf{x}$, if we are not

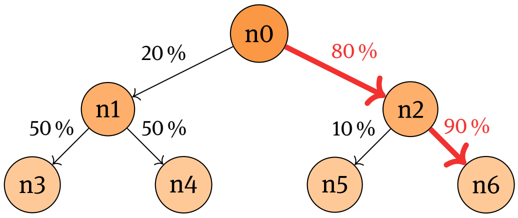


Fig. 7.12: Decision tree with probabilities of the path.

interested in the path of a specific observation. As established in [110], we model each comparison at node i as a Bernoulli experiment in which we take the path towards the left child with probability $p(i \to l(i))$ and towards the right child with $p(i \to r(i))$. It holds that $p(i \to l(i)) = 1 - p(i \to r(i))$. An example can be found in Figure 7.12.

The probabilities $p(i \to l(i))$ and $p(i \to r(i))$ can be estimated with the training data by counting the number of samples at each node i taking the left and right path. Assume a path of length L with $\pi = (i_1, i_2, \ldots, i_L)$, where i_{j+1} is either the left or the right child of the j^{th} node on the path. Following this path consists of a series of Bernoulli experiments, each with probability $p(i_j \to i_{j+1})$. Let $\mathcal{P}$ denote the set of all paths in the tree. The probability of taking path $\pi \in \mathcal{P}$ is given by

$$p(\pi) = p(i_0 \to i_1) \cdot \ldots \cdot p(i_{L-1} \to i_L) = \prod_{j=0}^{L} p(i_j \to i_{j+1}) \tag{7.13}$$

Again, let i be a node, there is exactly one path $\pi = (0, \ldots, i)$ ending in node i. We call the probability of the path leading to node i the probability of that node, that is $p(i) = p((0, \ldots, i))$. Let $\mathcal{T}$ be the set of all nodes in the tree. We define the probability for every subset of nodes $T \subseteq \mathcal{T}$ as:

$$p(T) = \sum_{i \in T} p(i) \tag{7.14}$$

7.3.4 Memory Locality and Tree Realization

As mentioned, tree ensembles can consist of millions of nodes that must be stored and managed in the main memory. Hence, the memory layout of tree ensembles is one of the most crucial aspects of efficient tree traversal. In order to mitigate the performance gap between the main memory and the processor, smaller and faster memory subsystems are often introduced in modern computer architectures to hide the long read/write latency, in the forms of cache and scratchpad memories. Here we focus on the cache memory, which is commonly equipped in modern computing systems.

The cache memory basically acts as a buffer between the main memory and the CPU and stores the data and instructions that the CPU uses more frequently. This way, frequently accessed parts of the memory can be loaded from the smaller, but much faster cache memory to reduce the latency of memory accesses. However, any misuse of cache memory might be even worse than no cache in the memory hierarchy because one cache miss triggers two loading instructions, one from the main memory to the cache and one from the cache to the processor. There are three types of cache misses [183]:

Compulsory misses are due to the first access to a memory block that the cache did not yet have a chance to buffer.

Capacity misses occur when some memory blocks are discarded from the cache memory due to the limited capacity, i.e., the program is working on more data than the cache capacity.

Conflict misses occur in set-associative or direct-mapped caches when several blocks are mapped to the same cache set.

The basic assumption of a cache is that of *memory localities*:

Temporal locality Recent data will be accessed in the near future, say, in small program loops.

Spatial locality Data at addresses close to the addresses of recently accessed data will be accessed in the near future, say, in sequential accesses to elements of an array.

These are the general assumptions for cache design, but please note that knowing how the caches exactly behave is difficult or even impossible. Caches are manufactured as parts of the closed IP of CPU manufacturers and hence the exact design of caches is unknown. Additionally, due to the fact that there are often competing processes running on a single CPU it is difficult to predict the cache behavior deterministically. In this contribution we suppose that the design of cache behaviors cannot be changed. The question we address is this: **How to realize a cache-friendly execution while preserving the functional behaviors of a given DT?**

First, we analyze the memory usage of two common realizations of DT, i.e., native Tree and If-else Tree that do not exploit the memory locality during the execution over the structure of DT. Then we discuss how we can make these two realizations more cache-friendly.

Native Tree The native tree implementation uses a while-loop to iterate over the individual tree nodes that are stored within a continuous data structure, say, in a one-dimensional array. An example code can be found in Listing 7.1. Although the usage of the simple loop with a few lines of codes preserves the temporal locality, the accesses over the nodes of a DT do not have spatial locality. The nodes are often allocated sequentially according to the indexes, whereas such indexes might not take the execution of the DT into consideration, e.g., the nodes on one path might not be allocated sequentially. In addition, if the distance between each node of the path is greater than the number of nodes that can be hosted into a cache set, some nodes will be loaded into caches but not used at all, leading to much *capacity and conflict cache misses*.

Listing 7.1: Example for native tree structure in C++.

```
struct Node {
  bool isLeaf;
  unsigned int prediction; // Predicted label
  unsigned char feature; // Targeted feature
  float split; // Threshold
  unsigned short leftChild, rightChild;
};
Node tree[] = {{0,0,0,8191,1,2},{0,0,1,2048,3,4},..]}
bool predict(short const x[3]){
  unsigned int i = 0;
  While(!tree[i].isLeaf) {
    if (x[tree[i].feature] <= tree[i].split) {
      i = tree[i].leftChild;
    } else {
      i = tree[i].rightChild;
    }
  }
  return tree[i].prediction;
}
```

If-Else Tree An alternative is the if-else tree, which statically encodes the split values of nodes in the instructions. This realization essentially avoids the indirect memory accesses required by the native tree and usually improves the runtime efficiency significantly. An example code can be found in Listing 7.2. However, the advantage of the temporal locality in the instruction cache might be completely abandoned. Since DTs are naturally composed of many branches, some encoded instructions might be prefetched into the instruction cache but not used. Additionally, if the size of the instructions for one DT is greater than the size of the instruction cache, the cached instructions may be evicted out by loading other instructions due to the *capacity and conflict cache misses*.

Listing 7.2: Example for if-else trees in C++.

```
bool predict(short const x[3]){
  if(x[0] <= 8191){
    if(x[1] <= 2048){
      return true;
    } else {
      return false;
    }
  } else {
    if(x[2] <= 512){
      return true;
    } else {
      return false;
    }
  }
}
```

7.3.5 Memory Layout Optimization

In the following, we analyze the caching behaviors of the two different realizations and present our tree-framing algorithms to optimize the memory layout at the application layer accordingly.

Native Tree As shown in Listing 7.1, a DT can be realized by allocating the tree nodes sequentially in an 1-D array, and a simple loop can access them according to the comparison between the feature and the split value. We first observe that, in fact, half of the nodes in a tree are leaf nodes storing a prediction value. This naive realization, however, assumes the same data type for each node, incurring unnecessary memory usage. Second, the access pattern of a DT forms a unique path from the root to a leaf for each input data, but the nodes are typically sequentially allocated in the array according to Breadth-First Search (BFS).[5] The distance between each accessed node becomes larger when the accessed nodes are placed deeper in the DT. The proposed optimization is twofold: 1) reducing compulsory cache misses by encoding the predicted label into the field of children, and 2) reducing capacity and conflict cache misses by allocating as many nodes as possible from the same path into the same cache set.

When a node is loaded, the following nodes in the array are prefetched into the data cache sequentially. If the size of memory for each node can be reduced, more nodes can be loaded into the cache at once so that overall compulsory cache misses can be reduced. To reduce memory consumption we can completely remove the `isLeaf`

5 Please note that the problem is not limited to BFS. Here we point out the demand of considering the access pattern when allocating nodes to memory.

and `prediction` fields, and store the predicted labels of the children directly in the respective fields by encoding the node type with an indicator field, i.e., removing one Boolean variable and two unsigned shorts by adding one unsigned short.

As mentioned earlier, the sequence of stored nodes is not consistent with the access pattern over the execution of the tree, so the benefit of caching cannot be utilized properly. A sensible solution is to leverage the probabilistic view on DT execution to identify nodes that were likely executed consecutively and place them in memory accordingly. Let τ be the cache set size and A be the array in which we place all nodes of $\mathcal{T}$. Furthermore, let $\mathcal{C}$ be the candidate list of nodes in $\mathcal{T}$ that have not been placed in A yet and let $\mathcal{S}$ denote the nodes that should be placed in the same cache set. For each node, we greedily choose a child that has the highest probability on the current path and place it in $\mathcal{S}$. Once $\mathcal{S}$ contains $\tau - 1$ elements (and hence is full), we append all nodes from $\mathcal{S}$ to the array A, clean up $\mathcal{S}$, and repeat the above procedure for the next set. The details are summarized in Algorithm 6.

```
Algorithm 6: Optimized path layout
  Data: Tree-nodes $\mathcal{T}$, maximum nodes per set $\tau$
  Result: A data array A with the path-oriented layout
1  A = [ ]
2  $\mathcal{C} \leftarrow \{0\}$
3  while $\mathcal{C} \neq \emptyset$ do
4      $i \leftarrow \arg\max_{j \in \mathcal{C}}\{p(\pi(j))\}$
5      $\mathcal{C} \leftarrow \mathcal{C} \setminus \{i\}$
6      $\mathcal{S} \leftarrow \{i\}$
7      while $|\mathcal{S}| \neq \tau$ do
8          if $i$ is leaf-node and $\mathcal{C} \neq \emptyset$ then
9              $i \leftarrow \arg\max_{j \in \mathcal{C}}\{p(\pi(j))\}$
10             $\mathcal{C} \leftarrow \mathcal{C} \setminus \{i\}$
11         else
12             $\mathcal{C} \leftarrow \mathcal{C} \cup \arg\min\{p(i \to l(i)), p(i \to r(i))\}$
13             $i \leftarrow \arg\max\{p(i \to l(i)), p(i \to r(i))\}$
14             if $|S| = \tau - 1$ then
15                 $\mathcal{C} \leftarrow \mathcal{C} \cup \{l(i), r(i)\}$
16         $\mathcal{S} \leftarrow \mathcal{S} \cup \{i\}$
17     A.append(S)
```

Please note that adding a new node to $\mathcal{S}$ (Line 7) has two possible actions for the encoding procedure:

- The current node is a split node. The algorithm picks the next node based on the children's probabilities and puts a more probable child in $\mathcal{S}$ and the other children into the candidate list $\mathcal{C}$.
- The checked node is a leaf node, i.e., the end of the path. The algorithm picks a sub-root with the highest probability from the candidate list $\mathcal{C}$ as long as it is not empty. The traversal starts again until $\mathcal{S}$ is full.

If the current $\mathcal{S}$ is full, but a path is not finished yet (Line 14), two children of the current node are returned to the candidate list $\mathcal{C}$ (Line 16). A sub-root that has the highest probability is picked from $\mathcal{C}$ for the next new set $\mathcal{S}$. The algorithm outputs the optimized memory layout over nodes in which path-oriented sets are sequentially allocated to the array.

If-Else Tree As shown in Listing 7.2, a DT can be realized by unrolling the comparisons of a DT into conditional statements with the if-else blocks. This version avoids the indirect memory accesses and does not consider the execution pattern of a DT. The proposed optimization is also twofold: 1) reducing compulsory cache misses by reducing the branch executions, and 2) reducing capacity and conflict cache misses by grouping those nodes used most of the time, e.g., the root node.

When a compulsory cache miss takes place, several consecutive instructions are fetched into the instruction cache, even though some of them might not be executed due to branches. An analysis of the corresponding assembly code reveals that only the branches for else statements are generated in general. In order to increase the chance of using prefetched instructions, the possibility of branch executions should be reduced. Towards this, we propose traversing all paths in the DT and swapping the children of every node i when $p(i \to l(i)) \geq p(i \to r(i))$.

Furthermore, unlike the native tree, the positions of unrolled nodes cannot be freely allocated. The size of nodes from a DT is likely greater than the size of the instruction cache. Because of the capacity and conflict cache misses the cached instructions may be evicted by fetching other instructions. We propose partitioning nodes into different computation kernel functions, and leveraging `goto` statements to break the tie between if-else blocks so that we can put probable nodes together.

Let $\mathcal{K}$ denote the kernel function and let $s(i)$ be a mapping function returning the instruction size of node i. We formulate the following optimization problem:

$$\mathcal{K} = \arg\max \left\{ p(T) \middle| T \subseteq \mathcal{T} \text{s.t.} \sum_{i \in T} s(i) \leq \beta \right\}, \tag{7.15}$$

where β is a given budget related to the size of the instruction cache on the targeted architecture. Given $\mathcal{K}$, these nodes likely remain in the cache once they are fetched, whereas the remaining nodes $\mathcal{L} = \mathcal{P} \setminus \mathcal{K}$ may be evicted more often. In order to avoid iterating over all possible subsets of $\mathcal{T}$, which might be computationally inefficient, we propose a greedy algorithm to partition nodes in a path-wise manner, summarized in

Algorithm 7. At first, the algorithm swaps the children according to their probabilities,

```
Algorithm 7: Optimized if-else tree
  Data: Tree $\mathcal{T}$, Paths $\mathcal{P} = \{\pi_1, \dots, \pi_M\}$
  Result: Kernel $\mathcal{K}$, Label $\mathcal{L}$
1 swapChildren($\mathcal{T}$)
2 $\mathcal{P} \leftarrow$ sortByProbabilities($\mathcal{P}$)
3 $b \leftarrow 0$
4 for $\pi \in \mathcal{P}$ do
5     for $i \in \pi$ do
6         if $b + s(i) > \mathcal{B}$ then
7             Add $i$ to $\mathcal{L}$
8         else
9             Add $i$ to $\mathcal{K}$
10            $b \leftarrow b + s(i)$
```

and sorts all paths in the tree by their probabilities. Afterwards, the approach greedily appends nodes one by one into $\mathcal{K}$ until the accumulated size of the added nodes b is greater than the given budget $\mathcal{B}$. The rest of the nodes are all added to $\mathcal{L}$. Once the nodes are grouped into $\mathcal{K}$ and $\mathcal{L}$, we can use `goto` statements to break the sequential generation of if-else blocks. First, we generate if-else blocks for all nodes in $\mathcal{K}$. Once the left/right child of one of those nodes is in $\mathcal{L}$, a `goto` statement is generated at the same position to replace the original if-else statement. Then, the corresponding if-else statements of this node and its children are all generated into a label block at the end, which is branched from the goto statement. Listing 7.3 shows an example based on Listing 7.2 by applying Algorithm 7.

Listing 7.3: If-else structure in C++ with goto statements.

```
bool predict(short const x[3]){
  if(x[0] > 8191){
    if(x[2] <= 512){
      return true;
    } else {
      return false;
    }
  } else {
    goto Label0;
  }
Label0:
  {
    if(x[1] <= 2048){
      return true;
    } else {
      return false;
    }
  }
}
```

The remaining question is how to estimate the instruction size $s(\cdot)$ of each node. In general, the instruction set size differs for two different types of nodes:

Split nodes require three types of instructions. First, the values of the target feature and the corresponding threshold are loaded into registers. Second, the values inside the registers are compared against constant values. Last, a jump out of the current block is performed based on the comparison.

Leaf nodes need two types of instructions. First, the return value of the prediction is stored in a register, and second, a jump back to the caller of the if-else tree is performed.

Therefore, we can estimate $s(\cdot)$ by counting the number of generated instructions for a tree node. Table 7.7 summarizes the expected size of instructions for ARM, X86 (Intel), and PPC in an isolated example.[6] Please note that in a real application, the actual number of instructions depends on the adopted compilation tool-chains and the actual realization. An advanced automation can be further explored by exploiting compiler features, e.g., annotations on the source code, to enforce the executing patterns. By doing so, the number of generated instructions can be firmed in the proposal algorithm as for example done in ongoing research such as [132].

6 We adopted GNU C++ (g++) compiler version 4.8.3 for ARM, version 4.9.2 for PPC, and version 5.4.0 for Intel with `-O0` option.

Tab. 7.7: The expected size of instructions for a split node and a leaf node in a decision tree on ARM (`Raspberry PI 2`), PPC (`NXP T4240 processors`) and Intel (`Intel Core i7-6700`) processors.

	ARM [Bytes]		**PPC [Bytes]**		**Intel [Bytes]**	
Type	Int	Float	Int	Float	Int	Float
Split	20	32	20	48	28	17
Leaf	8	8	8	8	10	10

7.3.6 Architecture-Aware Code Generator

As noted earlier, for each combination of tree ensembles and target hardware architecture a different implementation might offer the best inferencing solution. Hence, we implement the discussed tree-framing methods in a single code-generator framework that generates the optimized realizations for a given forest and target platform. Figure 7.13 gives an overview of the whole workflow. First, the pre-trained forest (in a `JSON` format) is loaded. Afterwards, the corresponding intermediate representation of the ML model is generated, and the proposed optimizations are performed, e.g., branch swapping, node re-indexing, etc. Finally, we provide a set of C-style templates that represent the specific implementation types (e.g. *native* or *if-else*). Several auxiliary scripts scripts are provided to automate the above procedures, e.g., selecting corresponding cross-compilers. Per default sci-kit learn models are targets [561] but other model definitions, in, say, the ONNX format are also supported. More details can be found at https://github.com/sbuschjaeger/fastinference.

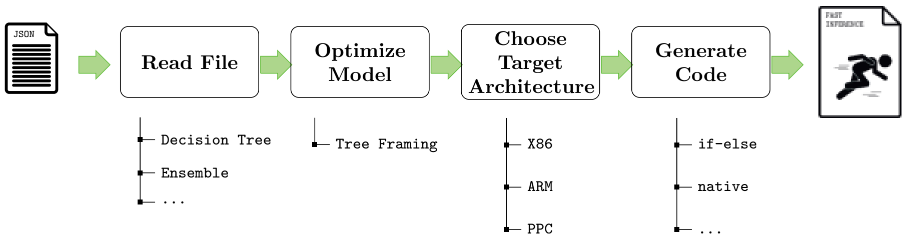


Fig. 7.13: Workflow of our code generator. The model configuration is loaded into an internal representation. If selected, optimizations are performed on the model before code generation. Afterwards, the target architecture and the appropriate templates are selected for final code generation.

7.3.7 Experimental Evaluation

We have performed 1800 different experiments by training Decision Trees (DT) [73], Random Forests (RF), [72] and Extremely randomized Trees (ET) [250] on 12 different datasets with varying tree-depths to generate the aforementioned realizations for different architectures, i.e., X86, PPC, and ARM CPUs. Table 7.8 shows the datasets we used during the experiments. All datasets are available in the UCI Machine Learning Repository [31] except for MNIST [420], IMDB [456], and FACT [17]. In addition to the number of features and the number of examples during test time, we also report the range of accuracy for the three different models DT, RF, and ET. In all experiments we used the CART algorithm with the Gini score criterion for node-splitting and trained models using the `sklearn` package[561]. For RF and ET, we used 25 trees. If the respective dataset comes with a pre-computed train/test split, we use this. Otherwise, we use 75 % of the data for training and 25 % of the data for testing. DTs often do not achieve high accuracy, whereas RF and ET perform best with large trees. We did not perform any hyperparameter optimization with respect to the classification accuracy and report the accuracy here to validate our code generator.

Since `sklearn` is arguably one of the most-used machine learning libraries we also compared its performance against our implementation. We found that, our realization is on average 500 – 1500 times faster than `sklearn`. However, we admit that this comparison is biased, because large parts of `sklearn` are written in Python and optimized for batch execution. Thus, we excluded these comparisons in the following discussion. For space reasons, we focus our evaluation on RF models, but found that DT and ET result in similar behaviors across all systems. We use the *naive native* realization as the baseline for all experiments, and measure the average speed-up for each dataset of each optimization against this realization. To minimize unfairness due to caching, we classify all samples in the test set twice, but only report the runtime of the second run. We repeat the whole process 50 times and report the average speed-up across these 50 repetitions.

For *native* optimizations, we choose $\tau = 25$ on X86, $\tau = 8$ on ARM, and $\tau = 8$ for the PPC CPU. For *if-else* optimizations, we use an instruction-cache size $\beta = 128\,000$ bytes on X86, $\beta = 32\,000$ bytes on ARM, and $\beta = 32\,000$ bytes on the PPC CPU. The experiments were performed on an `Intel Core i7-6700` desktop machine with 16 GB RAM for `X86`. For `PPC`, we use a `NXP Reference Design Board` with `T4240 processors` and 6 GB RAM. For `ARM`, we use an `Raspberry PI 2` with an `ARMv7` CPU and 1 GB RAM.

Experiments on the X86 CPU Architecture Figure 7.14 depicts the average speed-up of the four different optimizations on Intel. First we note, that the *if-else* trees are the fastest on Intel and offer a speed-up of around three across all tree depths. For smaller tree depths from 1 – 10, we see that optimizing *if-else* trees only offers marginal speed-up. However, for larger tree depths of around 15 and 20, we can see that optimized

Tab. 7.8: Summary of datasets for our experiments based on UCI datasets [31], IMDB [456], MNIST [420], FACT [17].

Dataset	# Examples	# Features	Accuracy
adult	8141	64	0.76 - 0.86
bank	10 297	59	0.86 - 0.90
covertype	145 253	54	0.51 - 0.88
fact	369 450	16	0.81 - 0.87
imdb	25 000	10 000	0.54 - 0.80
letter	5000	16	0.06 - 0.95
magic	4755	10	0.64 - 0.87
mnist	10 000	784	0.17 - 0.96
satlog	2000	36	0.40 - 0.90
sensorless	14 628	48	0.10 - 0.99
wearable	41 409	17	0.57 - 0.99
wine-quality	1625	11	0.49 - 0.68

if-else trees can retain their speed-up and outperform unoptimized *if-else* trees with a speed-up factor larger than 3.

Native trees do not perform as well as *if-else* trees on Intel CPUs. Overall, the speed-up compared with *naive native* trees is only marginal for smaller trees below depth 15. Here, both versions, i.e., the *StandardNativeTree* and the *OptimizedNativeTree*, offer a speed-up of 1.5 at most. Interestingly, for larger trees around depth 15 and more, we again notice that our optimizations improve performance.

Experiments on the PPC CPU architecture Figure 7.15 depicts the average speed-up of the four different optimizations on PPC. We can observe that the results here are similar to Figure 7.14, in which *if-else* trees always outperform *native* trees with a speed-up in the range from 2 – 5. Along with the increment of tree depth, the speed-up from both *if-else* tree versions drops. Un-optimized *if-else* trees suffer especially from degraded performance, dropping to almost 2, whereas the optimized version can retain a speed-up of around 3.5.

Similar to X86 CPU, the *native* realization does not seem to be the best choice as it provides a speed-up under 2 in all cases. However, with increasing tree depths, optimizations are more important. It is worth noting, that we can observe cases where the *native* trees outperform *if-else* trees when tree depth is larger than 15.

Experiments on the ARM CPU Architecture Figure 7.16 depicts the average speed-up of the four different optimizations on ARM. We observe that the situation on ARM is more fragmented than that of X86 and PPC. In general, we are able to achieve a speed-up of around 4 for small trees, which drops to around 2 – 3 for larger trees. Both realizations roughly start with the same speed-up factor for small trees, but then quickly diverge for tree depth from around 5 – 15. In this range of tree depth, we see that *if-else* trees are the

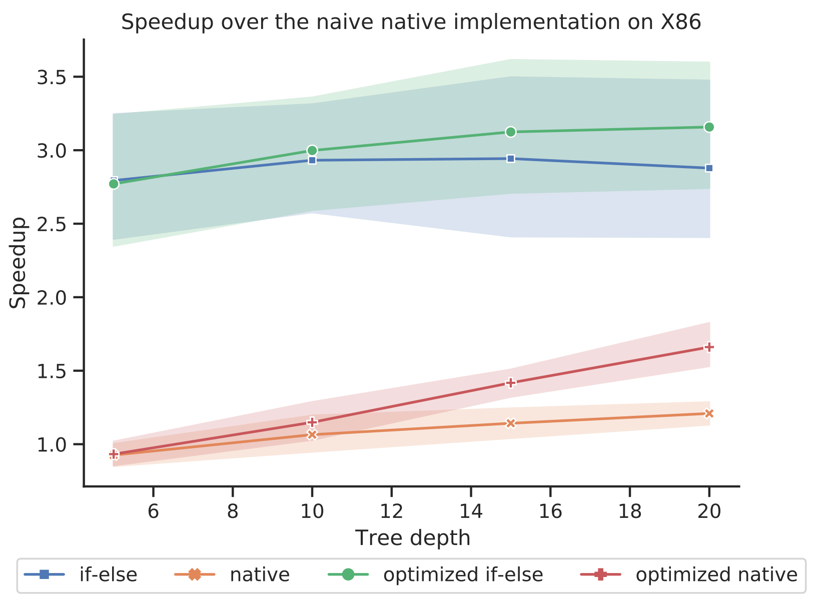


Fig. 7.14: Average speed-up factor for real-time execution compared with the naive native realization on Intel for tree depths from 1 – 20.

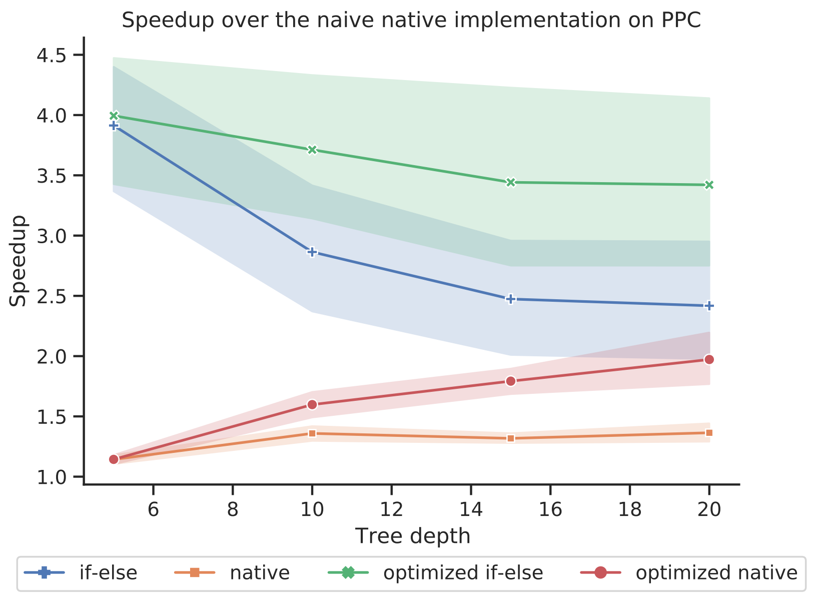


Fig. 7.15: Average speed-up factor for real-time execution compared to the naive native realization on PPC for tree depths from 1 – 20.

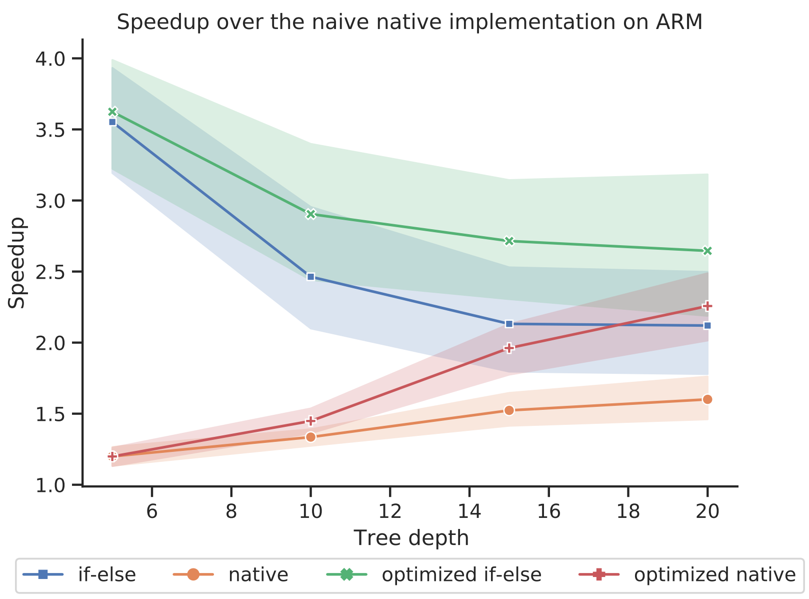


Fig. 7.16: Average speed-up factor for real-time execution compared with the naive native realization on ARM for tree depths from 1 – 20.

fastest choice on ARM. Additionally, we notice that with increasing tree depths cache optimizations become more important and consistently outperform their un-optimized counterpart. Once trees are sufficiently large, we see that the *native* trees match again the performance of *if-else* trees and even outperform them for tree depths of 15 and 20 in some cases. In this sense, the results are similar to what we have seen on the PPC architecture.

7.3.8 Discussion of the Experiments

The experiments show differences and similarities across the three architectures. Here, we want to discuss these phenomena in terms of the properties of the specific architectures, as well as the particular CPU models used for experiments. We note that one of the main architectural differences between X86, ARM, and PPC are the available instructions. Since *native* trees only use a small amount of hot-code, the differences between CPU architectures will likely not matter much here. However, while looking at *if-else* trees, we can expect a larger difference. To further investigate the interplay between

CPU architectures and code size, we consider Table 7.9, [7] which depicts the instruction size of a tree kernel function for varying tree depths over the FACT dataset (containing floating-point features) and the covertype dataset (containing integer features) under the standard *if-else* tree realization. For Intel CPUs, as shown in Figure 7.14, we notice

Tab. 7.9: The actual size of instructions for *if-else* tree executing kernel functions on different architectures with the O3 option.

(a) Kernel size with integer features for covertype dataset

DateType	DT-1	DT-5	DT-10	DT-15	DT-20
Intel	224	575	8185	51005	167644
PPC	232	604	7732	51840	170772
ARM	204	604	9040	55012	180628

(b) Kernel size with floating point features for fact dataset

DateType	DT-1	DT-5	DT-10	DT-15	DT-20
Intel	96	415	17023	127330	404722
PPC	96	556	20996	169696	577952
ARM	88	428	18436	154992	542020

that *if-else* trees are the best choice. There are mainly two reasons. First, X86 CPUs are Complex Instruction Set Computers (CISC) offering a very rich set of instructions that include all sorts of specialized operations. Since *if-else* trees unroll the complete tree structure into instructions, they give the compiler the opportunity to utilize this multitude of instructions to the fullest by encoding larger parts of the tree in single instructions. From Table 7.9 we can also see that the Intel CPU almost always requires the fewest instructions per decision tree. Second, in our experimental setting, the `Intel Core i7-6700` CPU has a comparably large instruction cache of 256 KiB combined with two larger shared caches of 1 MiB (L2 Cache) and 8 MiB (L3 Cache). Thus, by encoding a single tree in only a few instructions, it is likely to fit it into the larger instruction cache. By contrast, *native* trees do not benefit from the CISC architecture and require additional space in the data cache by encoding the tree nodes as data instead of instructions.

As with the X86 architecture, we have seen that *if-else* trees perform very well on the PPC architecture, but to a lesser extent. The PPC CPU architecture is a Reduced Instruction Set Computer (RISC) with performance enhancement for high performance computing. RISC does not offer instructions for specialized operations as CISC does.

7 Although the instructions generated by the compiler may differ due to aggressive compiler optimization (O3) compared with the presented node sizes (O0 optimization) in Table 7.7, the code generator at the end selects the O3 option to accelerate the realizations as much as possible.

Thus, the compiler must largely rely on the combination of (comparably) simple instructions to implement *if-else* trees. This, in turn, results in larger code that is less likely to fit into the instruction cache. Comparing the instruction size of PPC with X86 in Table 7.9 we see that the PPC architecture indeed requires more instructions than with X86. Interestingly, this case is less severe for integer features, due to the enhancements in this instruction set architecture. Considering the cache sizes of the T4240 processors used in the experiments, we see that it only has a 32 KiB instruction cache, but also comes with a 2 MiB shared L2 cache, which is even larger than the Intel Core i7-6700 CPU. For smaller trees of around 5 – 10, the cache sizes are still large enough to hold all trees, and thus *if-else* trees are still the fastest choice. If trees become large (depth 10 or more), the instruction cache is not enough to hold all trees and we must rely on the larger L2 cache. However, this cache is slower, which in combination with the larger code size explains the performance drop for larger trees.

Finally, we discuss the fragmented behavior of the ARM architecture. Much like its PPC counterpart, ARM also uses a RISC architecture. However, ARM's RISC does not come with specialized instructions for high-performance computing, and thus the compiler has to completely rely on the combination of simple instructions for *if-else* realization. This in turn results in even larger code for integer features, which is less likely to fit into the instruction cache as shown in Table 7.9. Interestingly, for floating-point features, we see that the ARM CPU uses fewer instructions than the PPC CPU, which is attributable to the specific CPU model used during experiments. The T4240 processors are optimized for high-performance computing in a low-power embedded computing setting, such as networking applications, and thus are optimized for integer operations. By contrast, the ARMv7 CPU of the Raspberry PI 2 is a general-purpose CPU aimed at the needs of the average user, and thus it places a larger emphasis on floating-point operations compared with the T4240 processors. It has a 32 KiB instruction cache in combination with a significantly smaller 512 KiB L2 shared cache. Compared with the other CPUs, this means that the ARM CPU has 2 – 16 times less L2/L3 cache available. For smaller trees around a depth of 5 – 10, the cache sizes are still enough to hold all trees, and thus *if-else* trees are still the fastest choice. For larger tree depths, however, the instruction cache is not enough and *native* structures using the data cache become faster. However, since the data cache is also small, both caches are filled quickly to their maximum. Interestingly, if we optimize both *if-else* and *native* trees, we end up with roughly the same performance.

7.3.9 Conclusion

DTs form one of the building blocks of modern machine learning and ensembles of decision trees are one of the most successful classifiers regularly achieving state-of-the-art performance in real-world applications. DTs are generally regarded as 'simple' classifiers that can be executed even on the tiniest of hardware. However, a tree easily contains up to millions of decision nodes that must be stored and managed which can be a challenge even for large server hardware. Cache memory is commonly adopted in today's von-Neumann computing architecture to hide the long latency between the main memory and the processor. Hence, an efficient realization of a given tree ensemble must respect this memory hierarchy and provide a suitable memory layout of the decision nodes for optimal performance. In every modern programming language there are at least two ways to implement a DT: either one decomposes the tree into its if-else structure or one uses a while-loop to iterate over a continuous array of nodes. Both approaches offer different caching behaviours that can be further enhanced by the tree-framing methods discussed in this contribution. At the core of these methods lies the fact that DTs do not have a deterministic runtime, but its execution time may vary depending on the current sample. Hence, a probabilistic view of DT execution estimates the most probable paths of the tree and frames the tree so that these paths are likely to remain in the cache. The experimental evaluation shows a speed-up around 2 – 4 across three different hardware architectures on a variety of datasets without any loss in accuracy occurs.

8 Communication Awareness

The ubiquity of connected devices and parallel computing platforms challenges efficient and reliable execution of machine learning algorithms. If machine learning workloads are executed merely locally, a system does not always have sufficient resources at its disposal to perform the necessary operations fast enough. Furthermore, at a smaller scale, multiple hardware components these days are interconnected via on-chip or off-chip networks to create many-core systems. Communication, synchronization, and offloading have thus become essential in designing embedded systems under communication and resource constraints.

This chapter presents (1) the timing predictability of embedded systems and (2) the communication architecture in heterogeneous CPU/GPU environments. Synchronization with resource sharing, communication with potential failures, and probabilistic timing information are presented in Section 8.1. Bandwidth limitations of different execution models and coprocessor-accelerated optimization are presented in Section 8.2.

https://doi.org/10.1515/9783110785944-008

8.1 Timing-Predictable Learning and Multiprocessor Synchronization

Kuan-Hsun Chen
Junjie Shi

Abstract: With the increasing demand for time-predictable machine learning applications, e.g., object detection in autonomous driving systems, such a trend poses several new challenges for resource synchronization in real-time systems, especially when hardware accelerators like Graphics Processing Units (GPUs) are considered as shared resources. When the shared resources have relatively high utilization, conventional synchronization mechanisms might result in performance downgrade.

We thus propose the emerging Dependency Graph Approach (DGA), where the precedence constraints of all the computation segments are pre-proceeded. Such a non-work-conserving approach can schedule long critical sections, which may be even longer than the period of another task. This is not the case in all the other work-conserving protocols typically in use. Throughout numerical experiments, we show that DGA outperforms all the other conventional protocols in all the evaluated configurations when shared resources are highly utilized.

Additionally, a system does not always have sufficient resources at its disposal to perform the necessary operations fast enough if machine learning workloads are executed merely locally. One sound approach is to offload heavy workload to powerful remote servers and expect the inference outcome can be received in time. However, since this approach highly depends on network connectivity and responsiveness, typically only non-critical tasks are offloaded, whose timing requirements are less strict than those of critical tasks. Against such a pessimistic design, we present two novel offloading protocols that offload both critical and non-critical tasks. They handle uncertain connections while providing certain timing guarantees.

To achieve a timing-predictable design, typical timing analyses always consider the worst-case execution pattern to derive timing guarantees. But this approach is often too restrictive for some machine learning applications with soft timing constraints. To mitigate the pessimism, we develop several timing analyses of the probability of deadline misses and the deadline miss rate, two important metrics considered in the literature to quantify timeliness.

8.1.1 Introduction

Under the von Neumann programming model, shared resources that require mutually exclusive accesses, e.g., shared files, data structures, etc., have to be protected by applying synchronization (e.g., *binary semaphores*) or locking (e.g., *mutex locks*) mechanisms. Protected code segments that have to access shared resource(s) mutually exclusively are called *critical sections*. For uni-processor real-time systems, longstanding protocols developed in the 90s, are the current state of the art. These are namely the Priority Inheritance Protocol (PIP) and the Priority Ceiling Protocol (PCP) by Sha et al. [623], as well as the Stack Resource Policy (SRP) by Baker [37].

Along with the development of multiprocessor platforms, multiprocessor resource synchronization and locking protocols have been proposed and extensively studied. These include Distributed-PCP (DPCP) [592], Multiprocessor PCP (MPCP) [591], Multiprocessor SRP (MSRP) [238], Flexible Multiprocessor Locking Protocol (FMLP) [58], $O(m)$ Locking Protocol (OMLP) [69], and Multiprocessor resource sharing Protocol (MrsP) [101].

However, the performance of aforementioned protocols highly depends on 1) how the tasks are partitioned and prioritized, 2) how the resources are shared locally and globally, and 3) whether a job/task being blocked should spin or suspend itself. In the literature, conventional synchronization mechanisms might result in performance downgrade, since most of them are designed for sporadic tasks with relatively low utilization for critical sections, which are often not able to represent emerging heavy-loaded machine learning applications. We thus propose a novel concept called DGA, which can serve high utilization of critical sections well.

Moreover, when the workload of a critical section, e.g., a machine learning workload on GPU, is extremely high, a system does not always have sufficient resources at its disposal to perform the necessary operations fast enough. A sound solution is to offload heavy workload to powerful remote servers and wait for the outcome of the inference processes. However, the performance and stability of this approach highly depends on the quality of the network. To improve flexibility, we propose several adaptive protocols to ensure that the timing requirements of safety- and mission-critical tasks are not violated even in the case of connectivity issues while obtaining the benefits of offloading computation shares.

Last but not least, to achieve a timing-predictable design, conventional timing analyses always focus on the worst-case execution pattern to derive hard timing guarantees. However, such analyses are sometimes too pessimistic when systems can accept rare deadline misses, e.g., for soft real-time systems. Limited deadline misses on many machine learning applications might only lead to performance degradation, e.g., for image and voice recognition on smart edge devices. Some end users might only feel inconvenienced without further serious consequences. However, people might still wonder how resilient the considered system is with respect to deadline misses in a probabilistic argument. To obtain the probability of deadline misses, we innovate a well-known convolution-based approach based on multinomial distribution, and adopt several concentration inequalities to derive analytical upper bounds to further improve the efficiency of calculation.

Overall, the presented contributions in this chapter are as follows:

Dependency Graph Approach (DGA) We propose a novel method for periodic real-time task systems, which predestines the sequence where tasks can access resources. The conducted numerical simulations show that DGA outperforms all state-of-the-art approaches in most evaluated configurations, especially when utilization of critical sections is relatively high.

Offloading protocols for unreliable connection We present two offloading protocols that offload both critical and non-critical tasks. They deal with uncertain connections while providing certain timing guarantees. A case study on a robotic system demonstrates the applicability of the proposed protocols under various configurations.

Deadline-miss analyses A novel approach based on the multinomial distribution is proposed that calculates the deadline miss probability with drastically better analysis runtime without any precision loss. Furthermore, we propose several analytical bounds based on various concentration inequalities. The evaluation shows that our approaches are applicable for significantly larger task sets while preserving the quality of derived results, compared with conventional convolutional-based approaches.

8.1.2 Related Work

For multiprocessor systems, many resource synchronization and locking protocols are extensions of these aforementioned well-known uni-processor protocols. For example, Rajkummar et al. [592] proposed DPCP, where each resource is assigned on a processor statically, and critical sections are executed on the corresponding processor where the requested resource is assigned on. The extension MPCP [591] enables tasks to execute their critical section locally. In order to minimize the usage of stack memory in real-time systems, Gai et al. [238] proposed MSRP. Block et al. [58] introduced FMLP, where resources are divided into two groups, i.e., long and short. For short resources, critical sections are executed in a non-preemptable manner and tasks spin on their processors while waiting for resources. For long resources, tasks suspend themselves into a First In First Out (FIFO) queue while waiting. Brandenburg and Anderson [69] proposed OMLP, which ensures $O(m)$ maximum pi-blocking for any task set. Burns et al. [101] proposed MrsP, which allows tasks to progress other tasks that have occupied the same requested resource, in order to reduce the blocking time. A comprehensive survey of multiprocessor real-time locking protocols can be found in [68].

Besides fully relying on local computational power, offloading computation to remote servers is a reasonable solution to ease the pressure of resource constraints on embedded systems. In 2012, a cloud-assisted system for autonomous driving was firstly studied by Kumar et al. [406]. In 2015, Esen et al. [203] presented a software architecture named *Control as a Service* in which all control functions are completely moved to the cloud. In 2018, Adiththan et al. [4] proposed an adaptive offloading technique for control applications that makes all offloading decisions online based on a network performance monitor. Recently, Al Maruf and Azim [469] proposed a strategy for task offloading in multiprocessor mixed-criticality systems with dynamic scheduling policies under overload conditions. For real-time systems that allow offloading, one concept for modeling this particular local system view is *self-suspension* [127]. One of the state-of-the-art models can be applied such as the dynamic self-suspension model (e.g. [324], [442]), the segmented self-suspension model (e.g. [611]), or a hybrid model, e.g. [84]. For a detailed overview, see [127, 128].

To safely derive probabilistic timing guarantees, which enable a tradeoff between system safety and hardware costs, several techniques have been developed in the literature. Diaz et al. [177] developed a framework for calculating the deadline miss probability based on convolution for periodic task systems. In addition, Tanasa et al. [657] used the Weierstrass Approximation to approximate any arbitrary execution time distributions and applied a customized decomposition procedure to search all the possible combinations. However, the two approaches can derive only the probability of deadline misses with 7 and 25 jobs in the hyper-period, respectively. For sporadic real-time task systems, in which two consecutive jobs of a task do not have to be released periodically, Axer et al. [27] proposed evaluating the response-time distribution and iterating over the activations of job releases for non-preemptive fixed-priority scheduling. Maxim et al. [476] provided a probabilistic response time analysis by assuming a probabilistic minimum inter-arrival as well as probabilistic worst-case execution times (WCETs) for the fixed-priority scheduling policy. Ben-Amor et al. [46] extended the probabilistic response time analysis in [476] by considering precedence-constrained tasks. These convolution-based approaches are in general not scalable due to the huge number of jobs in the interval of interest.

8.1.3 Dependency Graph Approach

In this subsection, the dependency graph approach is presented in detail, including the primary design of DGA, the extension for supporting periodic task systems, and the corresponding scheduling algorithms.

8.1.3.1 Primary Design of DGA

We consider a set of n *frame-based* real-time tasks $\mathbf{T} = \{\tau_1, \ldots, \tau_n\}$ that is scheduled on M identical (homogeneous) processors. Each task is described by $\tau_i = ((C_{i,1}, A_{i,1}, C_{i,2}), T_i, D_i)$. The given tasks release their jobs at the same time and have the same period and relative deadline. Specifically, each task τ_i releases a job (at time 0 for notational brevity) with the following properties:

- $C_{i,1}$ is the execution time of the first non-critical section of the job.
- $A_{i,1}$ is the execution time of the (first) non-nested critical section of the job, in which a binary semaphore or a mutex $\sigma(\tau_{i,1})$ is used to control the access to the critical section.
- $C_{i,2}$ is the execution time of the second non-critical section of the job.

A sub-job is a critical section or a non-critical section. Therefore, each job of task τ_i has three sub-jobs. We assume the task set $\mathbf{T}$ is given and a constrained deadline is considered, i.e., $D_i \le T_i$. We also make the following assumptions:

- For each task τ_i in $\mathbf{T}$, $C_{i,1} \ge 0$, $C_{i,2} \ge 0$, and $A_{i,1} \ge 0$.
- The execution of the critical sections guarded by one binary semaphore s must be sequentially executed under a total order. That is, if two tasks share the same semaphore, their critical sections must be executed one after another without any interleaving.
- The execution of a job cannot be parallelized, i.e., a job must be sequentially executed in the order of $C_{i,1}, A_{i,1}, C_{i,2}$.
- There are in total Z binary semaphores.

The dependency graph approach consists of the following two steps:

- In the first step, a directed graph $G = (V, E)$ is constructed. A subjob (i.e., a critical or a non-critical section) is a vertex in V and the edges in E describe the precedence constraints of these jobs. The subjob $C_{i,1}$ is a predecessor of the subjob $A_{i,1}$, and $A_{i,1}$ is a predecessor of the subjob $C_{i,2}$. If two jobs of τ_i and τ_j share the same binary semaphore, i.e., $\sigma(\tau_{i,1}) = \sigma(\tau_{j,1})$, then either the subjob $A_{i,1}$ is the predecessor of $A_{j,1}$ or the subjob $A_{j,1}$ is the predecessor of $A_{i,1}$. All the critical sections guarded by a binary semaphore form a chain in G, i.e., the critical sections of the same binary semaphore follow a total order. Therefore, we have the following properties in set E:
  - The two directed edges $(C_{i,1}, A_{i,1})$ and $(A_{i,1}, C_{i,2})$ are in E.
  - Suppose that $\mathbf{T}_k$ is the set of tasks that require the same binary semaphore s_k. Then, the $|\mathbf{T}_k|$ tasks in $\mathbf{T}_k$ follow a certain total order π such that $(A_{i,1}, A_{j,1})$ is a directed edge in E when $\pi(\tau_i) = \pi(\tau_j) - 1$.

  Figure 8.1 provides an example of a task dependency graph with one binary semaphore. Since there are Z binary semaphores in the task set, the task dependency graph G has in total Z connected subgraphs, denoted as $G_1, G_2, \ldots, G_Z$. In

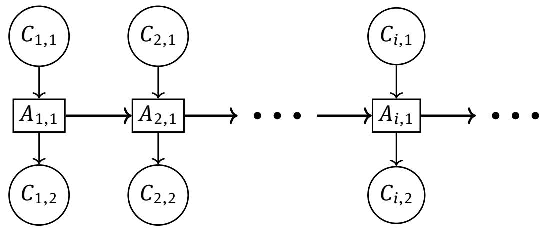


Fig. 8.1: A task dependency graph for a task set with one binary semaphore.

each connected subgraph G_ℓ, the corresponding critical sections of the tasks that request critical sections guarded by the same semaphore form a chain and have to be executed sequentially. For example, in Figure 8.1, the dependency graph forces the scheduler to execute the critical section $A_{1,1}$ prior to any of the other three critical sections.

– In the second step, a corresponding schedule of G on M processors is generated. The schedule can be based on system restrictions or user preferences, i.e., it can be based on either preemptive or non-preemptive schedules, or on either global, semi-partitioned, or partitioned schedules.

Algorithms to Construct G The objective of constructing dependency graph, i.e., G, is to minimize the makespan, i.e., the latest finishing time of all tasks, with the assumption that the number of *virtual processors* is the same as the number of tasks, based on uni-processor non-preemptive scheduling. For each task, $C_{i,1}$ is considered as release time r_i, and $C_{i,2}$ is considered as delivery time. There are several existing algorithms to derive good approximations of $G^\star$, where $G^\star$ is the graph with the optimal makespan: 1) the **extended Jackson's rule** [289], which is a polynomial-time algorithm with 2-approximation [377]; 2) the **Potts** [583], which is a polynomial-time 1.5-approximation algorithm [289]; 3) and the improvement of the approximation ratio to 4/3 by Hall and Shmoys [289].

8.1.3.2 Extension to Periodic Task Systems

To increase the applicability, we extend the DGA to handle multiprocessor synchronization for *periodic* real-time task systems. That is, we unroll the jobs of all the tasks in one hyper-period and then construct a dependency graph of these jobs. Suppose that the hyper-period H of a task set is the least common multiple (LCM) of the periods of all the tasks in this set. For each task τ_i that requests (at least) one resource, we create H/T_i jobs of task τ_i. For the ℓ-th job of task τ_i, we set its release time to $(\ell-1)T_i$ and its absolute deadline must be no later than $(\ell-1)T_i + D_i$. Since the jobs for one task should not have any execution overlap with each other, we only need one virtual processor or dedicated shop for them, but the release time constraint is added for each job. The three methods in Section 8.1.3.1 can still be applied by adding the release time constraint for each job. Afterward, a dependency graph for all the jobs in one hyper-period is generated. In the end, the schedules are generated offline. And the generated schedules will be repeated in the upcoming hyper-periods.

Please note that such an extension can be applied to any periodic real-time task system but that it comes at the cost of space and computation, due to the increasing number of jobs in one hyper-period.

8.1.3.3 Scheduling Algorithms

In the following, we show three scheduling algorithms for the same dependency graph(s) under different system specifications.

List-EDF Here, we show how to schedule the unrolled dependency graphs over the hyper-period. For the ℓ-th job of τ_i, J_i^ℓ has three subjobs $J_{i,1}^\ell, J_{i,2}^\ell, J_{i,3}^\ell$ that represent the related subjobs $C_{i,1}, A_{i,1}, C_{i,2}$, respectively. The release time of the first subjob is $J_{i,1}^\ell$ is $(\ell-1)T_i$, and the absolute deadline of the last subjob $J_{i,3}^\ell$ is $(\ell-1)T_i+D_i$. Regarding the release times of the second and third subjob, we initially set the earliest possible time the job may be released based on the WCETs of the other subjobs. Meanwhile, regarding the deadline of the first and second subjob, we initially assign the latest possible time the subjob can finish while still allowing schedulability. To be precise, the release time of $J_{i,2}^\ell$ is set to $(\ell-1)T_i + C_{i,1}$, the release time of $J_{i,3}^\ell$ is set to $(\ell-1)T_i + C_{i,1} + A_{i,1}$, the absolute deadline of $J_{i,2}^\ell$ is set to $(\ell-1)T_i + D_i - C_{i,2}$, and the absolute time of $J_{i,1}^\ell$ is set to $(\ell-1)T_i + D_i - C_{i,2} - A_{i,1}$.

We assume that each dependency graph $\mathbf{G}_s$ for a binary semaphore s is constructed for the corresponding jobs released (strictly) within one hyper-period H. If $H_s < H$, then $\frac{H}{H_s}$ copies of $\mathbf{G}_s$ are applied in a consecutive order to represent the precedence constraints of the critical sections. For notational brevity, we denote $r_{i,j}^\ell$ as the release time of the subjob $J_{i,j}^\ell$ and $d_{i,j}^\ell$ as the absolute deadline of $J_{i,j}^\ell$. If the absolute deadline of an immediate predecessor of $J_{i,j}^\ell$, denoted as $IPre(J_{i,j}^\ell)$, is larger than $d_{i,j}^\ell$, the absolute deadline of the immediate predecessor should be reassigned to $d_{i,j}^\ell$ minus the WCET of $J_{i,j}^\ell$. This is a standard procedure for scheduling jobs subject to release dates and precedence constraints. Details can be found in [36].

We assume that the absolute deadline assignment is adjusted accordingly so that $d_{i,j}^\ell$ for the subjob $J_{i,j}^\ell$ is always greater than the absolute deadline of $IPre(J_{i,j}^\ell)$. Scheduling $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_z$ on M homogeneous (identical) processors is a special case of the classical scheduling problem $P|prec; r_j|L_{\max}$, i.e., scheduling a set of jobs with specified release times and precedence constraints on M identical processors, minimizing the maximum lateness. One possible scheduling strategy is to use the List scheduling developed by Graham [269] in combination with Earliest Deadline First scheduling (EDF). A List schedule works as follows: Whenever a processor idles and there are subjobs eligible to be executed (i.e., all of their predecessors in the dependency graph have finished), one of the eligible subjobs is executed on the processor. If more subjobs than processors are available, we prioritize the subjobs that have the earlier absolute deadlines. If two subjobs have the same absolute deadline, the one with the larger remaining workload has a higher priority. We call this scheduling algorithm List-EDF.

Federated-Based Partitioning Algorithm Federated scheduling was proposed by Li et al. [430] in order to schedule parallel real-time task systems with internal precedence constraints that can be modeled as a Directed-Acyclic Graph (DAG). The foremost intention of this scheduling algorithm is to provide provably good approximations with respect to an optimal scheduling algorithm while considering implementation constraints, e.g., cache hit-rates and memory accesses during runtime. The idea of federated scheduling is to assign DAGs (in our case the DAGs resulting from the dependency graph construction) that need to utilize more than one processor (so-called *heavy* graphs) to those processors exclusively. Analogously, the graphs that can be feasibly scheduled on a single processor are denoted as *light* graphs and are scheduled jointly on the remaining processors, i.e., non-exclusively allocated processors. After this initial partition, the actual scheduling is done by a work-conserving scheduler on the assigned processors. If the graphs in both the *heavy* group and the *light* group can be scheduled feasibly, the corresponding partition is returned. Otherwise, there is no feasible partition.

Worst Fit-Based Heuristic In addition, a worst-fit heuristic is proposed in which the tasks are partitioned one by one. The tasks are first sorted according to a sorting strategy. After that, they are partitioned to the available processors using a worst-fit strategy, i.e., each task is assigned to the processor with the currently lowest utilization. Again, Partitioned-EDF (P-EDF) scheduling is applied to verify whether the resulting partition on M processors is feasible.

We proposed two sorting strategies: 1) sort all the tasks decreasingly with regard to the tasks' utilization, no matter which resources they request; 2) sort the graphs decreasingly with regard to the graph utilization and then sort the tasks inside each graph decreasingly with regard to the task utilization. In our proposed heuristic, both sorting strategies are applied. If the partition P generated by the first sorting strategy is not applicable, i.e., if the task set is not schedulable on M processors based on the current partition P using P-EDF, the second sorting strategy and the resulting partition $P^{'}$ are considered, and P-EDF is applied to verify the new partition $P^{'}$ once again. The algorithm only returns infeasible when both aforementioned sorting strategies cannot generate a schedulable partition. Otherwise, the task set is schedulable and the partition is returned. Again, if a time-driven schedule is created, the schedule can be returned as well.

8.1.3.4 Evaluation

We randomly generated task sets based on the number of processors M, shared resources Z, and relative utilization of the critical sections H as parameters. In our evaluation, we considered $M \in \{4, 8, 16\}$, $Z \in \{4, 8, 16\}$, and $H \in \{[5\,\% - 10\,\%], [10\,\% - 40\,\%], [40\,\% - 50\,\%]\}$.

For a given configuration of M, Z, and H, we generated task sets with $10 \times M$ tasks for each total utilization value $\sum_{\tau_i \in \mathbf{T}} U_{\tau_i} \in [0, M]$ with a step 5 %, applying the RandFixedSum method [199]. We enforced the total utilization $U_{\tau_i} \leq 0.5$ for each task τ_i. To determine the subtask utilization of one task, i.e., $U_{C_{i,1}}$, $U_{C_{i,2}}$, and $U_{A_{i,1}}$, we first decided the utilization of the critical section $U_{A_{i,1}}$ by randomly drawing a percentage of the task's total utilization U_{τ_i} based on the parameter H. Next, the remaining utilization U_{C_i} was split by drawing $U_{C_{i,1}}$ randomly uniform from $[0, U_{C_i}]$ and setting $U_{C_{i,2}}$ to $U_{C_i} - U_{C_{i,1}}$. The resource that each critical section of a task requests was selected randomly from all the available resources. In addition, we generated two kinds of task sets according to their settings of available periods:

Periodic task sets with semi-harmonic periods The task periods T_i are selected randomly from a set of semi-harmonic periods, i.e., $T_i \in \{1, 2, 5, 10\}$, which is a subset of the periods used in automotive systems [86, 290, 392, 606, 662].

Frame-based task sets As a special case of periodic task sets, all the tasks have the common period 1. Hence, i.e., $C_{i,1} = U_{C_{i,1}}$, $A_{i,1} = U_{A_{i,1}}$, and $C_{i,2} = U_{C_{i,2}}$.

For each of these setting of periods, 54 configurations are considered in total. For each of the utilization step values, 1000 task sets were randomly generated.

Evaluated Approaches To construct the dependency graphs, `POTTS` [583] is applied. Other evaluated methods to schedule the tasks sets were: 1) `FED-P-EDF`: the algorithm based on federated scheduling; 2) `WF-P-EDF`: the algorithm based on global worst-fit partitioning; 3) `LIST-EDF`: the List schedule based approach; 4) `ROP-FP`: Resource-Oriented Partitioned under Fixed-Priority [82]; 5) `ROP-EDF`: ROP under Dynamic-priority; 6) `LP-GFP-FMLP` [58]; 7) `LP-GFP-PIP` [194]; and 8) `GS-MSRP` [704].

Evaluation Results Only a subset of the results is presented, as the other results show similar trends. The evaluation results for periodic task systems are shown in Figure 8.2. If the workload of the critical sections is increased (Figure 8.2-(a) to (c)), the performance of all methods is reduced, and the difference between methods is decreased as well. The reason is that, when $\beta = [40\,\% - 50\,\%]$, the execution time of the critical section for tasks with period 10 time units can be large, i.e., longer than 2 time units. Therefore, tasks with period 1 time unit directly miss the deadline by default for all other approaches, no matter what kind of partitioning algorithm is applied. The performance drops down quickly when the utilization is increased and the critical section workload is large, as shown in Figure 8.2 (c).

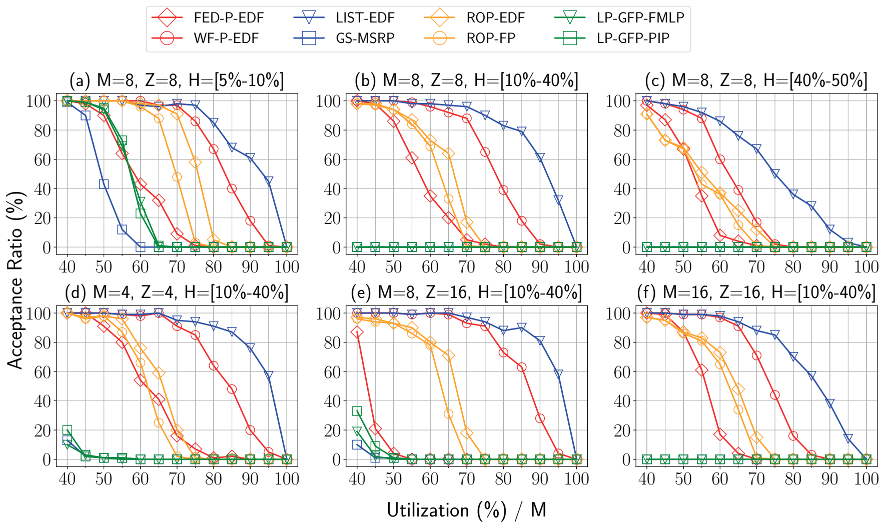


Fig. 8.2: Schedulability of different approaches for periodic task sets.

The evaluation results for frame-based task systems are shown in Figure 8.3. The proposed worst-fit heuristic `WF-P-EDF` outperforms `ROP-EDF` and other partitioned scheduling methods significantly. Furthermore, Figure 8.3 shows that `WF-P-EDF` has a good performance compared with `LIST-EDF`. In most cases, both `LIST-EDF` and `WF-P-EDF` can reach a 100 % acceptance ratio even with a 95 % utilization per processor.

8.1.4 Offloading Protocols for Unreliable Connection

In this subsection, two offloading protocols are presented in detail, addressing two system requirements: 1) the *service protocol*, which provides as much service for non-critical tasks as possible at any point in time, and 2) the *return protocol*, which allows a fast return to normal system behavior in the case of an unsuccessful offloading operation.

8.1.4.1 System Model

We consider a cyber-physical system comprising a set of tasks $\mathcal{T}$ that can be divided into two subsets with different requirements, namely, the set of *critical* tasks $\mathcal{T}_{crit}$, and the set of *non-critical* tasks $\mathcal{T}_{non}$, such that $\mathcal{T} = \mathcal{T}_{crit} \cup \mathcal{T}_{non}$ and $\mathcal{T}_{crit} \cap \mathcal{T}_{non} = \emptyset$. While for each $\tau_k \in \mathcal{T}_{crit}$ timing constraints must be satisfied at any point in time, for each $\tau_k \in \mathcal{T}_{non}$ timing violations may be unpleasant but not hazardous. According to the classification of tasks into two subsets, we specify two different system execution behaviors, i.e., *normal* and *local* execution behavior. When the system exhibits normal

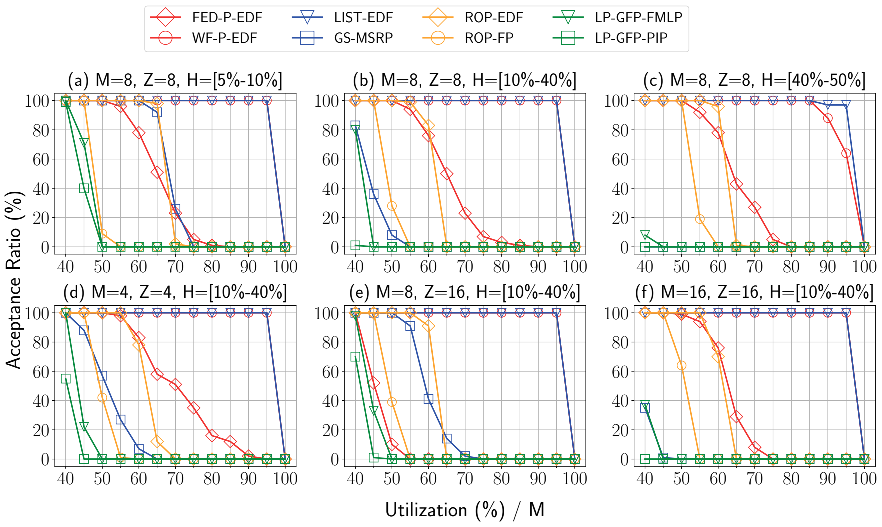


Fig. 8.3: Schedulability of different approaches for frame-based task sets(1).

execution behavior, all timing requirements of all tasks are satisfied at any point in time, whereas, if the system exhibits local execution behavior, timing guarantees can only be given for all critical tasks $\tau_k \in \mathcal{T}_{crit}$.

Each recurrent real-time task $\tau_k \in \mathcal{T}$ is assumed to have a sporadic arrival pattern and is characterized by a tuple $(C_{k,1}, C_{k,s}, C_{k,2}, S_k, p_k, q_k, D_k, T_k)$:

– Each τ_k releases an infinite number of task instances denoted as *jobs*. T_k indicates the minimum inter-arrival time of τ_k.
– D_k describes the relative deadline of τ_k. A constrained-deadline task system is considered, in which $D_k \le T_k$ for each task τ_k.
– $C_{k,1}$ and $C_{k,2}$ denote the WCETs of the first and second *computation segments*, respectively.
– $C_{k,s}$ is the WCET of the typically offloaded task share if executed on the *local* system.
– p_k and q_k are the WCETs of the pre- and post-processing routines, which are executed before and after the offloading operation of a job of task τ_k, respectively.
– S_k is the offloading or *suspension* time of τ_k.

We assume that $T_k \ge D_k > 0$ and $C_{k,1}, C_{k,s}, C_{k,2}, S_k, p_k, q_k \ge 0$. Moreover, we assume that WCET of pre- and post-processing routines are less than or equal to the WCET of local execution, i.e., $p_k + q_k \le C_{k,s}$. Furthermore, the WCET of a job of task τ_k under any possible execution scenario is greater than 0, i.e., $C_{k,1} + C_{k,s} + C_{k,2} > 0$ and $C_{k,1} + p_k + q_k + C_{k,2} > 0$. For notational brevity, we denote $C_k^\sharp = C_{k,1} + C_{k,s} + C_{k,2}$ and $C_k^\flat = C_{k,1} + p_k + q_k + C_{k,2}$.
In addition, we assume that the local cyber-physical real-time system, termed *local system*, is a uniprocessor system, in which tasks are scheduled according to a preemptive

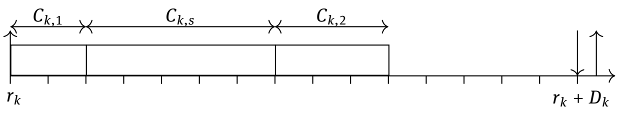


Fig. 8.4: A job of task τ_k is executed locally (local execution behavior).

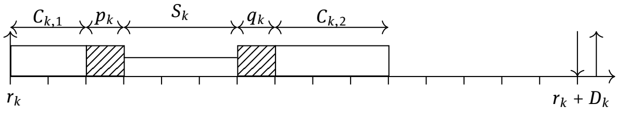


Fig. 8.5: An offloading operation of a job of task τ_k is performed successfully (normal execution behavior).

fixed-priority policy. More precisely, each task is assigned a unique priority, i.e., all jobs of task τ_k have the same priority. If at any point in time multiple jobs are ready, i.e., eligible for being executed on the local system, the job having the highest priority is executed. For each task τ_k, the unique set of the higher-priority tasks is denoted as $hp(\tau_k)$.

For a job of task τ_k arriving at time r_k the following execution scenarios are possible:

- The job is *executed locally* (Figure 8.4). In this case, the WCET of the job released at time r_k is $C_{k,1} + C_{k,s} + C_{k,2}$, i.e., $C_k^\sharp$.
- The job is *offloaded*. In this case, the job is first executed locally for up to $C_{k,1}$ execution time units and thereon enters the pre-processing routine for up to p_k execution time units. Suppose that the first computation segment as well as the pre-processing routine are finished at time ρ. Then, the considered job is offloaded to the remote system at time ρ. The actual offloading operation can be either *successful* or *unsuccessful*:
  - *Offloading is successful* if the computation result or *offloading response* is returned to the local system until time $\rho + S_k$. In this case, the offloading response is post-processed for up to q_k time units and the second computation segment is executed for up to $C_{k,2}$ time units (Figure 8.5). Accordingly, the execution time of the job of τ_k on the local system is at most $C_k^\flat$.
  - *Offloading is unsuccessful* otherwise. In this case, at time $\rho + S_k$, a local re-execution of the offloaded task share is performed for up to $C_{k,s}$ time units followed by the execution of the second computation segment for up to $C_{k,2}$ time units. In this case, the execution time of the job of τ_k on the local system is at most $C_k^\sharp + p_k$.

8.1.4.2 Recovery Protocols

Cyber-physical systems are applied throughout a broad range of areas, each exhibiting individual requirements and thus a need for situationally appropriate system behavior. For safety-critical cyber-physical systems, the timeliness of critical tasks must be guaranteed under any circumstances - even in the event of an unsuccessful offloading operation. Since in this case a larger amount of local resources is required, less resources remain to serve the non-critical tasks, as we explained in Section 8.1.4.1. However, depending on the actual system characteristics, timing constraints for non-critical tasks tend to be less strict. For instance, it is possible that a non-critical task misses its deadline, but that the results are still useful up to a certain degree [83, 87]. Nevertheless, it may be desirable to return to the normal execution behavior and to re-establish timing guarantees for both critical and non-critical tasks as soon as possible, especially since a non-critical task is not necessarily unimportant and thus should provide functionally and temporally correct results most of the time. Further discussion on the relation between criticality and importance can be found in [204].

Against this backdrop, we propose two recovery protocols allowing the system to satisfy its requirements under local execution behavior and to return to normal execution behavior:
- The *service protocol* aims to provide as much service as possible for non-critical tasks, even under local execution behavior.
- The *return protocol* aims to minimize the amount of time, in which the system exhibits local execution behavior after an unsuccessful offloading operation.

Independent of the actual protocol, we assume that the local system exhibits normal execution behavior at time 0, such that offloading is enabled for all tasks in $\mathcal{T}$. The schedule considers the execution of all tasks until the first moment $\gamma_{1,\searrow}$ in which the offloading operation of a certain task τ_k is unsuccessful. That is, a job of task τ_k, which has offloaded its computation at time $\gamma_{1,\searrow} - S_k$, does not receive the offloading response until time $\gamma_{1,\searrow}$ (Figure 8.6). Immediately after $\gamma_{1,\searrow}$, the local system exhibits local execution behavior. Until time $\gamma_{1,\searrow}$, three scenarios are possible for each incomplete job of all critical tasks τ_i in $\mathcal{T}_{crit}$:
- *The job of τ_i has not been offloaded*: In this case, no offloading operation will be performed for this job, but it is executed locally instead. Since it is possible that the pre-processing routine for offloading is already active at time $\gamma_{1,\searrow}$, the WCET of this job is upper-bounded by $C_{i,1} + p_i + C_{i,s} + C_{i,2}$, i.e., $C_i^\sharp + p_i$.
- *The job of τ_i is already offloaded, but no offloading response was received until time $\gamma_{1,\searrow}$*: In this case, the offloading process is aborted and the job is executed locally as of time $\gamma_{1,\searrow}$. Therefore, the WCET of this job is upper-bounded by $C_{i,1}+p_i+C_{i,s}+C_{i,2}$, i.e., $C_i^\sharp + p_i$.
- *The job of τ_i is already offloaded and the offloading response has been received prior to time $\gamma_{1,\searrow}$*: In this case, the job continues its final processing. Therefore, the WCET of this job is upper-bounded by $C_{i,1} + p_i + q_i + C_{i,2}$, i.e., $C_i^\flat$.

After $\gamma_{1,\searrow}$, timing guarantees are provided only for $\mathcal{T}_{crit}$. Moreover, offloading is inhibited for all critical tasks in the near future of $\gamma_{1,\searrow}$ due to the currently unreliable

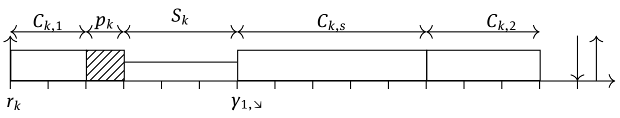


Fig. 8.6: An unsuccessful offloading operation of τ_k resulting in the transition to the local system behavior at time $\gamma_{1,\searrow}$.

connection leading to the missing offloading response. The offloading decision for non-critical tasks, however, depends on the applied recovery protocol:

Service Protocol Under the *service protocol*, offloading is inhibited for all instances of all tasks that are active as long as the system exhibits local execution behavior. The task share of each $\tau_i \in \mathcal{T}$ that is offloaded under normal execution behavior is executed locally within $C_{i,s}$ units of execution time. Since this leads to a higher workload on the local system, timeliness cannot be guaranteed for any non-critical task. Nevertheless, no non-critical task is aborted.

Return Protocol The *return protocol* does not inhibit offloading for *all* tasks, only for critical ones under local execution behavior. Non-critical tasks, by contrast, are offloaded regardless, but neither a re-execution nor a re-transmission is performed if an offloading response is not received in time. More precisely, the second subtask of τ_i is executed only if an offloading response is received, and aborted otherwise. Moreover, a job of τ_i in $\mathcal{T}_{non}$ is aborted whenever it misses its deadline.

As of time $\gamma_{1,\searrow}$, the local system exhibits local execution behavior until the point in time $\gamma_{1,\nearrow}$, in which timing guarantees can be given again for all tasks in $\mathcal{T}$. In the proposed protocols, two options are considered for the transit from local to normal execution behavior. They should be chosen based on the actual system requirements:

Abort-Transit This option aims to re-establish the normal system execution behavior as quickly as possible. Suppose that $\gamma_{1,\nearrow}$ is the earliest moment (after $\gamma_{1,\searrow}$) in which there is no incomplete job from $\mathcal{T}_{crit}$ at $\gamma_{1,\nearrow}$. All released but not yet finished instances of non-critical tasks are discarded.

Idle-Transit This option re-establishes the normal system execution behavior at the earliest moment $\gamma_{1,\nearrow}$ (after $\gamma_{1,\searrow}$) in which there is no incomplete job from $\mathcal{T}$ at $\gamma_{1,\nearrow}$.

We note that the above transitions are well-defined and the local system exhibits normal and local execution behavior in an interleaving manner.

8.1.4.3 Evaluation

In this subsection, we perform a case study on a robotic system to compare the acceptance ratio of schedulability over different protocols. More comprehensive numerical simulations can be found in the original paper [612].

Tab. 8.1: Periodic, implicit-deadline tasks; measurements of a Robotnik RB-1 Base robot platform. Note that the frequency of task τ_{laser} is 15.5 Hz.

Task	WCET [ms]	Period [ms]
τ_{laser}	6.732	64.516
τ_{odom}	1.046	60.0
τ_{tf}	0.333	60.0

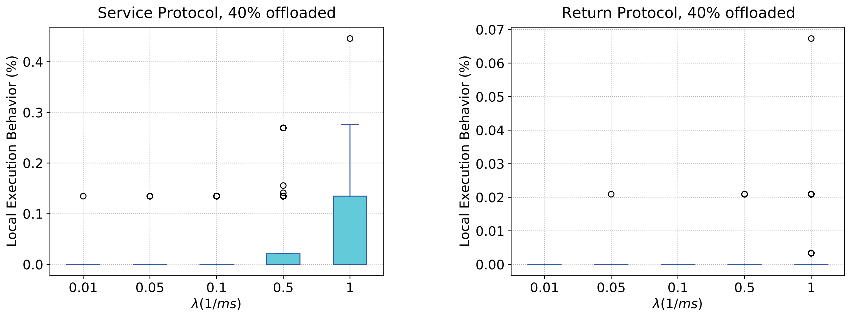


Fig. 8.7: The percentage of time the robot exhibits local execution behavior during the simulation for different probabilities of unsuccessful offloading operations and different percentages of offloaded workload under the service and the return protocol with 40 % offloaded workload per task.

Case Study on a Robotic System We adopt a Robotnik RB-1 Base robot platform [598], which uses the well-known Robot Operating System (ROS) [601]. We simulated the navigation of the robot in a virtual map and measured the timing data of the `move_base` node during a time frame of 60 seconds by using the Real-Time Scheduling Framework for ROS (ROSCH) [607] and RESCH [362]. We obtained three periodic, implicit-deadline tasks, as shown in Table 8.1, which are transformed into self-suspending tasks analogously to the tasks in experiment 1), and we considered the cases that 40 %, and 60 % of the task workload are offloaded. Moreover, we assume that $\mathcal{T}_{crit} = \{\tau_{odom}\}$ and $\mathcal{T}_{non} = \{\tau_{laser}, \tau_{tf}\}$. We simulate the system behavior using the event-based miss rate simulator from experiments 1) with $\lambda = 0.1 \cdot \frac{1}{ms}$. For each offloading case, the simulation was repeated 100 times.

Under the return protocol, Figure 8.8 shows that the amount of offloaded workload has no significant impact on the time that the system exhibits local execution behavior. Under the service protocol, we can observe that the time that the system exhibits local execution behavior is increased along with the increasing amount of offloaded workload. Overall, the derived results suggest that the amount of offloaded workload per task has a strong impact on the system execution behavior under the service protocol and thus should be taken into consideration at system design time.

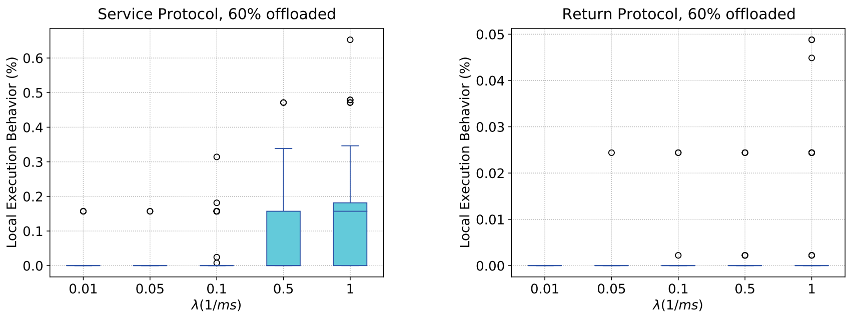


Fig. 8.8: The percentage of time the robot exhibits local execution behavior during the simulation for different probabilities of unsuccessful offloading operations and different percentages of offloaded workload under the service and the return protocol with 40 % and 60 % offloaded workload per task.

8.1.5 Probability-Based Timing Analysis

In this subsection, we present a multinomial-based approach to efficiently calculate the deadline miss probability. Additionally, three analytical approaches are presented, i.e., Chernoff bound, Hoeffding's inequality, and Bernstein's inequality.

8.1.5.1 System Model and Notation

We consider a given set of n independent periodic (or sporadic) tasks $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$ in a uniprocessor system. Each task τ_i releases an infinite number of task instances, called jobs, and is defined by a tuple $((C_{i,1}, \ldots, C_{i,h}), D_i, T_i)$, where D_i is the relative deadline of τ_i and T_i is its minimum interarrival time. In addition, each task has a set of h distinct execution modes $\mathcal{M}$ and each mode j with $j \in \{1, \ldots, h\}$ is associated with a different WCET $C_{i,j}$. We assume those execution modes to be ordered increasingly according to their WCETs, i.e., $C_{i,m} \leq C_{i,m+1}\ \forall m \in \{1, \ldots, h-1\}$. Furthermore, we assume that each job of τ_i is executed in one of those distinct execution modes. To fulfill its timing requirements in the j^{th} execution mode, a job of τ_i that is released at time t_a must be able to execute $C_{i,j}$ units of time before $t_a + D_i$. The next job of τ_i must be released at $t_a + T_i$ for a periodic task and for a sporadic task the next job is released at or after $t_a + T_i$. In this work, we focus on *implicit-deadline* task sets, i.e., $D_i = T_i$ for all tasks, and *constrained-deadline* task sets, i.e., $D_i \leq T_i$ for all tasks. We assume that a job execution is aborted as soon as the absolute deadline is reached, to ensure that there is no 'domino effect' to jeopardize the execution of the other jobs.

We assume a preemptive fixed-priority scheduling policy is used in the considered system. The tasks are indexed according to their priority, i.e., τ_1 has the highest and τ_n has the lowest priority. In addition, $hp(\tau_k)$ denotes the set of tasks with higher priority than τ_k and $hep(\tau_k)$ is $hp(\tau_k) \cup \{\tau_k\}$. $\mathbb{P}_i(j)$ denotes the probability that a job of task τ_i is executed in mode j with related WCET $C_{i,j}$ and we assume that each job is executed in exactly one of these distinct execution modes, i.e., $\sum_{j=1}^{h} \mathbb{P}_i(j) = 1$. In addition, we assume that these probabilities are independent from each other according to the following definition:

Definition 27 (Independent Random Variables). *Two random variables are (probabilistically) independent if the realization of one does not have any impact on the probability of the other.*

Particularly, for a newly arriving job the probability of the execution modes is independent from the execution mode of the jobs of previous jobs.

8.1.5.2 Definition of Deadline Miss Probability

To derive the probability of deadline misses, we look for the probability that the accumulated workload S_t over an interval of length t is at most t, where S_t can be calculated by the sum of random variables, i.e., the sum of probabilistic WCETs from all tasks $\tau_i \in hep(\tau_k)$ over. That is, the situation where S_t is larger than t for an interval of length t and hence $\mathbb{P}(S_t > t)$ is the overload probability at time t. To upper bound the probability that this test fails, the minimum probability among all time points at which the test could fail should be derived. Hence, the probability of a deadline miss Φ_k can be upper bounded by

$$\Phi_k = \min_{0<t\leq D_k} \mathbb{P}(S_t > t) \tag{8.1}$$

When analytical bounds are in use, we seek $\mathbb{P}(S_t \geq t)$ instead of $\mathbb{P}(S_t > t)$. By definition $\mathbb{P}(S_t \geq t) \geq \mathbb{P}(S_t > t)$, so these values can be used directly when looking for an upper bound of $\mathbb{P}(S_t > t)$.

8.1.5.3 A Multinomial-Based Approach

Conventionally, the probability of deadline misses can be derived from convolution-based approaches [476]. In such approaches, the underlying random variable represents the execution mode of each single job. This state space in fact can be transformed into an equivalent space that describes the states on a task-based level by proving the invariance when considering equivalence classes for each task. As a result, we introduce a novel approach that is based on the multinomial distribution. For the simplicity of presentation, we only highlight the insight of the aforementioned transformation.
The traditional convolution-based approach determines the *overload probability* by successively calculating the probability for all other points of interest in the analysis interval. However, the probability for t is evaluated based on the resulting states after all jobs in the analysis interval are convoluted. With respect to t, the intermediate states are not considered. By utilizing this insight, we can merge the states to efficiently calculate the vector representing the possible states at time t. If the number of jobs for a task is known, all possible combinations and the related probabilities can be calculated

directly using the multinomial distribution. The rationale is to construct a tree based on the tasks, which means that the number of children on each level depends on the number of jobs the related task releases.

8.1.5.4 Analytical Upper Bounds

In the following, we demonstrate how common concentration inequalities used in machine learning, statistics, and discrete-mathematics can be used to derive analytical bounds on $\mathbb{P}(S_t \geq t)$.

Chernoff Bound can be exploited to over-approximate the probability that a random variable exceeds a given value. This statement is summarized in the following lemma:

Lemma 28 (Lemma 1 from Chen and Chen [131]). *Suppose that S_t is the sum of the execution times of the $\rho_{k,t} + \sum_{\tau_i \in hp(\tau_k)} \rho_{i,t}$ jobs in $hep(\tau_k)$ at time t. In this case*

$$\mathbb{P}(S_t \geq t) \leq min_{s>0} \left(\frac{\prod_{\tau_i \in hep(\tau_k)} (mgf_i(s))^{\rho_{i,t}}}{\exp(s \cdot t)} \right) \tag{8.2}$$

It is in general pessimistic and there is no guarantee for the quality of the approximation. To find the optimal value of s to minimize the right-hand side in Equation 8.2, it has been proven as a log-convex optimization problem [129].

Hoeffding's Inequality derives the targeted probability that the sum of independent random variables exceeds a given value. For completeness, we present the original theorem here:

Theorem 29 (Theorem 2 from [319]). *Suppose that we are given M independent random variables, i.e., $X_1, X_2, \ldots, X_M$. Let $S = \sum_{i=1}^{M} X_i$, $\bar{X} = S/M$ and $\mu = \mathbb{E}[\bar{X}] = \mathbb{E}[S/M]$. If $a_i \leq X_i \leq b_i$, $i = 1, 2, \ldots, M$, then for $s > 0$,*

$$\mathbb{P}(\bar{X} - \mu \geq s) \leq \exp\left(-\frac{2M^2 s^2}{\sum_{i=1}^{M} (b_i - a_i)^2} \right) \tag{8.3}$$

Let $s' = sM$, i.e, $s = s'/M$. Hoeffding's Inequality can also be stated with respect to S:

$$\mathbb{P}(S - \mathbb{E}[S] \geq s') \leq \exp\left(-\frac{2s'^2}{\sum_{i=1}^{M} (b_i - a_i)^2} \right) \tag{8.4}$$

By adopting Theorem 29, we can derive the probability that the sum of the execution times of the jobs in $hep(\tau_k)$ from time 0 to time t is no less than t. The detailed proof can be found in [85].

Bernstein's Inequality generalizes the Chernoff bound and the related inequality by Hoeffding and Azuma. The original corollary is also stated here:

Theorem 30 (Corollary 7.31 from [232]). *Suppose that we are given L independent random variables, i.e., $X_1, X_2, \dots, X_L$, each with zero mean, such that $|X_i| \le K$ almost surely for*
$i = 1, 2, \dots, L$ and some constant $K > 0$. Let $S = \sum_{i=1}^{L} X_i$. Furthermore, assume that $\mathbb{E}[X_i^2] \le \theta_i^2$ for a constant $\theta_i > 0$. Then for $s > 0$,

$$\mathbb{P}(S \ge s) \le \exp\left(-\frac{s^2/2}{\sum_{i=1}^{L} \theta_i^2 + Ks/3}\right) \tag{8.5}$$

The proof can be found in [232]. Note, however, that the result in [232] is stated for the two-sided inequality, i.e., as upper bound on $\mathbb{P}(|S| \ge s)$. Here, the one-sided result, which is a direct consequence of the proof in [232] (page 198), is tighter. Similarly, it can also be used to derive the probability of deadline misses. The detailed proof can also be found in [85].

Final Remark Considering the required runtime and the accuracy of different approaches, when a given task set needs to be analyzed, we suggest first running *Chernoff's*, *Hoeffding's*, and *Bernstein's* bounds. If a sufficiently low deadline miss probability cannot be guaranteed from these analytical bounds, we propose running the multinomial-based approach with equivalence class union in parallel on multiple machines by partitioning the time points equally.

8.1.6 Summary

In this section, we showed a novel resource-sharing protocol for multiprocessors, named DGA, that can serve a high utilization of critical sections while guaranteeing the given hard real-time constraints. In addition, we presented adaptive protocols for computation offloading that are able to countermeasure the unreliable connection. Unlike conventional analyses for hard real-time systems, our innovated convolution-based approach is able to efficiently derive safe upper bounds for the probability of deadline misses under soft real-time constraints.

8.2 Communication Architecture for Heterogeneous Hardware

Henning Funke
Jens Teubner

Abstract: In this section, we look at distributed processing on a smaller scale. Even a single-machine system today internally looks—and behaves—like a distributed system: multiple *processing modules* of different flavors (*e.g.*, CPUs, GPUs, FPGAs) interact with *memory modules*, which are scattered over the system, through an *interconnect* that is comprised of, say, QPI, PCIe, or “real” network links. In such environments, *communication* quickly becomes the limiting factor—not only for observable performance, but also for other system aspects, such as energy consumption.

We specifically look at communication patterns in heterogeneous CPU/GPU environments, and we illustrate how novel processing models can minimize communication overhead in such systems, which in turn results in significant performance improvements for real-world settings.

In GPU-accelerated, data-intensive systems, the PCIe link is often perceived as the limiting factor. Oftentimes, successions of fine-granular GPU kernel invocations amplify the problem, since they tend to cause multiple round-trips over the bandwidth-limited link. As we see in this section, unnecessary round-trips can be avoided by fusing fine-granular kernels into larger work units that can hide costly PCIe transfer times (query compilation can be a device to implement kernel fusion).

Eliminating the PCIe bottleneck, however, only exposes the GPU’s *on-chip communication* as the new bottleneck to GPU-assisted data processing. Here, the data-parallel processing modes of graphics processors and the synchronization between parallel units are the cause for redundant round-trips over the precious on-chip communication interfaces. These bottlenecks can be avoided when processing models are deliberately designed to be *communication aware*. A specific example is a novel combination of pipelining/streaming processing models with the data-parallel nature of GPUs, which aligns particularly well with the semantics of database query execution. For real-world settings, this results in a reduction of memory access volumes by factors of up to 7.5× and shorter GPU kernel execution times by factors of up to 9.5×.

8.2.1 Introduction

Graphics Processing Units (GPUs) are frequently used as powerful accelerators for database query processing. As the arithmetic throughput of the coprocessor peaks in the teraflop range, it becomes a challenge to provision enough data. For this reason,

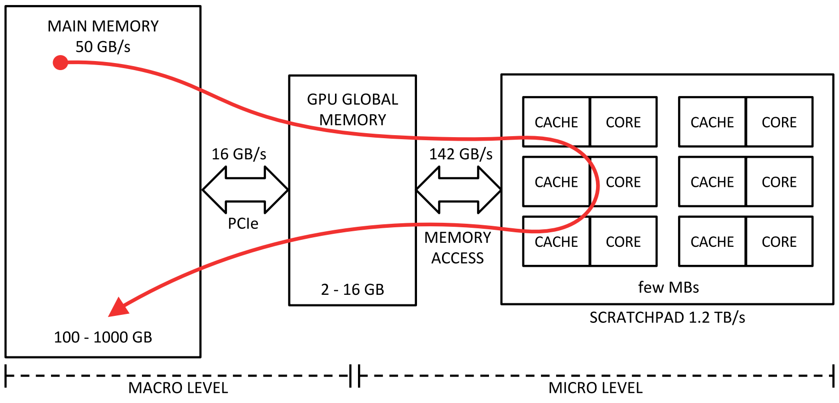


Fig. 8.9: The path of a tuple through the memory levels of a coprocessor environment.

hardware vendors equip graphics cards with high-bandwidth memory that has read and write rates of hundreds of GB/s. Still, memory intensive applications such as query processing fall behind regarding the cost of data movement for different reasons. Figure 8.9 shows the path of relational data through the hierarchical memory levels in a typical coprocessor system. Along the path, several bandwidth and capacity constraints need to be considered to achieve scalability and performance:

PCIe / OpenCAPI / NVLink A widely-acknowledged problem is the data transfer bottleneck between the host system and the coprocessor [270], typically via PCIe. Due to the coprocessor's limited memory capacity, data transfers are necessary *during* computations. With an order of magnitude between internal and external memory bandwidth, database developers are challenged with data locality-aware algorithms that efficiently use inter-processor communication. Recent technologies, i.e., OpenCAPI and NVLink, increase the bandwidth over PCIe, shifting the bottleneck toward GPU global memory.

GPU Global Memory The fine-grained data parallelism of a GPU typically requires that kernels perform additional passes over the data. Performing multiple passes, however, can significantly inflate memory loads and can cause a bandwidth bottleneck especially for random memory accesses.

Main Memory Integrated GPU-style coprocessors are a recent development to directly access the memory of the host CPU. Such an *Accelerated Processing Unit (APU)* allows the use of massively parallel processing without additional data transfers. However, the available memory bandwidth is lower than that of a dedicated GPU (30 GB/s vs. hundreds of GB/s).

Scratchpad Memory[1] Scratchpad memory is located on-chip and placed next to each compute unit of a GPU. It can be controlled as an explicit cache for low-level computations and offers a very high bandwidth. However, the capacity is limited to 16 kB–96 kB per core, which makes it challenging to use it for large-scale computations.

8.2.2 Contributions

With *HorseQC*, we developed a database query compiler that accounts for the hierarchical memory structure of modern coprocessor environments and for their inherent bandwidth limitations. In this section, we elaborate on the key building blocks of *HorseQC*, which can serve as a poster child in bandwidth-aware systems for other application contexts as well.

Specifically, we *(a) analyze the bandwidth limitations* in different database execution models; *(b)* demonstrate a *query compiler* for a coprocessor-accelerated database engine; *(c)* show how database sub-tasks can be realized in a *single pass over the data* (thus avoiding expensive memory round-trips); and *(d)* integrate these contributions in a *fully working system* that we use to evaluate our work.

Coprocessor-enabled database engines are typically classified by the *macro execution model* that they use to orchestrate the processing of query plans. Orthogonally, we devise a *micro execution model* that can be paired with different existing macro execution models, enhancing their communication- and resource-awareness.

8.2.3 Macro Execution Model

We begin by looking at macro execution models that have been employed in the past. To evaluate a relational query operator, state-of-the-art systems will select a number of primitives and execute the corresponding kernel sequence on the GPU. To feed the kernels with data, the macro execution model defines how data transfers will be interleaved with kernel executions. Here, the data movement from kernel to kernel may result in additional bandwidth demand compared with conventional systems. To understand the effect, we study the implications that existing macro execution models have on the use of bandwidth at multiple levels (PCIe, GPU global memory, etc.). We profiled the execution of Query 3.1 as a poster child from the star schema benchmark (SSB) [543]. The query was executed at scale factor 10 with CoGaDB [74] on a NVIDIA

1 We use the term *scratchpad memory* to disambiguate *shared memory* for CUDA and *local memory* for OpenCL.

GTX970 GPU.[2] In the following, we discuss three macro execution models: *run-to-finish*, *kernel-at-a-time*, and *batch processing*.

Algorithm 8: Run-to-finish execution of two successive kernels.

```
1 RUN-TO-FINISH – input: R, output: P
2 move R Host → GPU
3 tmp ← op1(R)                          /* invoke first GPU kernel */
4 P ← op2(tmp)                          /* invoke second GPU kernel */
5 move P GPU → Host
```

8.2.3.1 Run-To-Finish (Not Scalable)

A straightforward way to execute a sequence of kernels is to first transfer all input, execute the kernels, and finally transfer all output. The approach, illustrated in Algorithm 8, has the advantage that intermediate data remains in GPU global memory in-between kernel executions and no significant PCIe transfers are necessary. However, run-to-finish has the disadvantage that it works only if *all* input, output, and intermediate data is small enough to fit in GPU memory. Run-to-finish macro execution models are used, e.g., by Ocelot [302], CoGaDB [74], and others. The *lack of scalability* leads us to evaluate the following execution models.

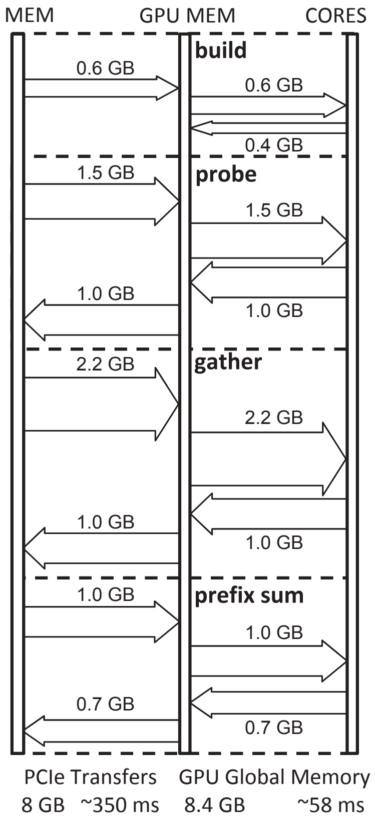
Algorithm 9: Kernel-at-a-time achieves scalability by transferring I/O for each kernel through PCIe.

```
1 KERNEL-AT-A-TIME – input: R, output: P
2 foreach $r_i$ in R=$r_1 \cup \cdots \cup r_m$ do
3     move $r_i$ Host → GPU
4     $m_i$ ← op1($r_i$)                 /* invoke first GPU kernel */
5     move $m_i$ GPU → Host (assemble into M)
6 foreach $m_j$ in M=$m_1 \cup \cdots \cup m_n$ do
7     move $m_j$ Host → GPU
8     $p_j$ ← op2($m_j$)                 /* invoke second GPU kernel */
9     move $p_j$ GPU → Host (assemble into P)
```

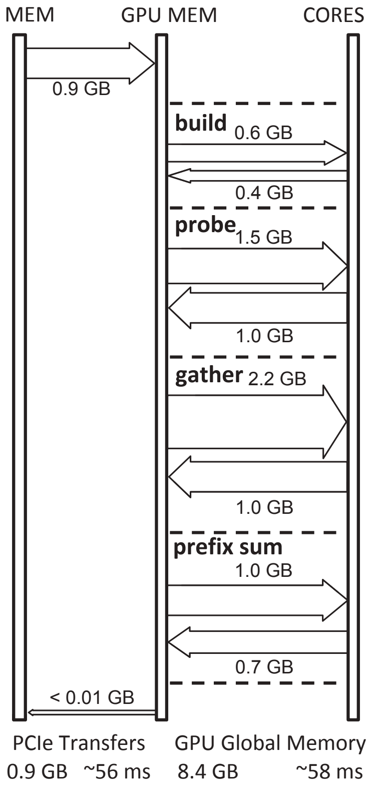
8.2.3.2 Kernel-At-A-Time

To process large data on coprocessors, we can execute each kernel on blocks of data. The pseudocode of this approach is shown in Algorithm 9. Processing blocks of data requires algorithm choices that can deal with partitioned inputs. Joins or aggregations, for instance, can be processed in this mode only if their internal state (e.g. a hash table) can fit in GPU global memory.

2 We measured 146.1 GB/s GPU global memory bandwidth in a host system with 16 GB/s bidirectional PCIe bandwidth.



(a) Kernel-At-A-Time



(b) Batch Processing

Fig. 8.10: Data movement for processing SSB Query 3.1. While the throughput of a is limited by PCIe transfers, b exposes GPU global memory access as the next bottleneck.

We analyze the data movement of kernel-at-a-time for SSB Query 3.1. Blocks are first moved via PCIe from the host to the coprocessor and then read by the kernel from GPU global memory (output passes both levels vice-versa). In this way, the data volumes for GPU global memory accesses equal the data volume transferred via PCIe, plus the cost to build up the hash tables in GPU global memory (0.4 GB here). Figure 8.10a shows the resulting data movement.

In the figure, the arrows annotated with data volumes represent PCIe transfers and GPU global memory accesses. We aggregated the data volumes by kernel types (e.g. scan, gather) and show only the most important kernels responsible for 88.2 % of the memory traffic. Given a PCIe bandwidth of 16 GB/s, all PCIe transfers together require at least 350 ms to complete. This exceeds the aggregate time for GPU global memory access by a factor of 5.8×. For kernel-at-a-time processing *the PCIe link is clearly the bottleneck*.

Kernel-at-a-time processing is used to scale out individual operators [358]. Unified Virtual Addressing (UVA) produces the same low-level access pattern, though it is transparent to the system developer.

8.2.3.3 Batch Processing

We can alleviate PCIe bandwidth limitations by rearranging the operations of kernel-at-a-time. Instead of running kernels until a column is processed, we can short-circuit the transfer of intermediate results to the host. Batch processing achieves this by reusing the output of the previous operation (op1) as input for the next operation (op2) instead of transferring to the host. This is applicable whenever intermediate batch results can be kept within GPU global memory. The corresponding pseudocode is shown in Algorithm 10.

Algorithm 10: Batch processing executes multiple kernels for each block that is transferred via PCIe.

```
1 BATCH PROCESSING – input: R, output: P
2 foreach $r_i$ in R=$r_1 \cup \cdots \cup r_m$ do
3     move $r_i$ Host → GPU
4     tmp$_i$ ← op1($r_i$)                 /* invoke first GPU kernel */
5     $p_i$ ← op2(tmp$_i$)                 /* invoke second GPU kernel */
6     move $p_i$ GPU → Host (assemble into P)
```

We analyze the data movement cost with the example of SSB Query 3.1. The GPU global memory load is the same as for kernel-at-a-time processing, because each kernel reads and writes I/O to GPU global memory. We obtain the PCIe transfer cost using the transfer volumes of the input columns of the query and the output of the final result. Figure 8.10b shows the resulting data movement cost. Batch processing reduces the amount of PCIe transfers by a factor of 8.8×. This shows that transferring data in blocks *and* performing multiple operators per block allows scalability and increases the efficiency compared to kernel-at-a-time.

Batch processing macro execution models have been used for coprocessing by GPUDB [728] and Hetero-DB [735]. Wu et al. [711] described the concept as *kernel fission* and identify opportunities to omit PCIe transfers automatically.

Limitations The lower amount of PCIe traffic can expose GPU global memory bandwidth as the next limitation. Batch processing reduces the PCIe transfer cost, but the amount of GPU global memory accesses remains unaffected. The memory access volume inside the device is now an order of magnitude larger. Despite the high bandwidth, this takes longer to process than the PCIe bus transfers (Figure 8.10b). For this reason, batch processing SSB Query 3.1 is *not* limited by PCIe transfers, but by accesses to the (high-speed) GPU global memory. Since in typical query plans, I/O and hashing operations both address the same GPU global memory, the situation may arise frequently in real-world workloads.

Tab. 8.2: Number of passes over the input data for benchmark queries. Out of 25 queries, 9 are definitely limited by GPU global memory.

Query	Passes	Query	Passes	Query	Passes
ssb11	7.5	ssb34	2.2	tpch5	7.2
ssb12	6.9	ssb41	7.4	tpch6	6.2
ssb13	6.7	ssb42	3.9	tpch7	**9.0**
ssb21	**9.6**	ssb43	3.5	tpch9	**9.0**
ssb22	**9.2**	tpch1	**15.5**	tpch10	5.8
ssb23	**9.1**	tpch2	**14.5**	tpch15	6.3
ssb31	**11.0**	tpch3	5.2	tpch18	**38.5**
ssb32	7.9	tpch4	6.6	tpch20	**10.5**
ssb33	7.5				

8.2.4 Micro Execution Model

Tuning the macro level helps to remove the main bottleneck for scalability: data transfers over PCIe. However, the macro level change exposes a new bottleneck: the memory bandwidth of GPU global memory. To utilize the GPU global memory bandwidth more efficiently, we need to apply additional micro-level optimizations using *micro execution models* and combine them with the macro execution model (batch processing) to achieve scalability *and* performance.

Existing micro-level optimizations such as *vector-at-a-time* processing [749] and *query compilation* [529] utilize memory bandwidth more efficiently by leveraging pipelining in on-chip processor caches. Therefore, both techniques are promising candidates for opening up the bottleneck of limited GPU global memory bandwidth. However, vector-at-a-time processing and query compilation are designed in the context of CPUs. While it is highly desirable to apply both techniques in the context of GPUs, mapping the techniques from CPU to GPU is challenging, as we discuss below.

Vector-At-A-Time To mediate the interpretation overhead of Volcano and the materialization overhead of operator-at-a-time, vector-at-a-time uses batches that fit in the processor caches. First, this reduces the number of `getNext()` calls from one per tuple to one per batch. Second, this makes materialization cheap because operators pick up the cached results of previous operators. On CPUs, vector-at-a-time benefits from batch sizes that are large enough to limit the function call overhead and small enough to fit in the CPU caches.

On GPUs, the compromise between tuple-at-a-time and full materialization strategies is not a sweet spot, however. Kernel invocations are an order of magnitude more expensive than CPU function calls. Furthermore, GPUs need much larger batch sizes to facilitate over-subscription and out-of-order execution. This leads to the problem that batches, which fit in the GPU caches, are too small to be processed efficiently. Alternatively, more recent GPUs support *pipes* to move a local execution context from one kernel to another. This has been used by GPL [557] for query processing. However, this technique still introduces an overhead for switching the execution context. In addition, it is limited to a depth of 2–32 kernels depending on the microarchitecture.

Query Compilation Query compilation is a common tool for avoiding excessive memory transfers during query processing. Compiling code for incoming queries becomes feasible with low-level code generation and achieves performance close to hand-written code. The compilation strategy of Neumann [529] keeps intermediate results in CPU registers and passes data between operators without accessing memory at all. The generated code processes full relations or blocks of tuples using a sequential tight loop.

To use query compilation on GPUs, we must integrate fine-grained data parallelism into compiled queries. The parallelization strategy of HyPer [425], however, uses a coarse-grained approach, so that it does not break with the concept of tight loops. In fact, HyPer does not use SIMD instructions [529] and thus omits fine-grained data parallelism. Even on CPUs with a moderate degree of parallelism in SIMD instructions, database researches are challenged by integrating query compilation and SIMD instructions [487, 639].

In summary, using a micro-level technique for efficient on-chip pipelining on GPUs remains a challenge. Applying any of the commonplace techniques makes it necessary to combine at least three things that are hardly compatible: fine-grained data-parallel processing, extensive out-of-order execution, and deep operator pipelines. To achieve our goal of mitigating the GPU global memory bottleneck, we need to develop a new micro execution model.

8.2.5 Data-Parallel Query Compilation

In the following, we show a micro-level execution strategy that reduces GPU global memory access volumes by means of pipelining in on-chip memory. To this end, we show the approach of our query compiler *HorseQC* and its integration with the operator-at-a-time execution engine of CoGaDB [74].

8.2.5.1 Fusion Operators

HorseQC extends the operator-at-a-time approach with the concept of *fusion operators*, operators that embrace multiple relational operations. A fusion operator replaces a

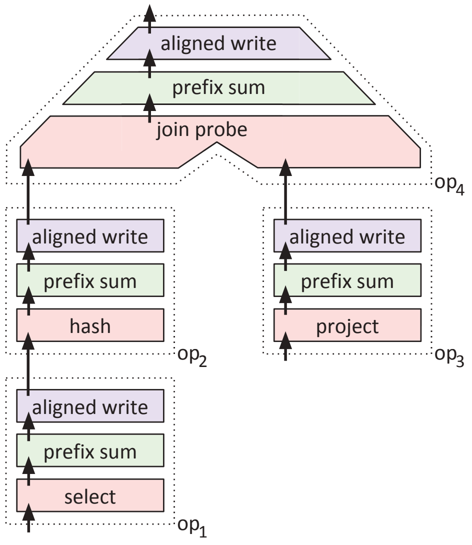


Fig. 8.11: Operator-at-a-time.

sequence of conventional operators in the physical execution plan with a micro-level-optimized pipeline. The data movement within a fusion operator can be improved by applying different micro level execution models.

8.2.5.2 Micro-Level Pipeline Layout

To keep matters simple, we first apply query compilation with the operator-at-a-time primitives described by He et al. [300]. This choice is not limiting as other data-parallel primitives may be used instead. However, a commonality of different primitive sets is that they use *relational primitives* with relational functionality (e.g. select) and *threading primitives* with thread coordination functionality (e.g., map, prefix sum, gather).

State Of The Art We look at a query with two input tables and a total of four relational operators $\mathtt{op}_1, \cdots, \mathtt{op}_4$. Operator-at-a-time runs three primitives per operator (cf. Figure 8.11 on the right): The first pass executes the relational primitive (e.g., select, project) and counts the number of outputs of each thread. The second pass computes a *prefix sum* to obtain unique per-thread write positions. The third pass performs an *aligned write*. This means that the output values are written into a dense array and may include executing the relational primitive for a second time to produce the output values. Thus, the query is processed in twelve operations with separate GPU global memory I/O.

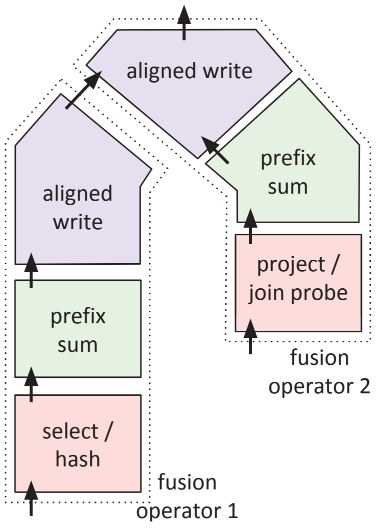


Fig. 8.12: Multi-pass QC.

Multi-Pass Query Compilation By grouping operations that are applied to the same input table, the query may be processed with two fusion operators. Within each fusion operator, we apply the following query compilation strategy (cf. Figure 8.12): We extract the prefix sum from the operators and execute it only once between all relational primitives and all aligned writes. The relational primitives are then compiled into one kernel called `count`, which is executed before the prefix sum. The aligned writes are compiled into one kernel called `write`, which is executed after the prefix sum. In this way, we apply *kernel fusion* [689] to the four relational primitives and to the four aligned writes. The same query is processed with six operations and the operations in compiled kernels communicate through on-chip memory instead of GPU global memory.

8.2.6 Memory Access and Limitations

In Figure 8.13, we illustrate the bandwidth characteristics of our example query when using code generation with three phases. The figure shows the behavior of the three-phase micro execution model described above with the batch processing macro execution model. To analyze the implications of forwarding intermediate results in the generated kernels through registers and scratchpad memory, we extended the illustration with an additional GPU-internal layer of memory.

GPU global memory access has previously been the bottleneck for query execution. Here the `count` kernel accesses 1.7 GB in GPU global memory, the prefix sum computation accesses 0.8 GB in GPU global memory, and the `write` kernel accesses 1.9 GB in GPU global memory. This is a reduction by a factor of 1.9× compared with batch

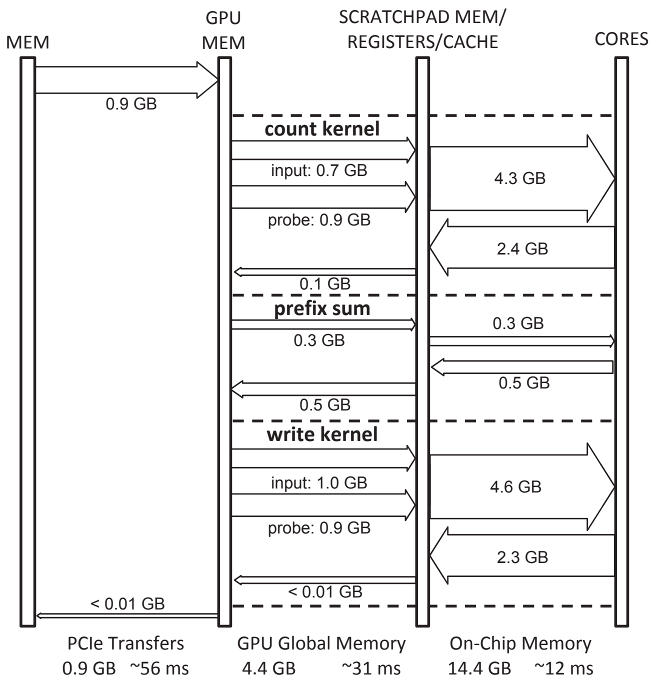


Fig. 8.13: Data movement for data-parallel query compilation with three phases.

processing. In the generated kernels, a substantial amount of memory traffic has moved to on-chip memory. In on-chip memory, the access volume of 14.4 GB is not a limiting factor due to the extremely high bandwidth of 1.2 TB/s of scratchpad memory.

Although the reduced GPU global memory traffic may suggest that the approach eliminates the bottleneck, real-world queries still experience limitations. In fact, Section 8.2.10.6 shows that compilation with three phases can still not saturate PCIe for 9 out of 12 SSB queries. This is because the query complexity prevents the strategy from utilizing the full GPU global memory bandwidth. Therefore, we investigate ways to further increase the processing efficiency in the next section.

8.2.7 Processing Pipelines in One Pass

The previous execution model relied on a typical programming concept of GPUs that executes operations with multiple kernels. The kernels that execute the actual work for the operations are interleaved with kernels that execute prefix sum computations. To further improve the processing efficiency, we have to break with this concept. With a new micro execution model, we avoid round trips to GPU global memory, which are caused by multi-pass implementations. This enables us to radically reduce GPU global memory traffic and lift the bandwidth bottleneck.

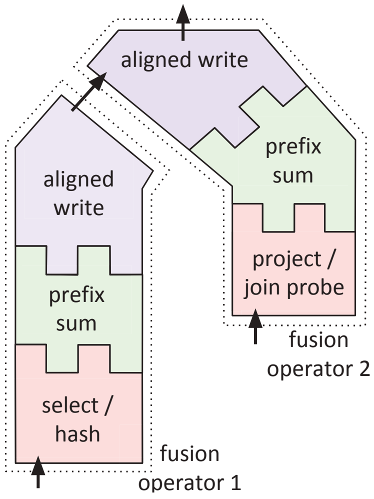


Fig. 8.14: Compound kernel.

Compound Kernel Kernel fusion brought reduction operations (e.g. prefix sum) as boundaries into the spotlight. Previously, we computed the prefix sum *between* two generated kernels to obtain write positions. Instead of two separate kernels, we now generate only one *compound kernel* that integrates the prefix sum computation (cf. Figure 8.14), which eliminates multiple passes. Computing write positions *within* a generated kernel makes it possible to process pipelines in one pass without intermediate materialization. In this way, each fusion operator is executed by a single compound kernel. In the following, we look at implementation strategies for reduction operations that enable fully pipelined processing.

Atomic Prefix Sum The separation into multiple reduction kernels with intermediate materialization impedes pipelining. To introduce a pipelined implementation, let us first look at a very simple sequential prefix sum:

```
for(i=0; i<n; i++)
  if(flags[i]) prefix_sum[i] = sum++;
```

The sequential prefix sum loops through the array `flags` while writing *and* incrementing `sum` for every valid entry. Figure 8.15a illustrates the use of the prefix sum for a dense write of selected input elements. When parallelizing the `for`-loop, this implementation runs into the issue of many threads trying to increment `sum` at the same time. To resolve this parallel dependency, atomic operations can be used to isolate parallel modifications of the same memory address. Atomic operations ensure a consistent state, yet are executed in an undefined order. The following code executes an *atomic prefix sum* to compute unordered, dense write positions:

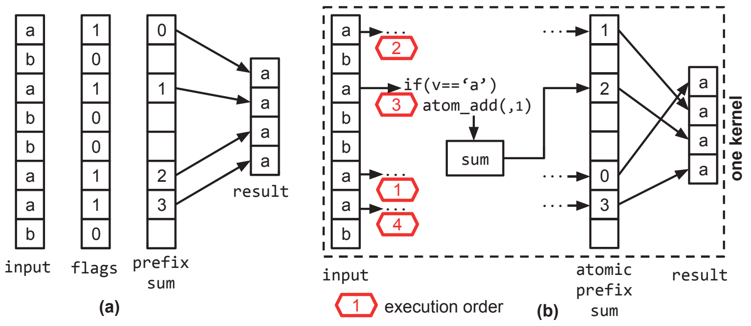


Fig. 8.15: The computation of a prefix sum for writing selected elements to a dense array (a) can be parallelized using atomic operations (b).

```
if(is_selected) wp = atom_add(&sum, 1);
```

Threads contribute an offset of `1` to the sum at address `&sum` by executing the expression conditionally. Each `atomic_add(..)` returns the previous state of `sum`. Thus, threads immediately obtain a unique global write offset as `wp` in register. This is illustrated in Figure 8.15b.

The use of atomic operations causes a break with the semantic of the prefix sum because the result has *no defined order*. For the relational semantic, however, only the *uniqueness* of output positions is critical. Output permutations lead to non-aligned GPU global memory access where adjacent threads do not write to adjacent memory addresses. The impact on write throughput, however, is limited, because the filter semantics lead to non-aligned access for separate prefix sums as well.

8.2.7.1 Memory Access and Limitations

The compound kernel micro execution model further reduces GPU global memory access by a factor of 2.4× to 1.8 GB (see Figures 8.13 and 8.16). Compared with operator-at-a-time, this is a reduction by a factor of 4.7×. Pipelining the prefix sum avoids round trips to GPU global memory that are necessary in the three-phase micro execution model. The compound kernel has only a minimal GPU global memory access volume for input, output, and hash-table access. Now the on-chip traffic is balanced with the GPU global memory traffic when relating each memory volume to the available bandwidth.

The described approach heavily relies on atomic operations. This has the disadvantage of causing limitations for parallelism. Although the execution order is undefined, the operations *are* sequentialized and reducing n values takes $O(n)$ parallel steps. However, Egielski et al. [195] showed that recent hardware support makes atomic operations competitive with parallel algorithms. Still, the integrated prefix sum puts significant pressure on the atomic functional units, which prevents pipeline kernels from utilizing

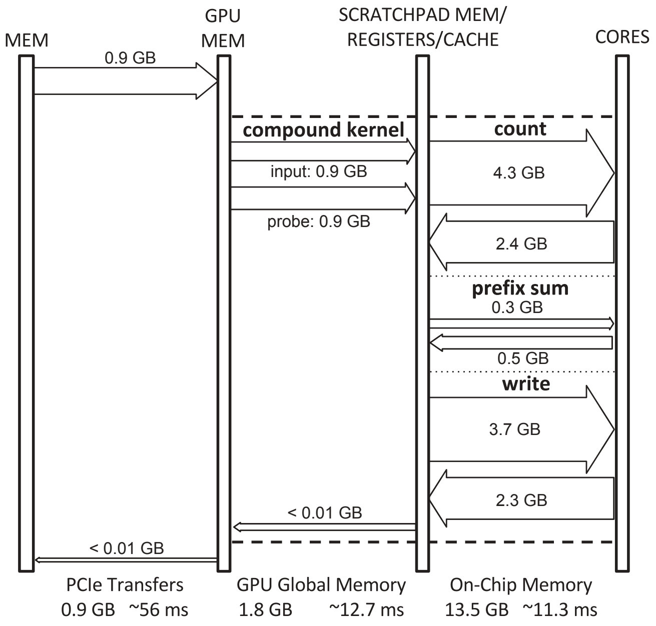


Fig. 8.16: Data movement for query compilation with one pass. The compound kernel reduces data movement by 4.7×.

full GPU global memory bandwidth. In the following, we address this issue and show how the efficiency of parallel reductions in compound kernels can be increased.

8.2.8 Efficient Pipelined Reductions

We have showed a way to pipeline reductions in generated kernels using atomic operations. This benefits the memory efficiency, but also reveals the atomic functional units of a GPU to be a bottleneck. This is especially critical because several operations that are combined in the compound kernel rely on atomic isolation as well. Specifically, the state-of-the-art implementations of hash joins and hash aggregations [358] use atomic operations to isolate hash table inserts.

This section addresses performance bottlenecks that occur when utilizing atomic reductions to pipeline relational operators. We show a new technique called *local resolution, global propagation*, that is used by *HorseQC* to pipeline prefix sums, single tuple aggregation, and grouped aggregation efficiently. The approach reduces the pressure on atomic functional units and offers tunability regarding hardware and thread-group granularity. We describe the approach in the following.

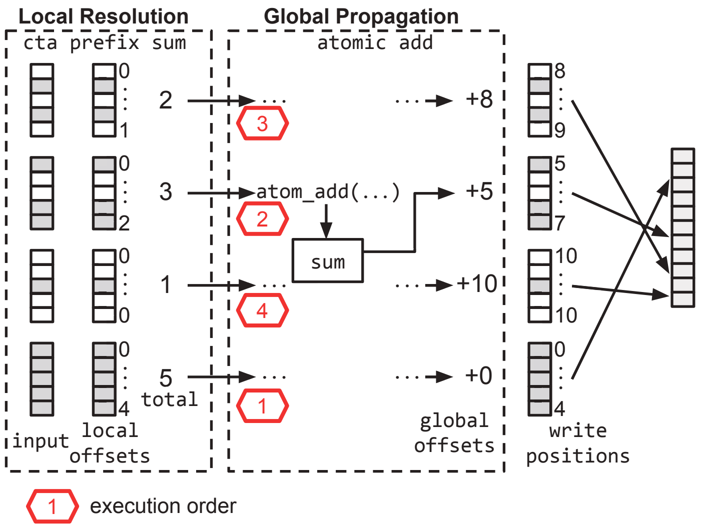


Fig. 8.17: Computing write positions with local resolution (local offset), global propagation (global offset).

8.2.8.1 Local Resolution, Global Propagation

Like other efficient GPU implementations such as in CUB [489], local resolution with global propagation consists of two levels of reductions. In contrast to other techniques, however, it always uses pipelined techniques on *both* levels. Local resolution is an additional pre-reduction step, computed by a local thread group, whereas global propagation is the same atomic reduction as described in Section 8.2.7. We use the term *Collaborative Thread Array* (CTA) for the thread groups in local resolution. CTAs can either match the workgroup (AMD) or thread-block (NVIDIA) size of the GPU kernel or work on a finer granularity.

The following code, illustrated in Figure 8.17, executes an atomic prefix sum using local resolution, global propagation:

```
l_os = cta_prfx,(flags, &cta_total);  //local res.
if,(cta_thread_idx == 0)
  g_os = atom_add,(&sum, cta_total);  //global prop.
wp = l_os + g_os;
```

First, each CTA executes `cta_prfx` to compute a local prefix sum on `flags`. This is the local resolution step. We implement `cta_prfx` with SIMD reductions (cf. Intra-Warp Scan Algorithm by Sengupta et al. [622]). The function returns the local offset `l_os` and the sum of all flags assigned to the CTA `cta_total`. Second, one thread of each CTA adds `cta_total` atomically to a global counter `sum`. This is the global propagation step.

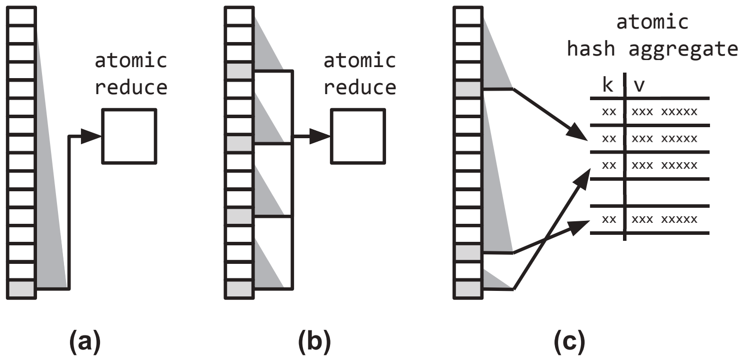


Fig. 8.18: Local resolution mechanisms: (a) Work-efficient reduction (b) SIMD reduction (c) segmented reduction.

The call to `atom_add` returns the global offsets `g_os`. Finally, the write position `wp` is the sum of `l_os` and `g_os`.

Compared with the simple atomic prefix sum, we now add pre-aggregates instead of 1/0 flags to `sum`. Accordingly, each atomic add obtains ranges of output indices instead of a single index. The process is analogous to *allocating* segments of output memory to CTAs. The order of the allocations is undefined, however. (See the execution order in Figure 8.17.) This leads to an output that is ordered *within segments* and permuted *between segments*. Further investigation reveals that, due to the GPUs stream processing engine, the permutations exhibit locality, leading to semi-ordered output data.

Local Resolution Mechanisms The mechanisms used for local resolution are interchangeable. This makes it possible to tune pipelined reductions and to apply them in different operations. Figure 8.18a and 8.18b show the integration of work-efficient reductions [56] and SIMD reductions [622]. Both techniques have different thread group granularities and we can choose between them to adapt to the hardware parallelism of different processors. Figure 8.18c shows the use of pipelined segmented reductions for grouping. First, segmented reductions compute pre-aggregates in scratchpad memory. Second, global propagation inserts the pre-aggregates into a hash table with an atomic operation. The ability to control scratchpad memory opens up a new design space for grouping algorithms in pipelined computations (e.g. handling frequent items). A similar approach PLAT [722] aggregates frequent grouping keys in a table local to each CPU core.

8.2.9 DBMS Integration

We integrated our query compiler *HorseQC* into the open source DBMS CoGaDB, leveraging the built-in code generator *Hawk* [75]. The DBMS uses a columnar data layout and processes full columns operator-at-a-time on GPUs and CPUs. We use the front-end and the storage layer of CoGaDB; *HorseQC* adds a compiler-based execution engine.

We added two components to the DBMS: 1. a query compiler that compiles fusion operators to GPU code (cf. Section 8.2.4); and 2. a *translation layer* that identifies fusion operators and drives the query compiler. Currently, there are two different workflows for the translation layer:

1. CoGaDB parses the SQL code for a query and generates a query plan. The translation layer applies the produce/consume model [529] to the query plan to determine fusion operators. We use this approach for the SSB queries and TPC-H Q6.
2. The translation layer parses a JSON file that describes the query plan including the fusion operators. This enables us to process queries when (1) cannot handle the queries via SQL (e.g. correlated subqueries or automatic unnesting). This is used for the other TPC-H queries.

When the fusion operators are defined, the translation layer drives the query compiler to compile and execute. Finally, the decompression of dictionary compressed columns and sorting are executed by CoGaDB's original execution engine.

8.2.10 Evaluation

Section 8.2.3.1 showed that query coprocessing in existing macro execution models is sensitive to memory bandwidth bottlenecks on various hierarchical levels. We proposed several micro execution models that allow to remove memory indirections to achieve a more efficient use of bandwidth. In this section, we evaluate our approaches and carefully assess bandwidth and throughput in identifying several benefits.

The experimental study is structured as follows. First, we evaluate the *micro execution models* and we execute specific queries to analyze the *reduction performance* of the proposed techniques in Experiments 1 and 2. Then, we evaluate the micro execution models for the SSB and TPC-H benchmarks in Experiments 3 and 4. Next, we analyze the *integration* of our micro execution model with the batch processing *macro execution model*. In doing so, we analyze the *real-world benefits* of our approach with a comparison of end-to-end performance in Experiment 5 and a scalability analysis in Experiment 6. Note that all experiments, except for Experiment 6, were executed with scale factor 10.

8.2.10.1 Processing Techniques

This section describes three micro execution models built into *HorseQC*. The goal is to use them within macro execution models to improve performance. Therefore, it is crucial to achieve a higher throughput than PCIe when executing queries. We show the benefit of our approaches by comparing them with an operator-at-a-time micro

Tab. 8.3: Coprocessors used in the evaluation.

Model	Type	Archi-tecture	Cores	Scratch pad (KB)	B/W (GB/s)
GTX970 (NV)	GPU	Maxwell	13	96	146.1
GTX770 (NV)	GPU	Kepler	8	48	167.6
RX480 (AMD)	GPU	Ellesmere	32	32	104.9
A10 (AMD)	APU	Godavari	8	32	18.7

execution model. In this way, we analyze the benefit of moving data transfers between relational operators to the on-chip level.

Multi-pass The first approach separates reductions from the generated kernels, which leads to an execution in multiple passes (Section 8.2.5). Each reduction is executed on materialized data using the `boost::compute` library.

Pipelined The second approach integrates reductions into a fully pipelined kernel using atomic operations (Section 8.2.7). By using atomic operations for each reduction input, the approach is an instance of local resolution, global propagation that has no local resolution step.

Resolution The third approach increases the efficiency of pipelined reductions with local resolution methods such as pre-aggregation (Section 8.2.8). We differentiate between local resolution implementations using Resolution:SIMD for SIMD reductions and Resolution:WE for work-efficient reductions.

Operator-at-a-time We use CoGaDB 0.4.1, which processes full columns of data in each operator with CUDA kernels. It features a run-to-finish macro execution model and an operator-at-a-time micro execution model.

8.2.10.2 Baselines

PCIE transfer The *PCIe transfer time* is the time it takes to transfer input and output data between the host's main-memory and GPU global memory. It is the target time used by micro execution models for balancing throughput and PCIe bandwidth. The PCIe transfer time is shown in each graph with a dashed line (- - -).

Memory bound The *GPU global memory bound execution time* is the time it takes to access the data. As each approach has to read the input columns and write the output columns, the baseline is a lower bound on the kernel execution time. We indicate it with a solid line (——) in each graph.

Listing 8.1: Query 1 is a simple selection and projection query inspired by the star schema benchmark.

```
SELECT lo_extprice * lo_discount + lo_tax AS revenue
FROM   lineorder
WHERE  lo_quantity BETWEEN 25 - x AND 25 + x
```

8.2.10.3 System Configuration

For the experiments, we use three dedicated GPUs with PCIe gen 3.0 links and one APU that accesses main-memory directly. Table 8.3 specifies the GPU models and shows hardware properties. The amount of scratchpad is available *per core*. The reported bandwidth refers to GPU global memory for the GPUs and to main-memory for the APU. It was measured using on-GPU `memcpy` of 1 GB data. We measured bidirectional PCIe transfers between CPU and GPU as 12.1 GB/s.
Both NVIDIA GPUs GTX770 and GTX970 run in a system with an Intel Xeon E5-1607 CPU. We use the NVIDIA 364.19 driver and CUDA Toolkit 7.5 with OpenCL drivers. The AMD RX480 GPU is placed in a separate system with the A10-7890K APU. We use the AMDGPU-Pro 16.40 driver for the GPU and the fglrx 15.201 driver for the APU. Each system is running Ubuntu 14.04 and uses the boost library 1.61.

We used the profiling tools `nvprof 2.0.28` for NVIDIA hardware and `CodeXLGpu-Profiler V4.0.511` for AMD hardware to measure kernel execution times, PCIe transfers, and GPU global memory access. For the measurements of kernel execution times, we used both tools to profile individual kernels and sum up the kernel execution times when multiple kernels were involved.

8.2.10.4 Experiment 1: Pipelined Prefix Sum

We compare several pipelined prefix sum techniques with one non-pipelined technique for a query that filters and projects one table. This allows us to analyze the benefit of integrating prefix sum computations into single-pass kernels. We execute Query 1, shown in Listing 8.1, and vary the selectivity in the range [0, 1] using `x`. By running the experiment on four GPUs, we aim to assess the best local resolution mechanisms for a given hardware. Figure 8.19 shows the results.

Observations Pipelined techniques perform better than Multi-pass in most cases. Integrating the prefix sum computation into single-pass kernels reduces the kernel execution times by a factor of up to 6.3×. While processing with Multi-pass takes up to 328.6 % of the PCIe time, Resolution:SIMD uses only 101.3 % of the PCIe time in the worst case (selectivity 1.0, RX480). This shows that the approach can saturate the bus bandwidth for a variety of configurations. On the A10 there are no PCIe transfers and Resolution:SIMD increases the overall throughput by factors of up to 1.6× over Multi-pass.

The results show that the local resolution step reduces the performance impact of atomic operations. This becomes visible for higher selectivity factors. Pipelined has higher executions times because the strategy executes one atomic addition per output. However, Resolution:SIMD and Resolution:WE show good performance across all selectivities due to local resolution.

Resolution:SIMD achieves the shortest kernel execution times in most cases and allows memory bound processing on the GTX970. On the GTX770, lowering the output

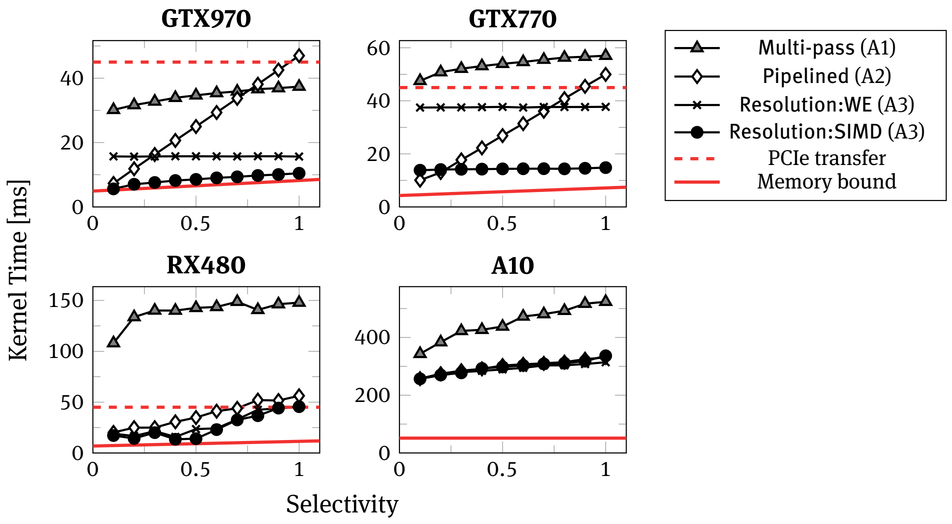


Fig. 8.19: Projection query executed with different approaches. Integrating prefix sums into kernels allows fastest execution.

size down to 0 does not affect the execution time. We conclude that the GTX770 is compute-bound earlier than the GTX970. The higher memory bandwidth of the GTX770 leads to an increased throughput for atomic operations and Pipelined can outperform Resolution:SIMD for selectivities below 10 %. On the RX480 and on the A10 there is no definite advantage for one of the reduction techniques. In the following, we use only Resolution:SIMD and skip the other techniques for a clear presentation.

8.2.10.5 Experiment 2: Pipelined `GROUP BY`

We evaluate the effect of pipelined `GROUP BY` aggregations using Operator-at-a-time, Pipelined, and Resolution. The query groups all tuples of `lineorder` according to the computed attribute `lo_orderkey%x` into sums. We vary the number of groups by increasing `x` from 2 to 16 384. We show the results of the experiment on a GTX970 GPU in Figure 8.20.

Observations The execution times of Operator-at-a-time do not depend on the group size. The main cost factor is sorting the input columns. Pipelined shows up to 11.1× lower execution times but only for larger group sizes. For group sizes below 64, we observe high execution times. This is caused by the heavy contention of parallel aggregation hash-table inserts.

The bottleneck is resolved by Resolution which uses pre-aggregations to reduce the contention. The results show that execution times reduce by a factor of up to 126×. However, the local pre-aggregations have a limited effect on larger group numbers. This

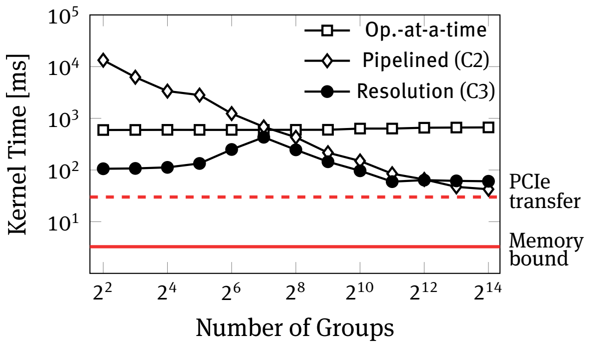


Fig. 8.20: Performance of grouped aggregations.

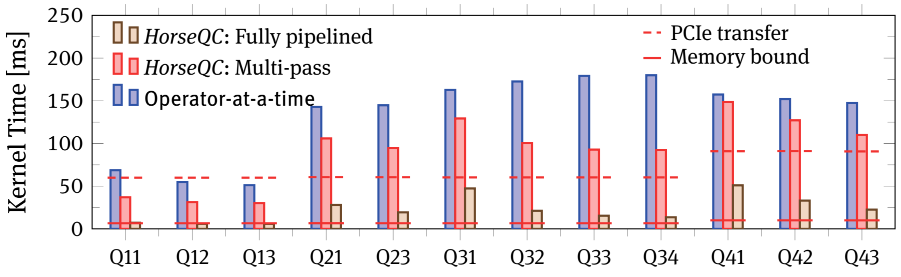


Fig. 8.21: Performance of SSB queries.

explains the spike at 128 groups, where both pre-aggregation and contention have an effect. While the approaches cannot saturate PCIe when aggregating a full table, filters reduce the cost of grouping for real-world queries.

8.2.10.6 Experiment 3: Star Schema Benchmark

The previous experiments showed that pipelining specific reduction operations helps to increase the throughput of query processing. In this experiment, we analyze whether this behavior carries over to real-world situations. To this end, we execute the SSB Queries[3] on the GTX970 GPU.

We use Operator-at-a-time and two variants of our query compiler. *HorseQC*: Multi-pass uses pipeline breaking implementations for reductions (A1, B1 and C1). *HorseQC*: Fully pipelined integrates all pipeline operations in one kernel (using A3, B3 and C2). We show the results of the experiment in Figure 8.21.

3 We could not process SSB Query 2.2 as we do not yet support range predicates on dictionary compressed columns.

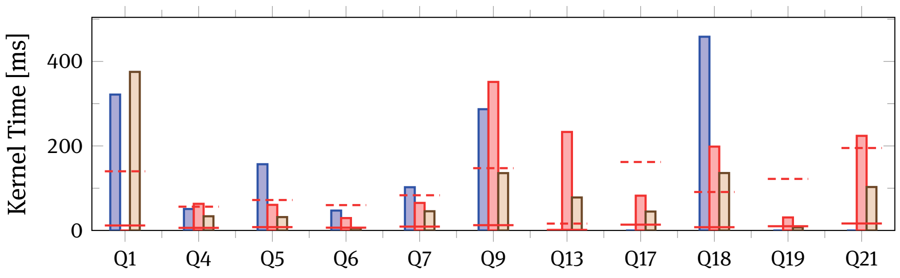


Fig. 8.22: Performance of TPC-H queries.

Observations The bandwidth analysis in Section 8.2.3.1 showed that 4 out of 12 queries are limited by GPU global memory access in operator-at-a-time processing.

- The kernel execution times of Operator-at-a-time show that compute and latencies make the problem worse. While PCIe would allow execution times between 60.6 ms to 90.9 ms, the kernel execution times take longer for 10 out 12 queries with up to 295.5 %.
- *HorseQC*: Multi-pass improves over Operator-at-a-time and uses only 50.5 % of the PCIe bandwidth transfer time in the best case and 215.5 % in the worst case. This shows that without efficient pipelining of reduction operations, the benefit of query compilation is limited.
- *HorseQC*: Fully pipelined lowers all kernel execution times to a level that is consistently lower than PCIe transfer times. This shows that compiling pipelines into one kernel with local resolution, global propagation provides an execution approach with sufficient throughput. Processing takes 9.7 % of the PCIe transfer time in the best case and 78.1 % in the worst case. For Queries 1.1, 1.2, and 1.3, kernel execution is memory bound by GPU global memory access.

8.2.10.7 Experiment 4: TPC-H Queries

We execute and profile queries from the TPC-H benchmark to show the effect when relaxing the specific assumptions of the star schema benchmark (e.g. using one centralized table). We select a subset of queries based on the work by Boncz et al. [61] to capture challenging aspects of the TPC-H benchmark: Q1, Q4, Q13, and Q21 contain heavy aggregation; Q9 and Q18 contain heavy joins; and Q4, Q19, and Q21 contain parallelism bottlenecks. We modified 4 queries, because *HorseQC* currently does not support all operations, e.g., `like` expressions. The results of the experiment are shown in Figure 8.22. For Q1, there is no result for *HorseQC*: Multi-pass, because the strategy ran out of GPU memory. The results shown for Operator-at-a-time are for all TPC-H queries supported by the DBMS.

Observations The PCIe and memory-bound baselines show larger variations than for the SSB benchmark. This is mainly caused by the join structure, e.g., Q13 joins three small tables, while Q17, Q18, and Q21 join multiple instances of the largest `lineitem` table.
The kernel execution times show that *HorseQC* can improve over operator-at-a-time by factors of up to 8.6×. For Q1, Q4, and Q9, there are cases where Operator-at-a-time has shorter kernel execution times than compiled strategies. Further investigation showed that in these cases Operator-at-a-time moves some operators to the CPU, which means that the measurements cover a limited amount of operations.

Comparing the variants of the query compiler, we observe that *HorseQC*: Fully pipelined consistently improves over *HorseQC*: Multi-pass by a factor of up to 5.4×. *HorseQC*: Fully pipelined achieves lower execution times than PCIe transfer times for 8 out of 11 queries. For Q1, Q13, and Q18, the PCIe bandwidth cannot be fully saturated. This is because the queries contain grouped aggregations of unfiltered columns (cf. Experiment 2). The execution times of *HorseQC*: Fully pipelined take 5.6 % of the PCIe transfer time in the best case and 268.1 % in the worst case.

8.2.10.8 Experiment 5: Scalability

Due to the deeply integrated storage layer implementations of the host DBMS CoGaDB, we were unable to build a fully scalable version of *HorseQC*. For this reason, we perform a separate experiment that integrates the Resolution micro execution model with the batch processing macro execution model for the star join from SSB Query 3.1. Decoupling this experiment allows us to apply the rules for coprocessor data management by Yuan et al. [728] and to measure end-to-end performance for larger datasets.

The star join recombines three dimension tables and one fact table with an overall selectivity of 3.4 %. We build hash tables for the dimension tables in GPU global memory. The fact table resides in pinned host memory and each column is partitioned into blocks of 0.5 MB, 2 MB, or 8 MB. The blocks are transferred asynchronously via PCIe into an inner kernel that computes the star join by probing each dimension hash table.

Figure 8.23 shows the end-to-end execution times for each block size when executing the experiment. We observe that execution times grow linearly with increasing scale factors and that block sizes larger than 2 MB can saturate the PCIe bandwidth. The computation does not become a bottleneck for the examined scale factors. With a block size of 4 MB and scale factor 300, the size of intermediate data in GPU global memory is only 473 MB. Therefore, we expect the approach to scale to even larger databases with linear performance.

8.2.10.9 Experiment 6: End-to-End Performance

To make a comparison with other database systems, we execute the TPC-H queries with different database systems and measure end-to-end performance. We compare MonetDB5 Dec2016-SP3 executed on CPUs, and CoGaDB 0.41 and *HorseQC* executed

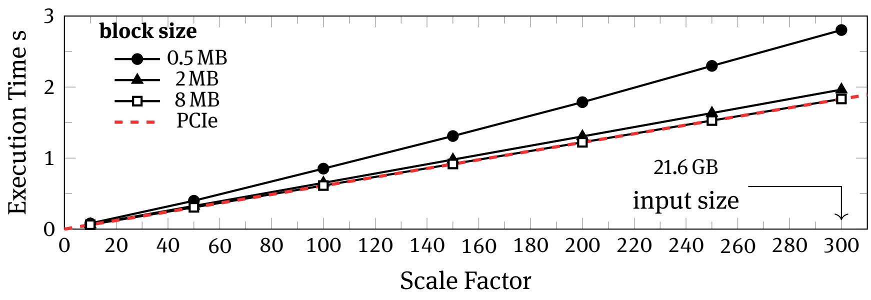


Fig. 8.23: End-to-end performance of star join computation for different scale factors.

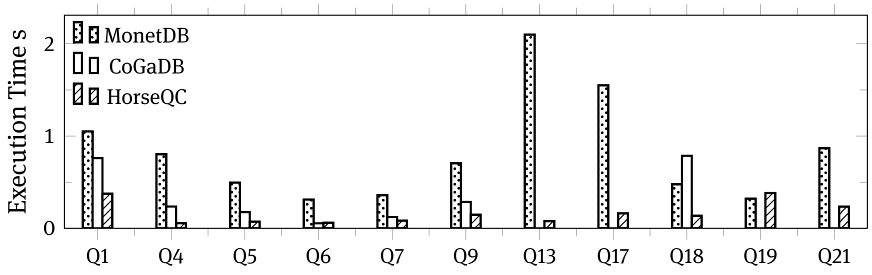


Fig. 8.24: End-to-end performance of TPC-H queries.

on GPUs. Both competitors feature an operator-at-a-time approach. We perform the measurements with warm caches. MonetDB runs on a workstation-class system with an Intel Xeon E5-1607 CPU and 32 GB RAM. CoGaDB and *HorseQC* run on the GTX970. The results are shown in Figure 8.24.

Observations For the supported queries, *HorseQC* is up to 5.8× faster than CoGaDB. While CoGaDB uses GPU global memory as a cache for frequently used columns, *HorseQC* does not cache data between queries. This shows that *HorseQC* uses memory and interconnects more efficiently. For Q6 there is no improvement, because query execution is PCIe bound.

HorseQC has lower execution times than MonetDB by a factor of up to 26.9×. Despite moving data through the PCIe bottleneck, the additional bandwidth resources of GPU global memory offer an acceleration. For Q19, MonetDB has a lower execution time than *HorseQC*. This shows that for queries with a low complexity, it is more effective to process data directly than moving it over PCIe.

8.2.11 Discussion

In the previous experiments, we evaluated our new approaches for querying compilation on coprocessors. Across all experiments, we were able to show improvements of query compilation over operator-at-a-time processing. Operator-at-a-time has a low memory efficiency due to large materialization volumes and repetitive operations. Therefore, the approach cannot efficiently utilize the memory systems surrounding the coprocessor.

While naive compilation techniques increase the memory efficiency, reductions and prefix sums split operator pipelines into multiple passes. In this way, the approach inherits the drawbacks of operator-at-a-time. This becomes visible because kernel execution times frequently exceed PCIe transfer times.

We demonstrated a query compilation technique that merges the operators of a pipeline into one compound kernel. When combined with efficient reduction techniques, the compound kernel achieves substantial advantages over other processing approaches. With upcoming OpenCAPI and NVLink interconnects, these improvements to GPU-local processing are essential in order to take advantage of the increased bandwidth of the new hardware. In the evaluation setting, the PCIe bandwidth can be saturated for all SSB queries. For the TPC-H benchmark, the approach is an improvement over operator-at-a-time and naive compilation, but saturates PCIe in only 8 out of 11 queries. We conclude that the compound kernel works particularly well with star join queries.

8.2.12 Summary

In this section, we showed query processing techniques that help to balance the data movement cost and compute throughput on GPU-style coprocessors. We measure the data transfer volumes in different scalable processing approaches to assess bandwidth bottlenecks. While naive scalable execution techniques are limited by PCIe bandwidth, batch processing is limited by GPU-local throughput. To address the bottleneck, we propose micro execution models that benefit from on-chip pipelining. Naive query compilation techniques allow simple code generation but inherit the memory-intensity of operator-at-a-time. We introduce compound kernels that merge several pipeline phases into one efficient kernel.

9 Energy Awareness

Energy is a fundamental resource constraint that is present almost everywhere in life. Many of the previous chapters indirectly discuss the energy demand of cyber-physical systems and machine learning. Taking a broader view of cyber-physical systems, we see that the total amount of energy required to solve a specific problem is (for constant P)

$$W = P \cdot t$$

where P is the power to run the hardware and t the amount of time this hardware needs to execute a given software. Hence the energy consumption is typically determined by

1. the hardware used (that is P)
2. the algorithm that is run on the hardware (this can influence P and t)
3. the time the hardware executes the implemented algorithm (that is t)

In practice there is often a trade-off between these quantities. Hardware that has great processing power can execute software very quickly, but often also requires much more energy. Similarly, less powerful hardware may take longer to execute a software pipeline while the overall energy consumption is smaller since it requires less power. Last, certain implementations might use specific hardware features (e.g. a GPU) that influences both the energy and time required for execution. With the ongoing integration of Machine Learning (ML) into cyber-physical systems, two research directions must be explored.

First, in order to apply and train Machine Learning models on small devices, the energy consumption of the ML algorithm itself must be reduced. Needed is a holistic approach that takes all steps of the ML pipeline into account, starting from its theoretical model down to its specific implementation on a specific hardware platform.

Second, the application of ML models to reduce the energy consumption of *other* parts of the cyber-physical system must be explored. Here, a cross-domain approach that combines domain-specific knowledge with ML for the right problems is necessary.

This chapter performs an exemplary discussion of both approaches. Section 9.1 discusses how probabilistic undirected models can be rephrased with integer-only operations such that floating-point co-processors are no longer required. It introduces Bit-Length Propagation (BL-Prop) and combines it with a novel IntGD algorithm for numerical optimization with integrality constraints. The resulting algorithm enables the training and inference of Markov random fields on small devices using integer-only arithmetics.

Section 9.2 discusses how ML models can be integrated into the wireless communication of cyber-physical systems. More specifically, methods for modeling power consumption for different communication technologies are discussed including LTE, LTE-A, and NB-IoT. The integration of ML models in the User Equipment (UE) for estimating transmission uplink power under external influences (e.g., signal strength or signal quality) is further explored and discussed in a real-world context.

https://doi.org/10.1515/9783110785944-009

9.1 Integer Exponential Families

Nico Piatkowski

Abstract: In this contribution, we study how knowledge about the underlying compute architecture can be incorporated directly into the learning problem. More precisely, we consider the arithmetic limitations of Ultra-Low Power (ULP) Micro-Controller Units (MCU). Such systems do not contain arithmetic co-processors, which implies that most arithmetic computations must be emulated via integer logic. However, this creates a large performance penalty for any machine learning method that relies heavily on floating-point arithmetic. To mitigate this issue, we show how the model itself can be rephrased with integer-only operations such that floating point co-processors are no longer required. We exemplify this procedure with probabilistic undirected models, so-called Markov random fields. All steps of learning and inference are discussed. An approximate but integer-only probabilistic inference procedure called bit-length propagation (BL-Prop) is presented. BL-Prop is based on belief propagation, where instead of the full messages, only their bit-length is propagated along the models' conditional independence structure. We analyze the algorithm and show what factors have the largest influence on the approximation quality.

Furthermore, we derive IntGD—a numerical optimization method for convex objective functions with integrality constraints. The method is based on an accelerated proximal algorithm for non-smooth and non-convex penalty terms. For integer gradients computed via BL-Prop, IntGD is guaranteed to deliver an integer learning procedure in which the final parameter vector as well as all intermediate results are integers. Numerical experiments on benchmark data show that integer models allow us to achieve a competitive prediction quality on low-end hardware while maintaining a large speedup compared with its double precision counterpart—thus, completely mitigating the performance penalty that arose from the missing floating point unit.

9.1.1 Introduction

Big data analytics for streaming sensor data challenges the resource efficiency of algorithms in several ways. Running data mining methods in resource-constrained computational environments generates challenges in terms of execution time and energy consumption. Fortunately, optimizations that reduce the number of cycles in which the CPU is busy also save energy consumption. When reviewing the specifications of processing units, one finds that integer arithmetic is usually cheaper in terms of instruction latency, i.e., it needs a smaller number of clock cycles until the result of an arithmetic instruction is ready. Table 9.1 shows the latencies of arithmetic instructions measured in terms of clock cycles for Intel CPUs and ARM CPUs and for Nvidia GPUs. Note that transcendental functions are composed of multiple instructions and therefore may take substantially more cycles than the ones reported in Table 9.1. This motivates reducing the number of cycles in which code is executed when designing new, resource-aware learning algorithms.

Nowadays, big data arises in social media, industry, and basically all scientific research areas. Data sets grow in size because they are increasingly being gathered by ubiquitous information-sensing mobile devices. The joint prediction of various unknowns based on multiple observed inputs is a ubiquitous subtask in real-world problems from various domains, including computational biology, computer vision, and natural language processing. Probabilistic graphical models are well-suited for such tasks, but they suffer from the high complexity of probabilistic inference. Many approximate approaches to probabilistic inference based on Belief Propagation (BP) [404, 560] were proposed in the last decade, e.g., Counting BP [369], Lifted BP [10], Stochastic BP [539], Tree-reweighted BP [691], Tree Block Coordinate Descent [642], and Particle BP [331]. Quadrature-based methods [572, 573] deliver promising results, but are not well-suited for embedded or resource-constrained environments. In contrast to these approaches, the underlying model class here is restricted to the integers, which results in a reduced runtime and energy savings, while maintaining good performance. Asymptotically, the new approach has the same complexity as the vanilla BP, but it uses cheaper operations.

This new approach should not be confused with models that are designed for integer state spaces, in which case the state space $\mathcal{X}$ is a subset of the natural numbers or, more generally, is a metric space. Here, the state space may be a random discrete space without any additional constraints.

Estimation in discrete parameter models was recently investigated by Chaorat and Seri [140]. They discuss consistency, asymptotic distribution theory, information inequalities, and their relations with efficiency and super-efficiency for a general class of m-estimators. Unfortunately, we do not consider the case when the true estimator is not included in the search space and therefore, their analysis cannot be used to estimate the error when the optimizer has to be approximated.

Bayesian network classifiers with reduced precision parameters were presented by Tschiatschek et al. [671]. They evaluate empirically the classification performance when reducing the precision of Bayesian networks probability parameters. After learning the parameters as usual in $\mathbb{R}$ (represented as 64-bit double-precision floating-point numbers), they varied the bit-width of mantissa and exponent, and reported the prediction accuracy in terms of the normalized number of correctly classified test instances. They found that after learning, the parameters may be multiplied by a sufficiently large integer constant (10^9) to convert the probabilities into integer numbers. However, Tschiatschek et al. missed an important point, namely that real value probability parameters are necessary only for Bayesian networks.

Tab. 9.1: Instruction latencies (in clock cycles) of Floating-Point (FP) and integer (INT) scalar arithmetic operations for three processing architectures [24, 334, 542]. x/y means that latency is x for 32-bit and y for 64-bit operands. A single value indicates that both latencies are the same or, in case of ARM and GPU, that 64-bit integer arithmetic is not supported. For GPU, the values are based on the operation throughput. Cycles of Intel Sandy Bridge integer division and ARM11 integer multiplication depend on the lengths of their operands.

	Sandy Bridge		ARM		GPU	
	FP	INT	FP	INT_{32}	FP	INT_{32}
Addition	3	1	8	1	3/7	3
Multiplication	5	3	8/9	4-5	3/7	14
Division	14/22	13-25	19/33	-	7/-	-
Bit shift	-	3	-	2	-	7
Square root	14/22	-	19/33	-	14/-	-

For undirected graphical models, this is not the case. As a result, the general framework of undirected graphical models [692] may be mapped to the integer domain. A new optimization scheme is proposed, that allows the resource-constrained learning of integer parameters without the need for floating-point computation. This opens up the opportunity of running data mining tasks on resource-constrained devices. To be more precise, based only on integers, it is possible to compute approximations to the

– the marginal probabilities,
– the Maximum-A-Posteriori (MAP) assignment of the model,
– the Maximum Likelihood Estimate (MLE) either via an approximate closed form solution or an integer variant of stochastic gradient descent.

In this contribution, algorithms for integer models are derived. It turns out that the integer approximations do deliver a reasonable quality and are around twice as fast as their floating-point counterparts. This contribution is based on [574] and [567], and is organized as follows. A short introduction to probabilistic graphical models is given in Section 9.1.2. In Section 9.1.3, the intuition behind integer undirected graphical models is explained, and the corresponding algorithms are derived. Furthermore, a bound on the training error is presented. Two instances of the integer framework, *Integer Markov random fields* and *Integer Conditional Random Fields*, are evaluated in Section 9.1.4 for synthetic and real world data.

9.1.2 Probabilistic Graphical Models

In the following, the basic notation and concepts of probabilistic graphical models are introduced. Let $G = (V, E)$ be a graph with $|V| = n$ and $\mathcal{N}_v := \{w \in V : (v, w) \in E\}$ the neighbors of vertex $v \in V$. Each vertex $v \in V$ corresponds to a random variable (RV) X_v with realization x_v and domain $\mathcal{X}_v$. Consider the n-dimensional RV $\boldsymbol{X} = (X_v)_{v \in V}$ with realization $\boldsymbol{x} \in \mathcal{X} = \otimes_{v \in V} \mathcal{X}_v$. The probability of the event $\{\boldsymbol{X} = \boldsymbol{x}\}$ is denoted by $p(\boldsymbol{X} = \boldsymbol{x})$. $p(\boldsymbol{x})$ is used as a shortcut for $p(\boldsymbol{X} = \boldsymbol{x})$ in the remainder of this report. For a set of vertices $A \subseteq V$, $\boldsymbol{X}_A$ addresses the components of $\boldsymbol{X}$ that correspond to the vertices in A. For ease of notation, $\boldsymbol{X}_v$ and $\boldsymbol{X}_{\{v\}}$ are regarded as the same. For undirected graphical models, the joint probability mass function of $\boldsymbol{X}$ is given by

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}(G)} \psi_C(\boldsymbol{x}_C) \tag{9.1}$$

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x} \in \mathcal{X}} \prod_{C \in \mathcal{C}(G)} \psi_C(\boldsymbol{x}_C) \tag{9.2}$$

where $\mathcal{C}(G)$ is the set of all cliques[1] in G and $Z(\boldsymbol{\theta})$ is the normalization constant (since it does not depend on $\boldsymbol{x}$). Let C be a clique of G and $\mathcal{X}_C$ the corresponding joint domain of all vertices in C. The parameter vector $\boldsymbol{\theta} \in \Theta = \Omega^d$ contains $|\mathcal{X}_C|$ weights for each clique $C \in \mathcal{C}(G)$, i.e., $\boldsymbol{\theta} = (\boldsymbol{\theta}_C)_{C \in \mathcal{C}(G)}$, which results in $d = \sum_{C \in \mathcal{C}(G)} |\mathcal{X}_C|$. The *compatibility functions* ψ_C (also known as *factors*) are typically chosen to be

$$\psi_C(\boldsymbol{x}_C) = \exp(\langle \boldsymbol{\theta}_C, \phi_C(\boldsymbol{x}) \rangle)$$

since this ensures the positivity of $p_{\boldsymbol{\theta}}$ and leads to a canonical form of the corresponding exponential family member.

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \exp(\langle \boldsymbol{\theta}, \phi(\boldsymbol{x}) \rangle - A(\boldsymbol{\theta}))$$

The function ϕ is a *sufficient statistic* for $\boldsymbol{x}$ and may be understood as transformation of $\boldsymbol{x}$ into a binary valued feature space $\phi : \mathcal{X} \to \{0, 1\}^d$. For convenience, the components of $\boldsymbol{\theta}$ and ϕ are indexed by C to denote the subvector of weights or features that corresponds to a clique C. To address a certain component of $\boldsymbol{\theta}$ or ϕ, the corresponding event $\{\boldsymbol{X}_C = \boldsymbol{x}_C\}$ is used as an index, i.e., $\boldsymbol{\theta}_{\boldsymbol{X}_C = \boldsymbol{x}_C}$ or even $\boldsymbol{\theta}_{C = \boldsymbol{x}_C}$. If the parameters $\boldsymbol{\theta}$ are known, the maximum a posteriori prediction of the most likely joint state of all vertices can be computed by

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in \mathcal{X}} p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{\theta}, \phi(\boldsymbol{x}) \rangle . \tag{9.3}$$

Parameter Estimation A common choice for learning the parameters $\boldsymbol{\theta}$ of an undirected model is the maximum likelihood estimation, where the likelihood

$$\mathcal{L}(\boldsymbol{\theta} \mid \mathcal{D}) = \prod_{\boldsymbol{x} \in \mathcal{D}} p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{9.4}$$

1 A clique corresponds to a fully connected subgraph.

of the parameters $\boldsymbol{\theta}$ for given i.i.d. data[2] $\mathcal{D}$ is maximized. The MLE $\boldsymbol{\theta}^{\star}$, i.e., the solution that maximizes $\mathcal{L}$, has a closed form, if and only if the underlying graphical structure is a tree or a triangulated graph. In this case, $\boldsymbol{\theta}^{\star}$ is induced by the empirical expectation of the sufficient statistics

$$\begin{aligned}\boldsymbol{\theta}^{\star}_{v=x} &= \log \mathbb{E}_{\mathcal{D}}\left[\phi_{v=x}(\boldsymbol{x})\right],\\ \boldsymbol{\theta}^{\star}_{vu=xy} &= \log \frac{\mathbb{E}_{\mathcal{D}}\left[\phi_{vu=xy}(\boldsymbol{x})\right]}{\mathbb{E}_{\mathcal{D}}\left[\phi_{v=x}(\boldsymbol{x})\right]\mathbb{E}_{\mathcal{D}}\left[\phi_{u=y}(\boldsymbol{x})\right]}.\end{aligned} \tag{9.5}$$

The MLE $\boldsymbol{\theta}^{\star}$ for partially observed data and certain classes of graphical models like Conditional Random Fields (CRF) [655] can be found with gradient-based methods. Taking the logarithm of Equation 9.4, dividing by $|\mathcal{D}|$, and substituting Equation 9.1 for $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ yields the average log-likelihood (see Equation 9.6). Since the logarithm is monotonic, maximizing ℓ will reveal the same optimizer as $\mathcal{L}$. Since $\mathbb{E}_{\mathcal{D}}\left[\phi(\boldsymbol{x})\right] = \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \phi(\boldsymbol{x})$, ℓ is given by

$$\ell(\boldsymbol{\theta} \mid \mathcal{D}) = \left\langle \boldsymbol{\theta}, \mathbb{E}_{\mathcal{D}}\left[\phi(\boldsymbol{x})\right]\right\rangle - \ln Z(\boldsymbol{\theta}). \tag{9.6}$$

Taking the natural logarithm to form the log-likelihood is a random choice that may be replaced with any other $\log_b$ if desired. Since the second term is the cumulant generating function of $p_{\boldsymbol{\theta}}$, its partial derivative is the expected sufficient statistic for a given $\boldsymbol{\theta}$. This is plugged into the partial derivative of ℓ with regard to $\boldsymbol{\theta}_{\boldsymbol{x}_C=\boldsymbol{x}_C}$ (Equation 9.6) to obtain

$$\frac{\partial \ell(\boldsymbol{\theta} \mid \mathcal{D})}{\partial \boldsymbol{\theta}_{\boldsymbol{x}_C=\boldsymbol{x}_C}} = \mathbb{E}_{\mathcal{D}}[\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{\theta}}[\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})]. \tag{9.7}$$

Here, $\mathbb{E}_{\mathcal{D}}[\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})]$ denotes the empirical expectation of $\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})$, i.e., its average value in $\mathcal{D}$. By using Equation 9.7, the model parameters $\boldsymbol{\theta}$ can be estimated by any first-order optimization technique.

Inference: In the following, it is explained shortly how $\mathbb{E}_{\boldsymbol{\theta}}[\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})]$ is computed with *Belief Propagation* (BP). From now on, assume that the underlying graphical structure is a tree. The maximum clique size is thus 2. The message update rule is

$$m_{v\to u}(x_u) = \sum_{x_v\in\mathcal{X}_v} \psi_{v,u}(x_v, x_u)\psi_v(x_v) \prod_{w\in\mathcal{N}_v\setminus\{u\}} m_{w\to v}(x_v). \tag{9.8}$$

The messages $m_{v\to u}(x_u)$ are computed for all pairs of vertex $v \in V$ and neighbor $u \in \mathcal{N}_v$ until convergence. Converged messages are denoted by $m^{\star}_{v\to u}(x_u)$. The product of all incoming messages of a vertex is given by $M_v(x) := \prod_{u\in\mathcal{N}_v} m_{u\to v}(x)$. After convergence, the vertex marginal probabilities $p_v(x_v)$ that are implied by $\boldsymbol{\theta}$ can be computed with

$$p_v(x_v) = \frac{\psi_v(x_v)M^{\star}_v(x_v)}{\sum_{x\in\mathcal{X}_v} \psi_v(x_v)M^{\star}_v(x)}, \tag{9.9}$$

2 It is assumed that every training instance in $\mathcal{D}$ is fully observed.

whereas $M_v^\star(x)$ is the product of converged messages $m_{v\to u}^\star(x_u)$. In case of non-tree-structured graphs, BP performs multiple passes over all vertices until the convergence of messages is reached. The convergence depends on the dynamic range of the potentials. For trees and triangulated graphs, efficient orderings of message computations (schedulings) are known that have polynomial runtime $\mathcal{O}(m\deg(G)|\mathcal{X}|^2)$ and result in the exact marginal probabilities. We refer to [404] for discussions on belief-propagation and related algorithms.

9.1.3 The Integer Approximation

In their fundamental book on graphical models, Wainwright and Jordan [692] write: "It is important to understand that for a general undirected graph the compatibility functions ψ_C need not have any obvious or direct relation to marginal or conditional distributions defined over the graph cliques. This property should be contrasted with the directed factorization, where the factors correspond to conditional probabilities over the child-parent sets." This explains why it might be possible to have an undirected graphical model that is parametrized by integers. But the identification of integer parameters is not enough for excluding every floating-point computation. Moreover, the computations that are required for training and prediction have to be based on integer arithmetic. Last, the integer approximation should still deliver a reasonable quality in terms of training error and test error.

The first step is directly related to the above statement. The potential function

$$\overline{\psi}_C(\boldsymbol{x}_C) := 2^{\langle \boldsymbol{\theta}_C, \phi_C(\boldsymbol{x})\rangle} = \exp\left(\ln(2)\langle \boldsymbol{\theta}_C, \phi_C(\boldsymbol{x})\rangle\right) \tag{9.10}$$

is defined in a way, that yields only integer values as long as the parameters are positive integers. It is easy to see that replacing $\psi_C(\boldsymbol{x}_C)$ with $\overline{\psi}_C(\boldsymbol{x}_C)$ does not alter the marginal probabilities as long as the parameters are scaled by $1/\ln 2$. It is possible to convert parameters that are estimated with $\psi_C(\boldsymbol{x}_C)$ to $\overline{\psi}_C(\boldsymbol{x}_C)$ and vice versa without altering the resulting probabilities. Notice that $\overline{\psi}_C(\boldsymbol{x}_C)$ can be computed by a logical bit shift to the left, which consumes less clock cycles than the corresponding transcendental function. As already mentioned above, it requires that $\boldsymbol{\theta} \in \mathbb{N}^d$ for $\overline{\psi}_C(\boldsymbol{x}_C)$ is an integer and the product of compatibility functions and the normalization constant (see Equation 9.2) are computable by means of non-negative integer arithmetic. This restricts $p(\boldsymbol{x})$ and its marginals to $[0, 1] \cap \mathbb{Q}$. Although the computation of a probability would require floating-point division, its actual value is not required for estimating the integer model parameters.

Inference Recalling the message update Equation 9.8, one sees that all messages are integer valued, if $\psi_C(\boldsymbol{x}_C)$ is replaced by $\overline{\psi}_C(\boldsymbol{x}_C)$ and the initial messages are set to 1. Thus, the whole message computation and propagation procedure is already stated without floating-point computation. Nevertheless, recall that a CPU's integer width is constrained by its wordsize ω. $m_{v \to u}(x)$ may exceed the machines' integer precision 2^{ω} quite easily. Thus, many overflows could occur during message computations, which destroy the semantics of the messages and the resulting beliefs are no longer usable.

Initial attempts to make the computation more robust against overflows relied on the fact that messages $m_{v \to u}(x)$ may be scaled arbitrarily without changing the resulting marginal probabilities as long as the same scale is used for all x. Nevertheless, the

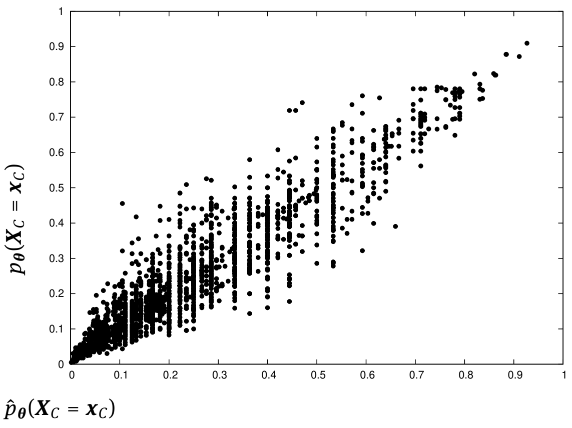


Fig. 9.1: Estimates of edge marginal probabilities for 50 random trees with 50 nodes and 2 states per node. Marginals are computed by the bit-length approximation ($\hat{p}$) and vanilla BP (p) on the same parameter vector $\boldsymbol{\theta}$.

messages cannot be simply divided by their sum as with floating-point arithmetic, since integer division will pin all messages down to 0. Numerous attempts to scale the integer messages by bit-shift operations have only worked on relatively small graphical structures, but all those approaches suffered from the loss of information that occurred whenever too many bits had to be shifted out in order to prevent overflows.

As a solution to this problem, new messages are defined. Instead of computing the original sum-product messages, we propose computing an approximation to the integer message bit length. The approximate bit length $\beta_{vu}(y)$ and the corresponding message $\hat{m}_{vu}(y)$ are given by

$$\beta_{vu}(y) := \max_x \boldsymbol{\theta}_{vu=xy} + \boldsymbol{\theta}_{v=x} + \boldsymbol{\theta}_{u=y} \tag{9.11}$$

$$+ \textstyle\sum_{w \in \mathcal{N}_v \setminus \{u\}} \beta_{wv}(x)\,, \tag{9.12}$$

$$\hat{m}_{vu}(y) := 2^{\beta_{vu}(y)} \tag{9.13}$$

$$= \max_x \psi_{vu}(x,y) \textstyle\prod_{w \in \mathcal{N}_v \setminus \{u\}} \hat{m}_{wu}(x)\,. \tag{9.14}$$

How m and $\hat{m}$ are related to each other is a natural question. The messages $\hat{m}$ that result from the bit-length approximation resemble max-product messages [404]. Their magnitude is related to the original messages m through the following lemma.

Lemma 1. *Let $(v, u) \in E$ be an edge of $G = (V, E)$, $h_v := |\mathcal{X}_v|$ the size of vs state space, and $n_v := |\mathcal{N}_v|$ the number of its neighbors. If $h_v \geq 2 \land \forall y \in \mathcal{X}_u : \exists x \in \mathcal{X}_v : \boldsymbol{\theta}_{vu=xy} + \boldsymbol{\theta}_{v=x} + \boldsymbol{\theta}_{u=y} > 0$, then*

$$\hat{m}_{vu}(x) < m_{vu}(x) \leq \hat{m}_{vu}^{h_v}(x)\,.$$

This statement can be proven by induction over the vertex degree. Note, that this implies $M_v(y) = \prod_{w \in n_v} m_{wv}(x) \leq \prod_{w \in n_v} \hat{m}_{wv}^{h_v}(x) = \hat{M}_v^{h_v}(y)$. When it comes to the point-wise estimates of the marginal probabilities, one finds that due to the approximate messages some marginals probabilities simply cannot be present. Figure 9.1 shows edge marginal probabilities for random parameters that are generated with m and $\hat{m}$, respectively. One clearly sees how the probability space is discretized by the approximate messages. One can also see that there is an error in the approximate marginal probabilities computed with $\hat{m}$, since in case of zero error all points would lie on the diagonal.

The previous lemma helps to derive an estimate of the distance between the true outcome of the inference and the one that results from the message update Equation 9.14.

Theorem 1. *Let $\beta_v^\star := \max_y \max_u \beta_{uv}(y)$ be the maximum incoming bit length at v and assume that the preconditions of Lemma 1 hold, then*

$$D\left(p_v \| \hat{p}_v\right) \quad \in \quad \mathcal{O}\left(n_v h_v \beta_v^\star\right),$$

where $D\left(p_v \| \hat{p}_v\right)$ denotes the Kullback-Leibler (KL) divergence between the marginal probability mass function of p_v, computed with the message update $m_{vu}(y)$, and $\hat{p}_v$, computed with $\hat{m}_{vu}(y)$.

This result can be derived by plugging the BP marginals (Equation 9.9) into the definition of the KL divergence and applying Lemma 1 two times. The KL is still unbounded, since there is no bound on $\beta_v^\star$. Nevertheless, it indicates a dependence of the KL of p_v and $\hat{p}_v$ on the state space size $|\mathcal{X}_v|$ and the neighborhood size $|\mathcal{N}_v|$. This relation can also be observed in the numerical experiments in Section 9.1.4. A comprehensive discussion of how message errors generally affect the result of belief propagation can be found in [332].

9.1.3.1 Parameter Estimation

In the following, an integer parameter estimation method based on the closed form solution to the MLE is derived. Recall that $\mathbb{E}_{\mathcal{D}}\left[\phi(\boldsymbol{x})\right] = \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x} \in \mathcal{D}} \phi(\boldsymbol{x})$ and let $\boldsymbol{f} := \sum_{\boldsymbol{x} \in \mathcal{D}} \phi(\boldsymbol{x})$ and $\mathrm{bl}(a) = \lfloor \log_2 a \rfloor + 1$ the bit length of a. With this, an integer upper bound on the optimal parameters can be found.

$$\begin{aligned}\log_2 \mathbb{E}_{\mathcal{D}}\left[\phi_{v=x}(\boldsymbol{x})\right] &= \log_2 \boldsymbol{f}_{v=x} - \log_2 |\mathcal{D}| && (9.15)\\ &\leq \operatorname{bl} \boldsymbol{f}_{v=x} - \operatorname{bl} |\mathcal{D}| =: \tilde{\boldsymbol{\theta}}_{v=x} && (9.16)\end{aligned}$$

$$\begin{aligned}\log_2 \mathbb{E}_{\mathcal{D}}\left[\phi_{vu=xy}(\boldsymbol{x})\right] &\leq \operatorname{bl} \boldsymbol{f}_{vu=xy} && (9.17)\\ &- \operatorname{bl} \boldsymbol{f}_{v=x} - \operatorname{bl} \boldsymbol{f}_{vu=xy} + \operatorname{bl} |\mathcal{D}| && (9.18)\\ &=: \tilde{\boldsymbol{\theta}}_{vu=xy}\end{aligned}$$

Unfortunately, most of those estimates are negative which is not allowed due to the integer restriction. Let $s := \max_{1\leq i\leq d} -\tilde{\boldsymbol{\theta}}_i$ be the negative component of $\tilde{\boldsymbol{\theta}}$ with the largest magnitude. Now, consider the weights

$$\tilde{\boldsymbol{\theta}}^{+}_{v=x} := \boldsymbol{s} + \tilde{\boldsymbol{\theta}}_{v=x}, \qquad \tilde{\boldsymbol{\theta}}^{+}_{vu=xy} := \boldsymbol{s} + \tilde{\boldsymbol{\theta}}_{vu=xy}$$

with $\boldsymbol{s} := (s, s, \ldots, s)^\top \in \mathbb{R}^d$. Clearly, an error is induced into $\tilde{\boldsymbol{\theta}}$ by replacing $\log_2$ with bl. The following lemma shows that shifting $\tilde{\boldsymbol{\theta}}$ by $\boldsymbol{s}$ introduces no new error.

Lemma 2. *Let $\boldsymbol{s} := (s, s, \ldots, s)^\top \in \mathbb{R}^d$ and ϕ be an overcomplete sufficient statistic, then $\ell(\boldsymbol{\theta} + \boldsymbol{s}) = \ell(\boldsymbol{\theta})$.*

Proof: Since ϕ is overcomplete, it holds that $\langle \boldsymbol{s}, \phi(\boldsymbol{x})\rangle = \text{const}, \forall \boldsymbol{x}$ and hence:

$$\begin{aligned}&\ell(\boldsymbol{\theta}) - \ell(\boldsymbol{\theta} + \boldsymbol{s}) && (9.19)\\ =&\frac{1}{D}\sum_{\boldsymbol{x}\in\mathcal{D}} \log \frac{\sum_{\boldsymbol{y}\in\mathcal{X}} \exp\langle \boldsymbol{\theta} + \boldsymbol{s}, \phi(\boldsymbol{y})\rangle}{\exp\langle \boldsymbol{s}, \phi(\boldsymbol{x})\rangle \sum_{\boldsymbol{y}'\in\mathcal{X}} \exp\langle \boldsymbol{\theta}, \phi(\boldsymbol{y}')\rangle} = 0.\end{aligned}$$

ϕ as defined in Section 9.1.2 is actually overcomplete. This can now be used to bind the training error of the shifted integer parameters $\tilde{\boldsymbol{\theta}}^{+}$.

Theorem 2. *Let $-s$ be the smallest value in the vector $\mathbb{E}_{\mathcal{D}}\left[\phi(\boldsymbol{x})\right]$. Furthermore, let $\boldsymbol{\theta}^{\star}_i := s + \log \mathbb{E}_{\mathcal{D}}\left[\phi_i(\boldsymbol{x})\right]$ and $\tilde{\boldsymbol{\theta}}^{+}_i := s + \operatorname{bl} \mathbb{E}_{\mathcal{D}}\left[\phi_i(\boldsymbol{x})\right]$ then*

$$\ell(\boldsymbol{\theta}^{\star}) - \ell(\tilde{\boldsymbol{\theta}}^{+}) \leq \|\nabla f(\tilde{\boldsymbol{\theta}}^{+})\|_1.$$

The result follows from the previous lemma, convexity, and the Cauchy-Schwarz inequality. Since each component of the gradient is a difference of two probabilities, its magnitude cannot be greater than 1. Hence, the gradient norm can be at most d. In the following section, the magnitude of the gradient relative to d is evaluated numerically.

Either due to restrictions in wordsize ω or for enlarging the number of representable marginal probabilities, a final scaling of the parameters might be desired. To allow an appropriate integer scaling, the parameter K is introduced. Let $s := \max_{1\leq i\leq d} -\tilde{\boldsymbol{\theta}}_i$ be the negative component of $\tilde{\boldsymbol{\theta}}$ with the largest magnitude and $m := \max_{1\leq i\leq d} \tilde{\boldsymbol{\theta}}_i$ be the positive component of $\tilde{\boldsymbol{\theta}}$ with the largest magnitude. The final integer parameters are computed by

$$\bar{\boldsymbol{\theta}}_{v=x} := \left\lfloor \frac{K}{s+m} \tilde{\boldsymbol{\theta}}^{+}_{v=x} \right\rfloor , \qquad \bar{\boldsymbol{\theta}}_{vu=xy} := \left\lfloor \frac{K}{s+m} \tilde{\boldsymbol{\theta}}^{+}_{vu=xy} \right\rfloor . \tag{9.20}$$

Thus, $\tilde{\boldsymbol{\theta}}^{+}$ is rescaled such that $\bar{\boldsymbol{\theta}} \in \{0, 1, \ldots, K\}^{d}$, which may also be interpreted as implicit base change. Note, that unless $K = (s+m)$, the parameter vector is scaled and an additional error is added to the gradient. Hence, the impact of K is empirically evaluated in Section 9.1.4. The method of choosing parameters according to Equation 9.20 is called *direct integer estimation.*

Gradient-Based Estimation As already mentioned in Section 9.1.2, in certain situations, it might be desired to estimate the parameters with gradient-based methods. Unfortunately, the partial derivatives from Equation 9.7 are not integers. Hence, the expression must be rearranged to obtain an integer form. Let $\boldsymbol{f} := \sum_{\boldsymbol{x}\in\mathcal{D}} \phi(\boldsymbol{x})$, so that

$$\begin{aligned} &\left\lceil \sum_{x\in\mathcal{X}_v} \hat{M}^{\star}_{v}(x) \right\rceil |\mathcal{D}| \frac{\partial \ell(\boldsymbol{\theta} \mid \mathcal{D})}{\partial \boldsymbol{\theta}_{X_v=x_v}} \\ &= \left\lceil \sum_{x\in\mathcal{X}_v} \hat{M}^{\star}_{v}(x) \right\rceil \boldsymbol{f}_{v=x} - |\mathcal{D}|\, \hat{M}^{\star}_{v}(x_v). \end{aligned} \tag{9.21}$$

This scaled version of the partial derivative is an integer expression that can be computed by using only integer addition, multiplication, and binary bit shift. The common gradient descent update makes use of a step size η to determine how far the current weight vector should move in the direction of the gradient. The smallest possible step size in integer space is 1. This means that any parameter can either be increased or decreased by 1. Therefore, in the beginning of an integer gradient-based optimization, all the model parameters are 0 and the gradient will tend to increase a large number of parameters. This results in a rather slow convergence, since due to the fixed step size of 1, most of the parameters are worse than before the update. To compensate, we suggest updating, for each clique only the parameter for which the corresponding partial derivative has the largest magnitude. This method is used when estimating the CRF parameters in the following section.

9.1.4 Numerical Results

The previous sections pointed to various factors that may have an influence on the training error, test performance, or runtime of the integer approximation. In order to show that integer undirected models are a general approach for approximate learning in discrete state spaces, generative and discriminative variants of undirected models are evaluated on synthetic data and real-world data. We consider in particular the following methods:

RealMRF The classic generative undirected model as described in Section 9.1.2.

RealCRF The discriminative classifier as it is defined in [407, 655].

IntMRF The integer approximation of generative undirected models as described in Section 9.1.3.

IntCRF The integer approximation of discriminative undirected models. Further details are explained in Section 9.1.4.5.

Both real variants are based on floating-point arithmetic. In the MRF experiments, the model parameters are estimated from the empirical expectations by Equations 9.5 and 9.20. Parameters of discriminative models are estimated by stochastic gradient methods [655]. Each MRF experiment was repeated 100 times on random input distributions and graphs. In most cases, only the average is reported, since the standard deviation is too small to be visualized in a plot. Whenever MAP accuracy is reported, it corresponds to the percentage of correctly labeled vertices, where the prediction is computed with Equation 9.3.

Of course, the implementations of the above-mentioned methods are equally efficient, e.g. the message computation (and therefore the probability computation) executes exactly the same code for all methods, except for the arithmetic instructions. A subsets of the results is presented below. Unless otherwise explicitly stated, the experiments are done on an Intel Core i7-2600K 3.4 GHz (Sandy Bridge architecture, Table 9.1) with 16 GB 1 333 MHz DDR3 main memory. An implementation of integer Markov random fieldsis available as part of the Python package pxpy.[3]

Synthetic Data In order to achieve robust results that capture the *average behavior* of the integer approximation, a synthetic data generator has been implemented that samples random empirical marginals with corresponding MAP states. Therefore, a sequential algorithm for random trees with given degrees [57] generates random tree structured graphs. For a random graph, the weights $\boldsymbol{\theta}_i^\star \sim \mathcal{N}(0, 1)$ are sampled from a Gaussian distribution. Additionally, for each vertex, a random state is selected that gets a constant extra amount of weight, thus enforcing low entropy. The weights are then used to generate marginals and MAP states with the double precision floating-point variant of belief propagation. The so generated marginals provide empirical input distribution. The MAP state is then compared with the MAP state that is estimated by IntMRF and RealMRF for the given empirical marginals.

CoNLL-2000 Data This dataset was proposed for the shared task at the Conference on Computational Natural Language Learning in 2000. The train and test data consist

3 https://pypi.org/project/pxpy.

of three columns separated by spaces. Each word has been put on a separate line and there is an empty line after each sentence. The first column contains the current word, the second contains its part-of-speech tag as derived by the Brill tagger, and the third contains its chunk tag as derived from the Wall Street Journal corpus. The chunk tags contain the name of the chunk type—I-NP for noun phrase words and I-VP for verb phrase words, say. Most chunk types have two types of chunk tags, B-CHUNK for the first word of the chunk and I-CHUNK for each other word in the chunk. In total, there are 22 chunk tags that correspond to the vertex states, i.e., $|\mathcal{X}| = 22$. For the computation of per chunk F_1-score, a chunk is treated as correct if and only if all consecutive tags that belong to the same chunk are correct. The dataset contains $8,936$ training instances and $2,012$ test instances. For each word, the surrounding words and part-of-speech tags are used as features. Because of the inherent dependency between neighboring vertex states, this dataset is especially well suited for evaluation if the dependency structure between vertices is preserved by the integer approximation.

9.1.4.1 The Impact of $|\mathcal{X}|$ and $|\mathcal{N}_v|$ on Quality and Runtime

In Section 9.1.3 an estimate of the error in marginal probabilities that are computed with bit length BP indicates that the size of a vertex state space $|\mathcal{X}_v|$ and the degree $|\mathcal{N}_v|$ have an impact on the training error. Figure 9.2 shows the training error in terms of normalized negative log-likelihood, the testing error in terms of MAP accuracy, and the runtime in seconds for various values of $|\mathcal{X}_v|$ and $|\mathcal{N}_v|$ for an increasing number of vertices on the synthetic data. Each curve is the average over 100 random trees with random parameters and $K = 8$. The results with varying $|\mathcal{X}_v|$ are generated with a maximum degree of 8 and the ones for varying $|\mathcal{N}_v|$ are generated with $|\mathcal{X}_v| = 4$.

In terms of training error, the top-left plot shows a clear offset between integer and floating-point estimates for the same number of states. In terms of varying degrees (center-left), the training error of the integer model shows a response to different neighborhood sizes, whereas the likelihood of the floating-point model is invariant against the degrees. A similar picture is drawn for the dependence of the test accuracy with $|\mathcal{X}|$ and $|\mathcal{N}_v|$ (top-right, top-center). The floating-point MAP estimate is not changed by an increasing number of states and neighbors, whereas the integer MRF shows a clear response. The accuracy of the integer MRF actually increases with increasing degrees. In general, the quality of the models seems to be independent of the number of vertices in the graph.

The floating-point model outperforms the approximate integer model in terms of the MAP accuracy and negative log-likelihood. However, the two plots at the bottom of Figure 9.2 show that the resource consumption in terms of clock cycles is largely reduced by the integer model. Time is measured for estimating parameters, computing the likelihood, and performing a MAP prediction. Since both algorithms (RealMRF and IntMRF) share exactly the same asymptotic complexity for these procedures, the substantial reduction in runtime that is shown by the results must be due to the reduction in clock cycles.

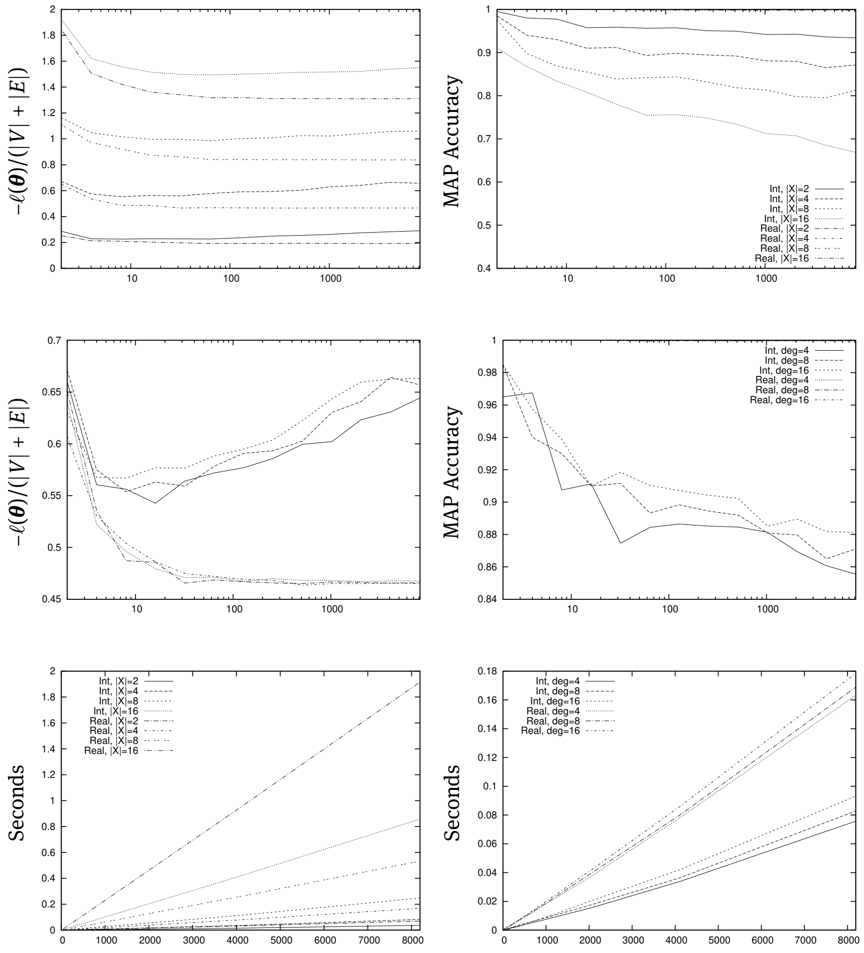


Fig. 9.2: MRF training error, test accuracy, and runtime in seconds for different choices of the state space size ($\mathcal{X}$) and maximum degree as a function of the number of vertices. Each of the top two rows shares legends and has an x-axis in logarithmic scale.

9.1.4.2 The Contribution of *K* to Quality

It might be convenient to scale the integer parameters such that the resulting parameter vector $\bar{\boldsymbol{\theta}}$ is in the set $\{0, 1, \dots, K\}$. We illustrate the effect of such scaling by the response of the integer model in terms of training quality and test error, as shown in the two plots on the top of Figure 9.3. The training error seems to be a smooth function of K whereas the MAP accuracy is sensitive to the choices of K. This is what we expected, since a large K basically means that a larger number of marginal probabilities can be represented. One can also see that, as soon as K is large enough (i.e., $K = 8$ in the right plot at the top of Figure 9.3), a further increase does not show any significant impact on either training error and test accuracy. Both results where generated on graphs with a maximum degree of 8, but as already known from the previous experiment, the effect of different degrees on the model's quality is negligible. The bottom-left plot in Figure 9.3 shows the width of the intrinsic parameter space, i.e., the sum of the smallest and the largest integer parameter before rescaling is performed. It turns out, that the width of the parameter space is naturally bounded, since $s + m$ seems to converge on the same value for various configurations of n and $\mathcal{X}$. Plotting s and m separately shows that the dynamics in $s + m$ are mainly influenced by the smallest parameter s, i.e., the width of the parameter space must increase in order to represent smaller probabilities.

9.1.4.3 The Impact of *K* on the Gradient Norm

As indicated by the analysis of the training error in Section 9.1.3, the distance between the maximum likelihood estimate and the result of the direct integer parameter estimation is basically bounded by the gradient norm of the integer parameters $\bar{\boldsymbol{\theta}}$. Since the components of the gradient cannot exceed 1, a trivial upper bound for the gradient norm is d, the dimension of the parameter vector. A very strong observation can be made in the rightmost picture which shows the relative gradient norm for an increasing number of vertices and various values K. This result suggests that there exists a bound on the relative gradient norm that is independent of the number of vertices and that this bound decreases with increasing K.

9.1.4.4 Integer Models on Resource-Constrained Devices

The motivation for the integer model was to save resources in terms of clock cycles. We can now demonstrate that the impact of this reduction is larger, if the underlying architecture is weaker, i.e., has slower floating-point arithmetic. The two bar charts in Figure 9.4 show a runtime comparison of the integer MRF on two different CPU architectures. One is Sandy Bridge, which has also been the platform for all the other experiments; the other is a Raspberry Pi device with ARM11 architecture. As expected, the integer model actually speeds up the execution on the Pi device more than on the other architecture, i.e., the Raspberry Pi gains a speedup of 2.56× and Sandy Bridge a speedup of 2.34×. In terms of standard deviation, the ARM11 architecture is more stable than the Sandy Bridge, which might be a result of a more sophisticated out-of-order instruction execution in the latter architecture.

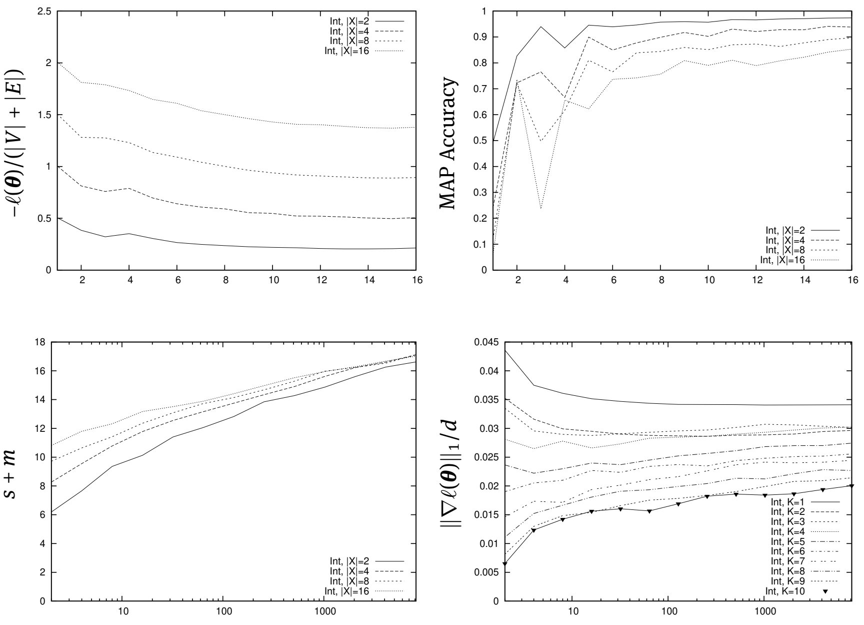


Fig. 9.3: Top: negative log-likelihood and MAP accuracy of MRF as a function of K. Bottom: the left plot shows how the width of the parameter space behaves as a function of the number of vertices (in log-scale) for different state space sizes, whereas $-s$ is the smallest and m the largest element of the corresponding estimated parameter vector. The relative norm of the gradient for various values of K is shown on the right.

9.1.4.5 Training Integer CRF with Stochastic Integer Gradient Descent

In the last evaluation, the randomized stochastic gradient training of discriminative models is investigated. An integer linear-chain CRF is constructed and trained by a stochastic gradient-descent algorithm. In case of the integer CRF, the parameter updates are computed by means of the scaled integer gradient (cf. the end of Section 9.1.3). Both algorithms perform 20 passes over the training data, each pass looping through the training instances in random order. This was repeated 50 times in order to compute an estimate of the expected quality of the randomized training procedure. The parameter update for the floating-point CRF is computed with the step size $\eta = 10^{-1}$. The ratio of quality per runtime is presented in Figure 9.4, where the negative log-likelihood is averaged over all training instances and the accuracy is computed with regard to the chunk tags. Chunk type precision, recall, and F_1-score are shown in Table 9.2, whereby the overall F_1-score for a model with $\boldsymbol{\theta} = \mathbf{0}$ is $\approx 26\,\%$. As desired, the performance of the integer approximation is reasonable. Except for one chunk type (INTJ), precision, recall, and F_1-score have a relatively small standard deviation of about 2 %. The precision is three times better using integer CRF; the recall and F_1-score are one time better using integer CRF than real CRF. IntCRF is substantially worse than RealCRF only for the

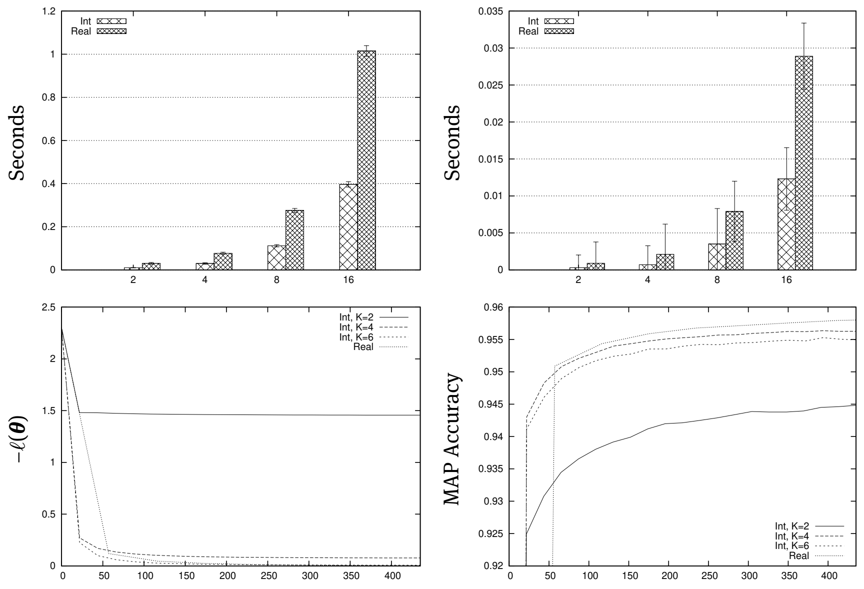


Fig. 9.4: Top: Runtime comparison of integer and floating-point MRF on two architectures for a varying number of states. Left: Raspberry PI @ 700 MHz (ARM11). Right: Intel Core i7-2600K @ 3.4 GHz (Sandy Bridge). Bottom: progress of stochastic gradient training in terms of training error and test accuracy of the CRFs over running seconds.

verb phrase (VP). For many real-world applications, this is a price that can be paid for IntCRF being about twice as fast as RealCRF.

9.1.5 Conclusion

In this contribution, integer undirected graphical models have been introduced, together with algorithms for probabilistic inference and parameter estimation that rely only on integer arithmetic. Generative and discriminative models have been evaluated in terms of prediction quality and runtime. We learned that optimal integer model parameters typically take values with small magnitude, reducing the storage requirement when compared with 64-bit double precision numbers. This allows us to sample from high-dimensional generative models and to use structured discriminative classifiers, even on computational devices without or with a slow floating-point unit, or in situations where energy has to be saved.

Tab. 9.2: The difference of RealCRF and IntCRF precision (Prec), recall (Rec), and F_1-score (F_1) averaged over 50 repetitions on the CoNLL-2000 dataset. Values are in percentage, i.e., a value of −0.29 means that the integer CRF was on average 0.29 % better at the corresponding measure than its floating-point counterpart.

	ADJP	ADVP	CONJP	INTJ	LST	NP	PP	PRT	SBAR	VP	Overall
Prec	−0.29	−0.46	4.81	−15.56	0	1.11	0.33	3.78	1.15	6.09	0.73
Rec	1.61	2.69	−3.55	0	0	0.34	0.13	1.82	1.78	6.31	0.48
F_1	0.74	1.17	1.60	−4.03	0	0.73	0.23	2.85	1.50	6.20	0.61

9.2 Power Consumption Analysis and Uplink Transmission Power

Robert Falkenberg

Abstract: The penetration of wireless communication systems in industrial and private environments is constantly increasing due to their flexible and mobile application possibilities. Wearables, smartphones, or industrial systems for tagging, tracking, and sensing are only a few examples from the tremendous variety of such systems. However, unleashing these systems from the power grid also means that the available energy is a limited resource that must be conserved and managed prudently.

The estimation of energy consumption by the communication system differs significantly from the other components, as it is strongly dependent on external influences. These include the quality of the radio channel, the channel access scheme, and the utilization of the shared transmission medium by other participants, who are often not part of the actual system. Data transmissions can last longer, require a higher transmission power, or fail due to collisions, so that they have to be repeated. The consequence is a longer activity time of the transceiver and a shorter dwell time in the efficient power saving mode. Therefore, realistic simulation models are required at design time, which take into account the properties of the communication interface as well as the external environment.

In the following, methods for modeling power consumption for different communication technologies are discussed. This includes decentralized narrow-band communication in the Short Range Devices (SRD) band and the comprehensive modeling of cellular technologies such as Long Term Evolution (LTE), LTE-Advanced (LTE-A) and Narrow Band Internet of Things (NB-IoT) by a Context-Aware Power Consumption Model (CoPoMo).

It is shown that a decentralized channel access with brisk activity on the radio channel leads to an increased power consumption of all waiting subscribers, if the channel occupancy is to be tracked continuously to keep the transmission latency as low as possible. Conversely, in centrally organized cellular radio networks, the energy consumption of the User Equipment (UE) is dominated by uplink transmissions, especially when high transmission power is required. The proportion for reception, however, depends mainly on the duration of the transmission. In fact, adding an additional reception path via Carrier Aggregation (CA) not only increases the data rate, but also reduces the energy consumption of the UE .

Since the knowledge of the transmit power is essential for the estimation of the power consumption, but most UEs do not provide this information to the application layer, a Machine Learning (ML)-based method for estimating the transmit power from

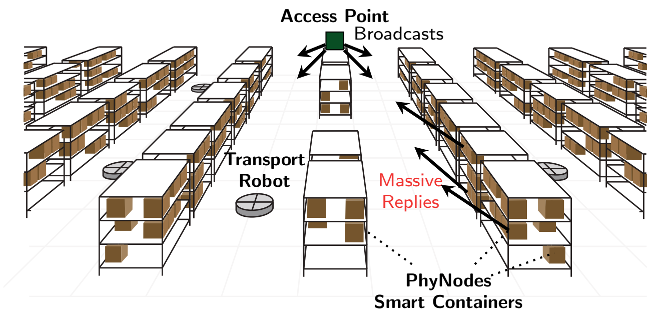


Fig. 9.5: Warehouse scenario. ©[2018] IEEE. Reprinted, with permission, from [206].

the available parameters such as strength and quality of the received signal, is also presented.

9.2.1 Introduction

Instead of treating inventory items as static resources, future intelligent warehouses will turn containers to become Cyber Physical Systems (CPSs) that actively and autonomously participate in the optimization of the logistical processes. Consequently, new challenges that are system-immanent for the massive Internet of Things (IoT), such as channel access in a shared communication medium, have to be addressed.

An example of such a warehouse scenario is shown schematically in Figure 9.5. A wide variety of autonomous transport systems are used to transport goods into or out of the warehouse. The individual goods are stored in smart containers that can provide information about their current contents and the goods they contain at any time by radio. Energy supply is a particular challenge for the embedded systems used for this purpose. Mains and battery operation are ruled out due to the size of the location, so that the platforms must obtain their energy for operation and communication through *energy harvesting*, using photovoltaic, say, and must manage it extremely efficiently.

To fetch a specific inventory, distributed Access Points (APs) transmit inventory queries to the warehouse. These are answered by containers with matching contents, specifying the quantity contained. Subsequently, the transport systems bring the requested quantity to a picking point for further use.

Since such requests can lead to massive replies depending on the distribution of goods and inventory, channel access must be coordinated to avoid collisions during transmission. Distributed channel access methods quickly reach their capacity limits and increase the energy consumption of network subscribers due to collisions, multiple transmissions, and prolonged waiting for a free channel. In this area, CRC 876 has developed and brought together innovative methods for recording, analyzing, and optimizing energy consumption [206, 209], which are discussed in the following subsections.

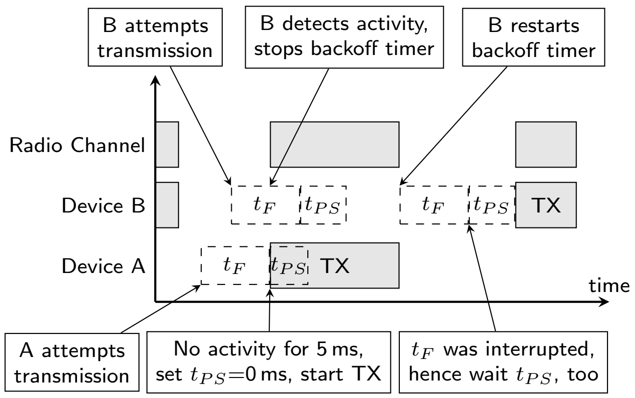
This contrasts with mobile communications networks that have centrally organized channel access, which are discussed later in this section. If we accept the restriction of operating only one specific technology on a frequency band, higher spectral efficiency can be achieved in return. Techniques such as central power control or inter-cell interference coordination enable resource-efficient transmission even at high subscriber density. Numerous studies of the CRC 876 have shown that the energy consumption of current mobile radio terminals (UE) is dominated by the transmission of data, especially at high transmission power. The specially developed CoPoMo enables a wide variety of trade-offs, e.g., between transmission time, energy consumption, and spectral resource requirements, for different frequency ranges, building densities, and mobility profiles. Section 9.2.4 introduces the basic concepts of CoPoMo and presents two studies, one dealing with a trade-off between transmission bandwidth and energy consumption, and the other presenting an ML-based method for the UE-based estimation of transmission power using available quality indicators.

9.2.2 Power Consumption with Distributed Channel Access

In unlicensed bands, a distributed channel access method is often used to enable fair coexistence of different technologies. These bands include the Industrial Scientific Medical (ISM) band at 2.4 GHz and the SRD band at 868 MHz. The latter is used for the communication of the PhyNode. Distributed channel access is based on the Listen Before Talk (LBT) principle, which is known in a similar form as Carrier Sense Multiple Access Collision Avoidance (CSMA/CA) for WLAN and ZigBee networks. For the SRD band, channel access is specified by European Telecommunications Standards Institute (ETSI), which is shown schematically in Figure 9.6. Stations with a transmission intent hold back their transmission until the transmission channel is free. When the channel becomes free, the system waits an additional backoff time $t_L = t_F + t_{PS}$ with $t_F = 5$ ms. Thereby t_{PS} is randomly selected for each startup in the interval 0 ms to 5 ms. If the channel is still free after t_L has elapsed, the transmission is carried out. Otherwise, the system waits again for a free channel including a newly selected backoff time t_L. For acceleration in case of low channel utilization, a station can set t_{PS} to 0 ms for the first transmission attempt if the channel is continuously free between the initial transmission request and the expiration of t_F.

Figure 9.7 shows the channel occupation by three stations in the radio spectrum. At the beginning of the recording, the channel is continuously occupied by a jammer. The three stations already have a transmission intent and hold back their transmission for the time being. After switching off the jammer, the three stations transmit one after the other according to the access scheme and the random backoff intervals.

However, short s result in low channel utilization and thus in reduced spectral efficiency due to the relatively long waiting times. In addition, the channel access method requires continuous monitoring of the radio channel between the arrival of the



B attempts transmission
B detects activity, stops backoff timer
B restarts backoff timer
Radio Channel
Device B
Device A
t_F
t_{PS}
TX
time
A attempts transmission
No activity for 5 ms, set t_{PS}=0 ms, start TX
t_F was interrupted, hence wait t_{PS}, too

Fig. 9.6: Timeline of the LBT access scheme. Dashed blocks represent back-off intervals and solid blocks indicate an occupied channel by a transmission over the air. ©[2017] IEEE. Reprinted, with permission, from [209].

transmission intent and the actual execution of the transmission. The duration of this monitoring increases with the utilization of the channel. Since the receive circuits must be active during this time, the power consumption of all competing stations increases significantly.

Figure 9.8 shows the distribution of the transceiver's energy consumption as a function of the number of simultaneously active devices responding to 10 product requests in the warehouse scenario (cf. Figure 9.5). The energy accounting is obtained from an energy-aware driver model. The measurement includes the constant part for the reception of the 10 requests and an additional message to terminate the measurement after 11.75 s, as well as the variable energy consumption for sending the replies. Compared with the empty channel, the energy consumption increases by up to a factor of 10 in case of more than 30 stations.

To enable an optimization of energy consumption given the scarce resource, a simulative hardware-in-the-loop design space exploration framework was developed, which is discussed in more detail in the following section.

9.2.3 Simulative Access-Scheme Optimization

In this section, we present a multi-methodological system model that brings together testbed experiments for measuring real hardware properties and simulative evaluations for large-scale considerations [206]. As a case study, we focus on parametrization of the 802.15.4-based radio communication system, which has to be energy-efficient due to the scarce amount of harvested energy, but avoid latencies for the maintenance of scalability of the overlaying warehouse system. The results show that a modification of

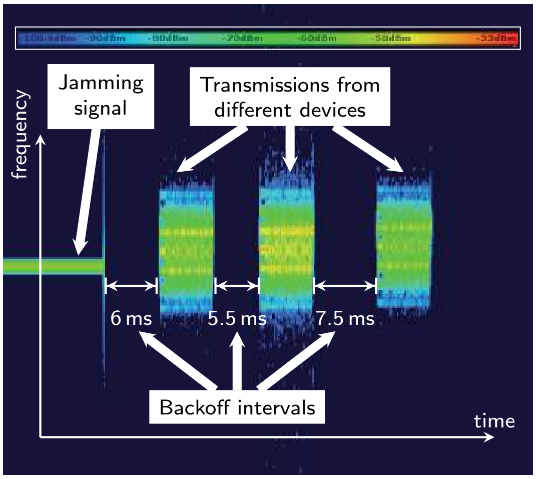


Fig. 9.7: Spectral view of LBT scheme in action. Three stations organize their pending transmissions when the channel becomes free (simulated by disabling a jamming signal).

the initial backoff time can lead to both energy and time savings in the order of 50 % compared with the standard.

Figure 9.9 shows the underlying modeling principle. On the right side is the physical system in the form of the PhyNetLab, in which a field evaluation can be performed [206]. The left side comprises the OMNeT++ simulation system, which models the communication system including the application layer in the form of a simulation. For a given system scenario with information on energy consumption and dwell time of individual operating states, data volume, available energy, and the number of network subscribers, a simulative optimization of the communication system is carried out that enables a trade-off between conflicting target variables, e.g. latency and energy consumption. This configuration is transferred to the physical testbed and evaluated in field experiments.

The energy consumption of the communication system of the PhyNode is modeled in the simulation as a state machine with four states (cf. Figure 9.10). In the LISTEN state, the device periodically listens on the channel for preambles that indicate the beginning of a new packet for reception. When this occurs, the transceiver enters receive mode (RX) to receive the packet and then returns to LISTEN. If a packet is to be transmitted, it enters the BACKOFF state with repeated short dwelling times in the RX mode to wait for the free channel. After the backoff timer expires, it finally sends the packet in TX mode. The consumption values of the individual energy states can be automatically captured and fed into the simulation using the hardware-in-the-loop approach and the energy-aware driver models.

Based on the presented framework, the channel access procedure can be optimized for a given warehouse scenario. One of the most common processes in a self-inventory

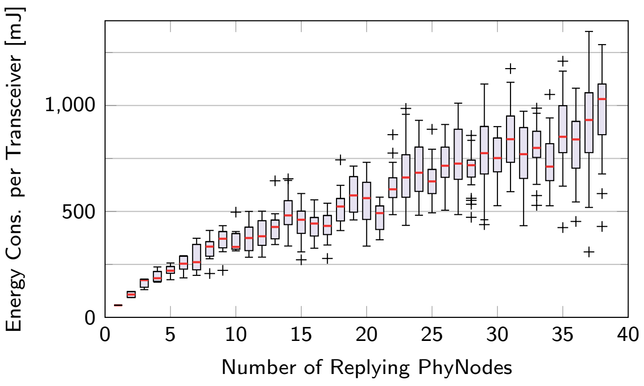


Fig. 9.8: Energy consumption of the radio transceiver for receiving 11 packets (queries) and transmitting 10 replies in a constant interval of 11.75s. ©[2017] IEEE. Reprinted, with permission, from [209].

warehouse is the request for specific products in a required quantity. For this purpose, a request is sent out as a broadcast and answered by matching containers, i.e. the communication systems located on them. Depending on the equipment of the warehouse and the usually requested quantity, only a fraction of the responses is sufficient to fulfill the requested product quantity. This value is called the Minimum Query Response Ratio ($\mathrm{QRR}_{\mathrm{min}} \in [0, 1]$).

The issued requests cause a large number of participants to attempt to send their responses at the same time. They select a random backoff and then send their packets. However, if the backoff can take on only a few discrete values compared with the number of subscribers, collisions inevitably occur, so that transmissions have to be repeated and the response time until $\mathrm{QRR}_{\mathrm{min}}$ is reached is increased as a result. To resolve such collisions, the 802.15.4 standard defines an exponential increase of the backoff window in the form of a backoff exponent BE, which is successively increased in case of collisions and then reset to the initial value BE_0. The minimum backoff exponent BE_0 is an optimization parameter that has to be chosen depending on the expected number of participants and the permitted delay in case of a smaller number of participants.

Figure 9.11 shows the exemplary application of the framework to optimize the initial backoff exponent BE_0 for different $\mathrm{QRR}_{\mathrm{min}}$.

The results show that for $\mathrm{QRR}_{\mathrm{min}} = 0.8$, an initial $\mathrm{BE}_0 = 8$ compared with $\mathrm{BE}_0 = 3$ for 420 responding nodes reduces the energy consumption by 49 % while reducing the time to fulfill the request by 56 %. However, even with smaller numbers of nodes, choosing a larger BE_0 has a positive effect on both objectives. However, at lower $\mathrm{QRR}_{\mathrm{min}} = 0.2$, a larger BE_0 leads to higher energy consumption in favor of reduced response time.

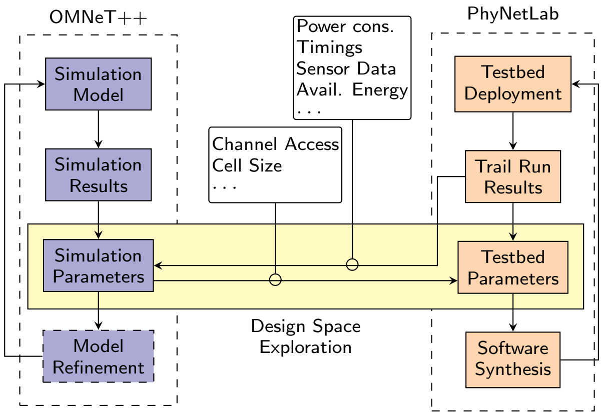
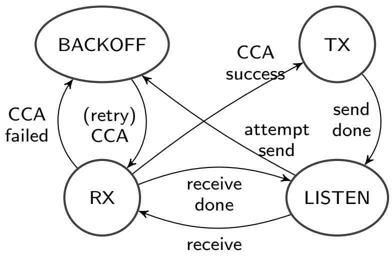


Fig. 9.9: System model for design-space exploration. ©[2018] IEEE. Reprinted, with permission, from [206].



State	Avg. Power
BACKOFF	1.5 µW
LISTEN	4.5 mW
RX	70 mW
TX	138 mW

Fig. 9.10: State machine model of the transceiver. ©[2018] IEEE. Reprinted, with permission, from [206].

9.2.4 Power Consumption in Cellular Networks

Due to the increasing spread and popularity of cellular radio networks for networking the smallest mobile devices, the analysis and optimization of energy efficiency is also gaining importance in this domain. Centralized control by static infrastructure, i.e. by the base station and backhaul, enables resource optimization without the intervention of the end devices. For example, it can be considered whether distant stations with poor channel conditions receive a greater short-term pensum of spectral resources to perform their transmission than stations with good channel conditions that can still achieve high data rates using lower transmit powers.

Optimizations of power consumption, however, require precise power consumption models that on the one hand provide accurate estimates and yet can be calculated efficiently. For this purpose, CRC 876 has contributed the CoPoMo [192], a Markovian power consumption model for calculating the power consumption of current LTE and LTE-A terminals. The calculation takes into account device-specific consumption char-

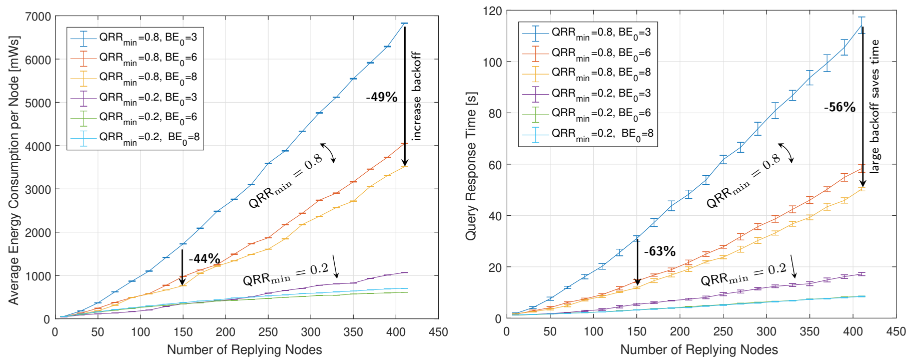


Fig. 9.11: Simulation results of the energy consumption (left) and query response time (right) as a function of the number of concurrently replying nodes for different backoff configurations and minimum query response ratio. ©[2018] IEEE. Reprinted, with permission, from [206].

acteristics as well as spectral resource utilization, the frequency range used, mobility, built environment, and the type of data traffic.

The following sections introduce the basic concepts of CoPoMo, and then present extensions and case studies for resource optimization.

9.2.4.1 Context-Aware Power Consumption Modeling

This section introduces the basic concepts of CoPoMo [192]. As in all communication systems, the power consumption of a UE depends on the current operating state, which in turn is influenced by numerous context and system parameters. The power consumption is caused by the digital signal processing and the operation of the High Frequency (HF) components for receiving and transmitting the radio signals. Due to the use of Application-Specific Integrated Circuits (ASICs), the signal processing is characterized by a relatively low power consumption, which scales only insignificantly with the effective data throughput, and can thus be assumed to be constant in many cases during an ongoing transmission. The consumption by the HF receiver is also usually not influenced by the received field strength and the effective data throughput, and thus also assumes a constant value for the respective frequency band during reception [191].

The power consumption of the UE is dominated by the consumption of the power amplifier for the transmission of messages in the uplink, especially at high transmission powers for overcoming a high path loss [191]. Figure 9.12 shows the average power consumption of a smartphone as a function of the transmit power of the power amplifier. The measurement covers the entire system, i.e. including the main processor, display, signal processing and all active HF components. Background activities and display brightness were reduced to a constant minimum.

A small increase in power at low transmitting powers and a steep increase in power consumption at high transmitting powers can be observed. The reason for this is the

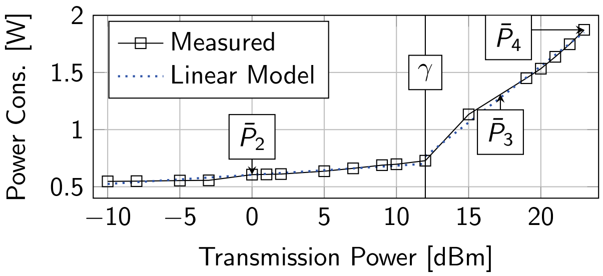


Fig. 9.12: Power consumption of a Samsung Galaxy S5 smartphone at 800 MHz in relation to transmission power. ©[2017] IEEE. Reprinted, with permission, from [208].

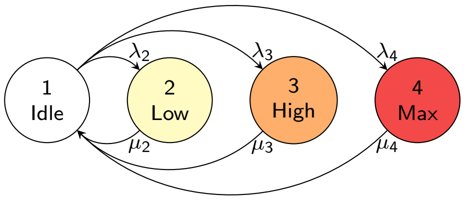


Fig. 9.13: Markovian power state model of LTE User Equipment (UE). ©[2017] IEEE. Reprinted, with permission, from [208].

typical use of two different power amplifiers with different operating ranges, which are switched to increase efficiency depending on a threshold value γ. Since the power consumption within the respective operating ranges is approximately linear, CoPoMo uses two linear models for power estimation, consisting of the respective slopes α, the y-intercept β, and the switching point γ, which are determined by empirical measurement series for each system and frequency band.

Further investigation has shown that the power consumption can be accurately estimated using four reference points of the linear model $\bar{P}_1$, $\bar{P}_2$, $\bar{P}_3$ and $\bar{P}_4$, so that the power consumption of an LTE UE can be described by a state model consisting of four corresponding states [190]. The state model is shown in Figure 9.13. State 1 represents the power consumption in idle state without outgoing data transmission. State 2 represents transmission with low transmit power (the point at 0 dB), state 3 represents transmission with high transmit power (midpoint between γ and maximum transmit power), and state 4 represents transmission with maximum transmit power of 23 dBm.

Transitions between states are given in terms of transition probabilities λ_i and μ_i and always lead over state 1, since state 2, 3, and 4 present states with outgoing transmission, during which the UE remains in this current state and is not able to switch into a different transmission-related power state. The state transitions are obtained

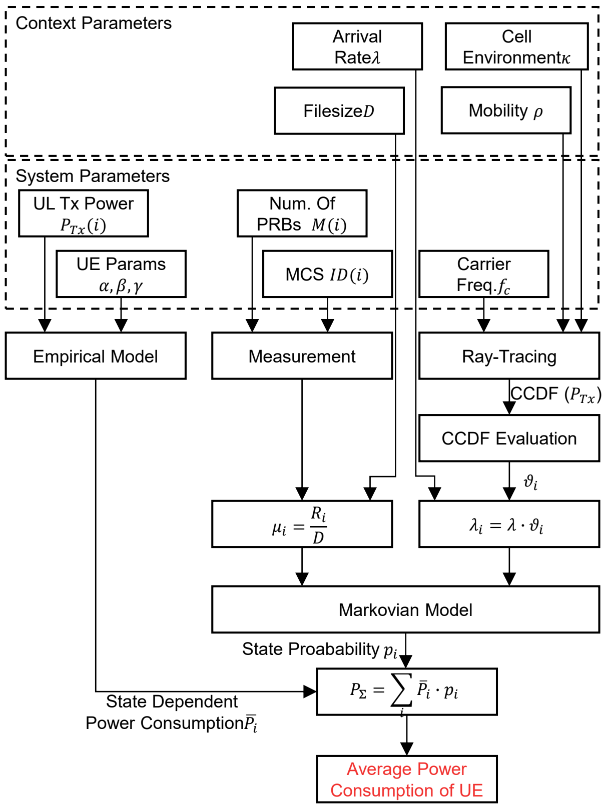


Fig. 9.14: Overview of CoPoMo. ©[2013] IEEE. Reprinted, with permission, from [192].

from the augmented overall model, which is shown in Figure 9.14. λ_i is calculated as $\lambda_i = \lambda \cdot \vartheta_i$, where $\frac{1}{\lambda}$ corresponds to the arrival rate of outgoing data transmissions and ϑ_i with $\sum_{i=2}^{4} \vartheta_i = 1$ indicates the distribution of the residence time in states 2, 3, and 4. The latter depends on the cell environment κ, mobility ρ, and carrier frequency f_c, and can be determined, say, by ray-tracing analysis and the statistical evaluation of path loss. μ_i is the inverse of the service rate and is calculated as $\mu_i = \frac{R_i}{D}$ with average file size D and the average uplink data rate R_i achieved in state i. The data rate in turn depends on the number of allocated RBs $M(i)$ and the Modulation and Coding Scheme (MCS) $ID(i)$, which are dynamically allocated according to the base station's scheduling strategy.

Finally, state probabilities can be determined from the transition probabilities, which then describe the average residence time in each state. The average power consumption of the UE can be determined together with the state-specific power consumption.

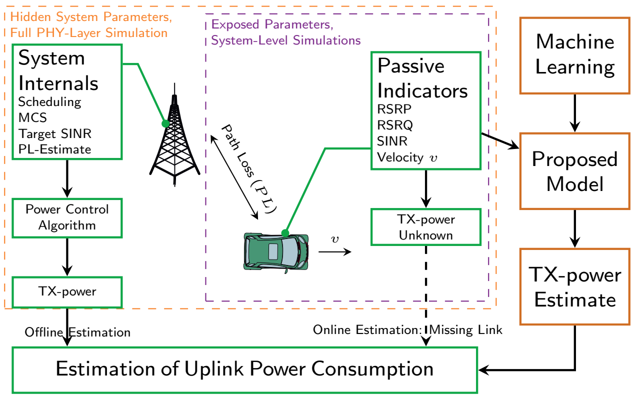


Fig. 9.15: Uplink power control of UE and its underlying system parameters are typically hidden to the application layers. However, this knowledge is crucial for predictions and estimations of the involved power consumption in energy-aware applications and system-level simulations. The proposed model derives this information from passive connectivity indicators. ©[2018] IEEE. Reprinted, with permission, from [207].

9.2.5 Uplink Power Prediction with Machine Learning

This section summarizes the work on ML-based uplink power prediction according to [207]. Energy-aware system design is an important optimization task for static and mobile IoT-based sensor nodes, especially for highly resource-constrained vehicles such as mobile robotic systems. For 4G/5G-based cellular communication systems, the effective transmission power of uplink data transmissions is of crucial importance for the overall system power consumption. Unfortunately, this information is usually hidden within off-the-shelf modems and mobile handsets and can therefore not be exploited for green communications. Moreover, the dynamic transmission power control behavior of the mobile device is not explicitly modeled in most of the established simulation frameworks.

In order to close this gap, we present a novel machine learning-based approach for forecasting the uplink transmission power used for data transmissions based on available passive network quality indicators and application-level information. A schematic illustration of the proposed solution approach is shown in Figure 9.15. The key idea is to leverage an SDR (Software-Defined Radio)-based measurement setup—capable of simultaneously determining the uplink transmission power P_{TX} and different network context indicators—in order to derive a machine learning-based prediction model that infers P_{TX} from the context measurements. This model can then be deployed to other platforms that are not capable of determining P_{TX} on their own.

The required machine learning model is derived from comprehensive field measurements of drive tests performed in a public cellular network and can be parameterized for integrating all measurements that a given target platform is able to provide for the

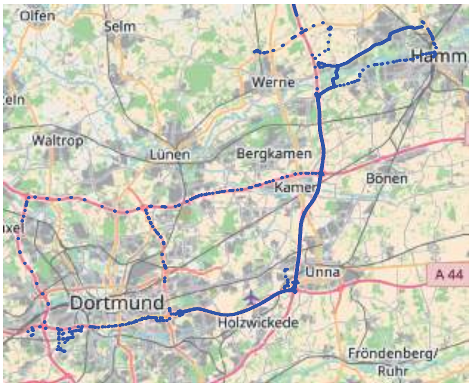


Fig. 9.16: Road map with locations of all data samples of the measurement campaign between two larger cities in Germany. Each blue point represents an intermediate status logging of all measured variables (cf. Table 9.3) during ongoing uplink transmissions. (Map: ©OpenStreetMap contributors, CC BY-SA). ©[2018] IEEE. Reprinted, with permission, from [207].

prediction process. Figure 9.16 shows a road map of the measurement points along a vehicular trajectory that covers urban, suburban, and rural environments. In total, 6172 have been acquired during the real world measurements. In focusing on the platform's sensing capabilities, we considered four different variants of feature sets. A summary of the feature sets and the implied impact factors of the contained features is given in Table 9.3.

For performing the actual prediction task, different regression models are considered:

Random Forest with 64 trees and a maximum depth of 32.

Deep Learning with three fully connected hidden layers, 64 neurons per layer, and Rectified Linear Unit (ReLU) activation function.

Ridge Regression with 12 model parameters (one per feature plus one bias term).

The results of the 10-fold cross validation are summarized in Figure 9.17. While the differences between the feature set variants are comparably small, larger differences between the machine learning models can be observed. The Random-Forest models thoroughly performed best with a mean average error of 3.166 dB. It can also be seen that the standard deviation between the different cross validation runs is small, which indicates a good model fit to unknown and independent data.

Tab. 9.3: Captured features and association with application-specific prediction models based on full-feature set $\mathbb{F}$, practical sets $\mathbb{P}1$/$\mathbb{P}2$, and simulation set $\mathbb{S}$. ©[2018] IEEE. Reprinted, with permission, from [207].

Parameter	**Model**	**Indicated Influences(s)**
Velocity	$\mathbb{F}$,$\mathbb{P}1$,$\mathbb{P}2$,$\mathbb{S}$	Distortions by fast fading
Upload size	$\mathbb{F}$,$\mathbb{P}1$,$\mathbb{P}2$,$\mathbb{S}$	Influence of TCP slow start
RSRP	$\mathbb{F}$,$\mathbb{P}1$,$\mathbb{P}2$,$\mathbb{S}$	Signal strength, distance
RSRQ, SINR	$\mathbb{F}$,$\mathbb{P}1$,$\mathbb{P}2$	Signal clarity, interference
Datarate	$\mathbb{F}$,$\mathbb{P}1$	Signal strength, allocated RBs *M*
RSSI	$\mathbb{F}$	Signal strength, distance
Frequency band	$\mathbb{F}$	Environment [349]
Number of neighbor cells (intra/inter freq.)	$\mathbb{F}$	Environment, cell density, interference
Cell bandwidth	$\mathbb{F}$	Exhaustion of TX-power headroom

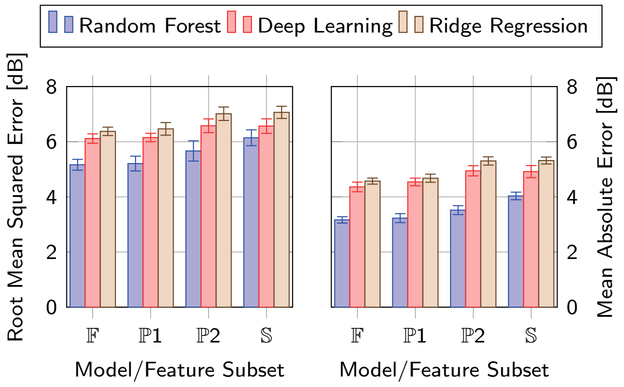


Fig. 9.17: Cross-validated error of trained prediction models for each feature subset ($\mathbb{F}$, $\mathbb{P}1$, $\mathbb{P}2$, $\mathbb{S}$) and each machine learning method (Random Forest, Deep Learning, Ridge Regression) in terms of Root Mean Squared Error (RMSE) (left) and Mean Absolute Error (MAE) (right). Lower is better. ©[2018] IEEE. Reprinted, with permission, from [207].

Bibliography

[1] D. J. Abadi et al. “Aurora: A New Model and Architecture for Data Stream Management”. In: *The VLDB Journal – The Int. Journal on Very Large Data Bases* 12.2 (Aug. 2003), pp. 120–139 (cit. on p. 21).

[2] M. R. Ackermann, J. Blömer, and C. Sohler. “Clustering for Metric and Nonmetric Distance Measures”. In: *ACM Transactions on Algorithms* 6.4 (2010). Previously appeared in the Procs. of the Symposium on Discrete Algorithms 2008, 59:1–59:26 (cit. on pp. 201, 203, 204).

[3] C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl, and D. Rousseau. “The Higgs boson machine learning challenge”. In: *Advances in Neural Information Processing Systems: Procs. of the Workshop on High-energy Physics and Machine Learning 2015*. 2015, pp. 19–55 (cit. on p. 250).

[4] A. Adiththan, S. Ramesh, and S. Samii. “Cloud-assisted control of ground vehicles using adaptive computation offloading techniques”. In: *Procs. of the Design, Automation and Test in Europe Conference 2018*. Mar. 2018, pp. 589–592 (cit. on p. 363).

[5] P. K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir. “Computing the Discrete Fréchet Distance in Subquadratic Time”. In: *SIAM Journal on Computing* 43.2 (2014). Previously appeared in the Procs. of the Symposium on Discrete Algorithms 2013, pp. 429–449 (cit. on p. 199).

[6] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. “Approximating extent measures of points”. In: *Journal of the Association for Computing Machinery* 51.4 (2004), pp. 606–635 (cit. on pp. 87, 88).

[7] P. K. Agarwal and R. Sharathkumar. “Streaming Algorithms for Extent Problems in High Dimensions”. In: *Algorithmica* 72.1 (2015), pp. 83–98 (cit. on p. 87).

[8] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy. “X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.12 (2018) (cit. on p. 4).

[9] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. “Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages”. In: *Procs. of the Int. Conference on World Wide Web 2013*. 2013, pp. 13–24 (cit. on p. 272).

[10] B. Ahmadi, K. Kersting, and S. Natarajan. “Lifted Online Training of Relational Models with Stochastic Gradient Methods”. In: *Procs. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2012*. Ed. by P. Flach, T. D. Bie, and N. Cristianini. Vol. 7523. Bristol, UK: Springer, Sept. 2012, pp. 585–600. URL: http://link.springer.com/chapter/10.1007/978-3-642-33460-3_43 (cit. on p. 407).

[11] R. Ahmed, B. Buchli, S. Draskovic, L. Sigrist, P. Kumar, and L. Thiele. “Optimal Power Management with Guaranteed Minimum Energy Utilization for Solar Energy Harvesting Systems”. In: *ACM Transactions on Embedded Computing Systems* 18.4 (2019), pp. 1–26 (cit. on p. 47).

[12] H.-K. Ahn, H. Alt, M. Buchin, E. Oh, L. Scharf, and C. Wenk. “Middle curves based on discrete Fréchet distance”. In: *Computational Geometry: Theory and Applications* 89 (2020). Previously appeared in LATIN 2016, p. 101621 (cit. on pp. 197, 201, 202, 204, 205).

[13] Y. Akhremtsev, P. Sanders, and C. Schulz. “High-Quality Shared-Memory Graph Partitioning”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.11 (2020), pp. 2710–2722 (cit. on p. 152).

[14] H. Alt and M. Godau. “Computing the Fréchet distance between two polygonal curves”. In: *Int. Journal of Computational Geometry and Applications* 5 (1995), pp. 75–91 (cit. on pp. 199, 211).

https://doi.org/10.1515/9783110785944-010

[15] F. Álvaro Muñoz, J. Sánchez Peiró, and J. Benedí Ruiz. “Recognition of On-line Handwritten Mathematical Expressions Using 2D Stochastic Context-Free Grammars and Hidden Markov Models”. In: *Pattern Recognition Letters* (2014), pp. 58–67 (cit. on p. 162).

[16] M. R. Anderberg. *Cluster analysis for applications*. Academic Press, 1973 (cit. on pp. 218, 222).

[17] H. Anderhub et al. “Design and operation of FACT - the first G-APD Cherenkov telescope”. In: *Journal of Instrumentation* 8.06 (June 2013), P06008. URL: http://iopscience.iop.org/1748-0221/8/06/P06008/ (cit. on pp. 351, 352).

[18] A. Andrae and T. Edler. “On Global Electricity Usage of Communication Technology: Trends to 2030”. In: *Challenges* 6.1 (2015), pp. 117–157 (cit. on p. 2).

[19] S. I. Ao et al. “CLUSTAG: hierarchical clustering and graph methods for selecting tag SNPs”. In: *Bioinformatics* 21.8 (2005), pp. 1735–1736 (cit. on p. 216).

[20] A. Arasu, S. Babu, and J. Widom. “The CQL Continuous Query Language: Semantic Foundations and Query Execution”. In: *The VLDB Journal – The Int. Journal on Very Large Data Bases* 15.2 (June 2006), pp. 121–142 (cit. on p. 21).

[21] A. Arasu et al. *STREAM: The Stanford Data Stream Management System*. Tech. rep. 2004-20. Stanford InfoLab, 2004. URL: http://ilpubs.stanford.edu:8090/641/ (cit. on p. 24).

[22] A. Arcangeli, I. Eidus, and C. Wright. “Increasing memory density by using KSM”. In: *Procs. of the Ottawa Linux Symposium 2009*. 2009, pp. 19–28 (cit. on p. 307).

[23] M. Arjovsky, S. Chintala, and L. Bottou. “Wasserstein Generative Adversarial Networks”. In: *Procs. of the Int. Conference on Machine Learning 2017*. 2017, pp. 214–223. URL: http://proceedings.mlr.press/v70/arjovsky17a.html (cit. on p. 260).

[24] ARM Limited. *Introducing NEON - Development Article*. URL: https://developer.arm.com/documentation/dht0002/a/ (cit. on p. 408).

[25] D. Arthur and S. Vassilvitskii. “k-means++: The advantages of careful seeding”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2007*. Association for Computing Machinery, 2007, pp. 1027–1035 (cit. on p. 188).

[26] N. Asadi, J. Lin, and A. P. De Vries. “Runtime optimizations for tree-based machine learning models”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.9 (2014), pp. 2281–2292 (cit. on p. 340).

[27] P. Axer and R. Ernst. “Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors”. In: *Procs. of the Design Automation Conference 2013*. 2013, 172:1–172:7 (cit. on p. 363).

[28] B. Weisfeiler and A. Leman. “The reduction of a graph to canonical form and the algebra which appears therein”. In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968). English translation by G. Ryabov is available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf, pp. 12–16 (cit. on pp. 117, 118).

[29] R. Babbar and B. Schölkopf. “Data scarcity, robustness and extreme multi-label classification”. In: *Machine Learning* 108.8 (2019), pp. 1329–1351 (cit. on pp. 275, 281, 284).

[30] R. Babbar and B. Schölkopf. “Dismec: Distributed sparse machines for extreme multi-label classification”. In: *Procs. of the ACM Int. Conference on Web Search and Data Mining 2017*. 2017, pp. 721–729 (cit. on p. 275).

[31] K. Bache and M. Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml (cit. on pp. 351, 352).

[32] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. “Streaming submodular maximization: Massive data summarization on the fly”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2014*. 2014 (cit. on pp. 76, 77).

[33] M. Badoiu and K. L. Clarkson. “Optimal core-sets for balls”. In: *Computational Geometry: Theory and Applications* 40.1 (May 2008), pp. 14–22 (cit. on p. 87).

[34] M. Badoiu and K. L. Clarkson. “Smaller core-sets for balls”. In: *Procs. of the Symposium on Discrete Algorithms 2003*. 2003, pp. 801–802 (cit. on p. 87).

[35] M. Badoiu, S. Har-Peled, and P. Indyk. “Approximate clustering via core-sets”. In: *Procs. of the ACM Symposium on Theory of Computing 2002*. 2002, pp. 250–257 (cit. on pp. 86, 87).

[36] K. R. Baker, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. “Preemptive Scheduling of a Single Machine to Minimize Maximum Cost Subject to Release Dates and Precedence Constraints”. In: *Operations Research* 31.2 (1983), pp. 381–386. DOI: https://doi.org/10.1287/opre.31.2.381 (cit. on p. 366).

[37] T. P. Baker. “Stack-based Scheduling of Realtime Processes”. In: *Real-Time Systems* 1 (1991), pp. 67–99 (cit. on p. 361).

[38] V. Balntas, E. Riba, D. Ponsa, and K. Mikolajczyk. “Learning local feature descriptors with triplets and shallow convolutional neural networks”. In: *Procs. of the British Machine Vision Conference 2016*. Ed. by R. C. Wilson, E. R. Hancock, and W. A. P. Smith. BMVA Press, 2016, pp. 119.1–119.11 (cit. on p. 168).

[39] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-hashimi, D. Brunelli, and L. Benini. “Hibernus : Sustaining Computation during Intermittent Supply for Energy-Harvesting Systems”. In: *IEEE Embedded Systems Letters* 7.1 (2015) (cit. on p. 53).

[40] A.-L. Barabasi and Z. N. Oltvai. “Network biology: Understanding the cell’s functional organization”. In: *Nature Reviews Genetics* 5.2 (2004), pp. 101–113 (cit. on pp. 116, 144).

[41] A. Barbalace, A. Iliopoulos, H. Rauchfuss, and G. Brasche. “It’s Time to Think About an Operating System for Near Data Processing Architectures”. In: *Procs. of the Workshop on Hot Topics in Operating Systems 2017*. HotOS ’17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 56–61. DOI: https://doi.org/10.1145/3102980.3102990 (cit. on p. 17).

[42] A. Barbalace, J. Picorel, and P. Bhatotia. “ExtOS: Data-Centric Extensible OS”. In: *Procs. of the ACM SIGOPS Asia-Pacific Workshop on Systems 2019*. APSys ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 31–39. DOI: https://doi.org/10.1145/3343737.3343742 (cit. on p. 17).

[43] R. C. Barros, A. C. P. L. F. de Carvalho, and A. A. Freitas. “Decision-Tree Induction”. In: *Automatic Design of Decision-Tree Induction Algorithms*. Springer International Publishing, 2015, pp. 7–45 (cit. on p. 340).

[44] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe. “Convexity, classification, and risk bounds”. In: *Journal of the American Statistical Association* 101.473 (2006), pp. 138–156 (cit. on p. 274).

[45] L. Becchetti, M. Bury, V. Cohen-Addad, F. Grandoni, and C. Schwiegelshohn. “Oblivious dimension reduction for *k*-means: beyond subspaces and the Johnson-Lindenstrauss lemma”. In: *Procs. of the ACM SIGACT Symposium on Theory of Computing 2019*. Ed. by M. Charikar and E. Cohen. ACM, 2019, pp. 1039–1050. DOI: 10.1145/3313276.3316318. URL: https://doi.org/10.1145/3313276.3316318 (cit. on p. 213).

[46] S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean. “Schedulability analysis of dependent probabilistic real-time tasks”. In: *Procs. of the Int. Conference on Real-Time Networks and Systems 2016*. 2016, pp. 99–107 (cit. on p. 363).

[47] J. L. Bentley and J. B. Saxe. “Decomposable Searching Problems I: Static-to-Dynamic Transformation”. In: *Journal of Algorithms* 1.4 (1980), pp. 301–358 (cit. on p. 87).

[48] E. Bernhardsson. *Annoy: Approximate Nearest Neighbors in C++/Python*. Python package version 1.13.0. 2018. URL: https://pypi.org/project/annoy/ (cit. on pp. 174, 175).

[49] N. Bertram, J. Ellert, and J. Fischer. “A Parallel Framework for Approximate Max-Dicut in Partitionable Graphs”. In: *Procs. of the Int. Symposium on Experimental Algorithms 2022*. Ed. by C. Schulz and B. Uçar. Vol. 233. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 10:1–10:15. DOI: 10.4230/LIPIcs.SEA.2022.10. URL: https://doi.org/10.4230/LIPIcs.SEA.2022.10 (cit. on p. 145).

[50] K. Bhatia, K. Dahiya, H. Jain, Y. Prabhu, and M. Varma. *The Extreme Classification Repository: Multi-label Datasets and Code*. 2016. URL: http://manikvarma.org/downloads/XC/XMLRepository.html (cit. on pp. 274, 282, 283).

[51] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. "Sparse Local Embeddings for Extreme Multi-Label Classification". In: *Procs. of the Int. Conference on Neural Information Processing Systems 2015*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 730–738 (cit. on p. 283).

[52] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. "Graph Neural Networks with Convolutional ARMA Filters". In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1901.01343 (cit. on p. 136).

[53] B. Bischl, J. Bossek, D. Horn, and M. Lang. *Model-Based Optimization for mlr*. 2014. URL: https://github.com/berndbischl/mlrMBO (cit. on pp. 289, 294).

[54] B. Bischl, S. Wessing, N. Bauer, K. Friedrichs, and C. Weihs. "MOI-MBO: Multiobjective Infill for Parallel Model-Based Optimization". In: *Procs. of the Learning and Intelligent Optimization Conference 2014*. 2014 (cit. on p. 287).

[55] B. Bischl et al. "Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges". In: *arXiv: Computing Research Repository* (2021). DOI: arXiv:2107.05847v2 (cit. on p. 286). **SFB876-A3**

[56] G. E. Blelloch. *Prefix Sums and Their Applications*. Tech. rep. Carnegie Mellon University, 1990 (cit. on p. 394).

[57] J. Blitzstein and P. Diaconis. "A sequential importance sampling algorithm for generating random graphs with prescribed degrees." In: *Internet Mathematics* 6.4 (2011), pp. 489–522 (cit. on p. 416).

[58] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. "A Flexible Real-Time Locking Protocol for Multiprocessors". In: *Procs. of the IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications 2007*. 2007 (cit. on pp. 361, 362, 368).

[59] S. Boettcher and A. G. Percus. "Extremal Optimization: Methods Derived from Co-Evolution". In: *Procs. of the Conference on Genetic and Evolutionary Computation 1999*. GECCO'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 825–832 (cit. on p. 145).

[60] J. Bolte, S. Sabach, and M. Teboulle. "Proximal alternating linearized minimization for nonconvex and nonsmooth problems". In: *Mathematical Programming* 146.1-2 (2014), pp. 459–494 (cit. on pp. 233, 234).

[61] P. Boncz, T. Neumann, and O. Erling. "TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark". In: *Procs. of the Technology Conference on Performance Evaluation and Benchmarking 2018*. Springer, 2018, pp. 61–76 (cit. on p. 400).

[62] H. Borchers. *adagio: Discrete and Global Optimization Routines*. R package version 0.6.5. 2016. URL: https://CRAN.R-project.org/package=adagio (cit. on p. 291).

[63] C. Borchert, D. Lohmann, and O. Spinczyk. "CiAO/IP: A Highly Configurable Aspect-Oriented IP Stack". In: *Procs. of the Int. Conference on Mobile Systems, Applications, and Services 2012*. New York, NY, USA: ACM, June 2012, pp. 435–448. DOI: http://doi.acm.org/10.1145/2307636.2307676 (cit. on p. 40). **SFB876-A1, SFB876-A4**

[64] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. "Protein function prediction via graph kernels." In: *Bioinformatics* 21 Suppl 1 (2005), pp. i47–i56 (cit. on p. 116).

[65] K. Borgwardt and H.-P. Kriegel. "Shortest-Path Kernels on Graphs". In: *Procs. of the IEEE Int. Conference on Data Mining 2005*. 2005, pp. 74–81 (cit. on p. 124).

[66] J. Bossek. *smoof: Single and Multi-Objective Optimization Test Functions*. R package version 1.4. 2016. URL: https://CRAN.R-project.org/package=smoof (cit. on p. 293).

[67] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein. “Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020 (cit. on p. 132).

[68] B. B. Brandenburg. “Multiprocessor Real-Time Locking Protocols: A Systematic Review”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1909.09600 (cit. on p. 362).

[69] B. B. Brandenburg and J. H. Anderson. “Optimality Results for Multiprocessor Real-Time Locking”. In: *Procs. of the IEEE Real-Time Systems Symposium 2010*. 2010, pp. 49–60. DOI: http://dx.doi.org/10.1109/RTSS.2010.17 (cit. on pp. 361, 362).

[70] V. Braverman, G. Frahling, H. Lang, C. Sohler, and L. F. Yang. “Clustering High Dimensional Dynamic Data Streams”. In: *Procs. of the Int. Conference on Machine Learning 2017*. Ed. by D. Precup and Y. W. Teh. PMLR, 2017, pp. 576–585. URL: http://proceedings.mlr.press/v70/braverman17a.html (cit. on pp. 10, 213). **SFB876-A2**

[71] L. Breiman. “Bagging Predictors”. In: *Machine Learning* 24.2 (Aug. 1996), pp. 123–140. DOI: https://doi.org/10.1023/A:1018054314350 (cit. on p. 341).

[72] L. Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32 (cit. on pp. 341, 351).

[73] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984 (cit. on p. 351).

[74] S. Breß, H. Funke, and J. Teubner. “Robust Query Processing in Co-Processor-Accelerated Databases”. In: *Procs. of the ACM SIGMOD Conference on Management of Data 2016*. San Francisco, CA, USA: ACM, June 2016 (cit. on pp. 381, 382, 386). **SFB876-C5**

[75] S. Breß, B. Köcher, H. Funke, S. Zeuch, T. Rabl, and V. Markl. “Generating Custom Code for Efficient Query Execution on Heterogeneous Processors”. In: *The VLDB Journal – The Int. Journal on Very Large Data Bases* 27.6 (Dec. 2018), pp. 797–822 (cit. on p. 394). **SFB876-A2**

[76] M. Brill, T. Fluschnik, V. Froese, B. J. Jain, R. Niedermeier, and D. Schultz. “Exact mean computation in dynamic time warping spaces”. In: *Data Mining and Knowledge Discovery* 33.1 (2019), pp. 252–291. DOI: https://doi.org/10.1007/s10618-018-0604-8 (cit. on p. 204).

[77] K. Bringmann. “Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails”. In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2014*. 2014, pp. 661–670 (cit. on p. 199).

[78] K. Bringmann, A. Driemel, A. Nusser, and I. Psarros. “Tight Bounds for Approximate Near Neighbor Searching for Time Series under the Fréchet Distance”. In: *Procs. of the Symposium on Discrete Algorithms 2022*. SIAM, 2022, pp. 517–550. DOI: 10.1137/1.9781611977073.25 (cit. on p. 203).

[79] K. Bringmann and W. Mulzer. “Approximability of the discrete Fréchet distance”. In: *Journal of Computational Geometry* 7.2 (2016). Previously appeared in Procs. of the Int. Symposium on Computational Geometry 2015, pp. 46–76. URL: http://jocg.org/index.php/jocg/article/view/261 (cit. on p. 200).

[80] E. Brochu, V. M. Cora, and N. De Freitas. “A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning”. In: *arXiv: Computing Research Repository* (2010). DOI: arXiv:1012.2599 (cit. on p. 95).

[81] L. D. Brown, T. T. Cai, and A. DasGupta. “Interval estimation for a binomial proportion”. In: *Statistical Science* (2001), pp. 101–117 (cit. on p. 79).

[82] G. von der Brüggen, J.-J. Chen, W.-H. Huang, and M. Yang. “Release Enforcement in Resource-Oriented Partitioned Scheduling for Multiprocessor Systems”. In: *Procs. of the Int. Conference on Real-Time Networks and Systems 2017*. 2017 (cit. on p. 368). **SFB876-B2**

[83] G. von der Brüggen, K.-H. Chen, W.-H. Huang, and J.-J. Chen. “Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments”. In: *Procs. of the IEEE Real-Time Systems Symposium 2016*. IEEE, 2016, pp. 303–314 (cit. on p. 372). **SFB876-A1**

[84] G. von der Brüggen, W.-H. Huang, and J.-J. Chen. “Hybrid self-suspension models in real-time embedded systems”. In: *Procs. of the IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications 2017*. IEEE, 2017, pp. 1–9. DOI: http://dx.doi.org/10.1109/RTCSA.2017.8046328 (cit. on p. 363). **SFB876-B2**

[85] G. von der Brüggen, N. Piatkowski, K.-H. Chen, J.-J. Chen, and K. Morik. “Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2018*. LIPIcs, 2018 (cit. on pp. 377, 378). **SFB876-A1, SFB876-B2**

[86] G. von der Brüggen, N. Ueter, J. Chen, and M. Freier. “Parametric utilization bounds for implicit-deadline periodic tasks in automotive systems”. In: *Procs. of the Int. Conference on Real-Time Networks and Systems 2017*. 2017, pp. 108–117 (cit. on p. 368).

[87] G. v. d. Brüggen, L. Schönberger, and J.-J. Chen. “Do Nothing, but Carefully: Fault Tolerance with Timing Guarantees for Multiprocessor Systems devoid of Online Adaptation”. In: *Procs. of the IEEE Pacific Rim Int. Symposium on Dependable Computing 2018*. Taipei, Taiwan, Dec. 2018. URL: https://ieeexplore.ieee.org/document/8639554 (cit. on p. 372).

[88] N. Bruno and S. Chaudhuri. “Physical design refinement: The ’merge-reduce’ approach”. In: *ACM Transactions on Database Systems* 32.4 (2007), p. 28 (cit. on p. 88).

[89] M. Bruynooghe. “Méthodes nouvelles en classification automatique de données taxinomiques nombreuses”. In: *Statistique et analyse des données* 2.3 (1977), pp. 24–42. URL: https://www.numdam.org/item/SAD_1977__2_3_24_0/ (cit. on pp. 218, 222).

[90] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. “A Tight Linear Time (1/2)-Approximation for Unconstrained Submodular Maximization”. In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2012*. 2012, pp. 649–658 (cit. on pp. 145, 153).

[91] N. Buchbinder, M. Feldman, and R. Schwartz. “Online submodular maximization with preemption”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2015*. Society for Industrial and Applied Mathematics. 2015, pp. 1202–1216 (cit. on p. 76).

[92] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. “Four Soviets Walk the Dog: Improved Bounds for Computing the Fréchet Distance”. In: *Discrete & Computational Geometry* 58.1 (2017). Previously appeared in Symposium on Discrete Algorithms 2014, pp. 180–216. DOI: https://doi.org/10.1007/s00454-017-9878-7 (cit. on p. 199).

[93] K. Buchin, A. Driemel, and M. Struijs. “On the Hardness of Computing an Average Curve”. In: *Procs. of the Scandinavian Symposium and Workshops on Algorithm Theory 2020*. Ed. by S. Albers. Vol. 162. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 19:1–19:19 (cit. on pp. 197, 202–205).

[94] K. Buchin, T. Ophelders, and B. Speckmann. “SETH Says: Weak Fréchet Distance is Faster, but only if it is Continuous and in One Dimension”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2019*. Ed. by T. M. Chan. SIAM, 2019, pp. 2887–2901. DOI: https://doi.org/10.1137/1.9781611975482.179 (cit. on p. 200).

[95] K. Buchin et al. “Approximating (k, ℓ)-center clustering for curves”. In: *Procs. of the Symposium on Discrete Algorithms 2019*. Ed. by T. M. Chan. ACM-SIAM, 2019, pp. 2922–2938. DOI: https://doi.org/10.1137/1.9781611975482.181 (cit. on pp. 197, 202–205, 210).

[96] K. Buchin et al. “Median Trajectories”. In: *Algorithmica* 66.3 (2013). Previously appeared in ESA 2010, pp. 595–614 (cit. on p. 202).

[97] M. Buchin, A. Driemel, and D. Rohde. “Approximating (k, ℓ)-Median Clustering for Polygonal Curves”. In: *Procs. of the Symposium on Discrete Algorithms 2021*. Ed. by D. Marx. ACM-

SIAM, 2021, pp. 2697–2717. DOI: https://doi.org/10.1137/1.9781611976465.160 (cit. on pp. 199, 203, 204).

[98] M. Buchin, N. Funk, and A. Krivošija. “On the complexity of the middle curve problem”. In: *arXiv: Computing Research Repository* (2020). Presented in EuroCG 2020. DOI: arXiv:2001.10298 (cit. on pp. 197, 205). **SFB876-A2**

[99] M. Buchin, A. Krivošija, and A. Neuhaus. “Computing the Fréchet distance of trees and graphs of bounded tree width”. In: *arXiv: Computing Research Repository* (2020). Presented in EuroCG 2020. DOI: arXiv:2001.10502 (cit. on p. 212). **SFB876-A2**

[100] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. “Recent Advances in Graph Partitioning”. In: *Algorithm Engineering: Selected Results and Surveys*. Ed. by P. Kliemann Lasseand Sanders. Cham: Springer Int. Publishing, 2016, pp. 117–158 (cit. on p. 145).

[101] A. Burns and A. J. Wellings. “A Schedulability Compatible Multiprocessor Resource Sharing Protocol - MrsP”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2013*. 2013, pp. 282–291. DOI: http://dx.doi.org/10.1109/ECRTS.2013.37 (cit. on pp. 361, 362).

[102] M. Bury and C. Schwiegelshohn. “On Finding the Jaccard Center”. In: *Procs. of the Int. Colloquium on Automata, Languages and Programming 2017*. Ed. by P. Indyk, F. Kuhn, and A. Muscholl. 2017 (cit. on p. 212). **SFB876-A2**

[103] M. Bury, C. Schwiegelshohn, and M. Sorella. “Sketch ’Em All: Approximate Similarity Search for Dynamic Data Streams”. In: *Procs. of the ACM Int. Conference on Web Search and Data Mining 2018*. Ed. by Y. Maarek and Y. Liu. ACM, 2018 (cit. on p. 10). **SFB876-A2**

[104] M. Buschhoff. “Energy-Aware Design of Hardware and Software for Ultra-Low-Power Systems”. PhD thesis. Dortmund: TU Dortmund University, 2019. DOI: http://dx.doi.org/10.17877/DE290R-20241 (cit. on pp. 35, 42, 43).

[105] M. Buschhoff, R. Falkenberg, and O. Spinczyk. “Energy-Aware Device Drivers for Embedded Operating Systems”. In: *SIGBED Review* 16.3 (Nov. 2019), pp. 8–13. DOI: https://doi.org/10.1145/3373400.3373401 (cit. on pp. 6, 41). **SFB876-A4**

[106] M. Buschhoff, D. Friesel, and O. Spinczyk. “Energy Models in the Loop”. In: *Procedia Computer Science* 130 (2018), pp. 1063–1068 (cit. on pp. 6, 43, 44). **SFB876-A4**

[107] S. Buschjäger. “Ensemble Learning with Discrete Classifiers on Small Devices”. PhD thesis. TU Dortmund University, 2022 (cit. on p. 339).

[108] S. Buschjäger, K.-h. Chen, J.-j. Chen, and K. Morik. “Realization of Random Forest for Real-Time Evaluation through Tree Framing”. In: 2018. URL: https://ieeexplore.ieee.org/document/8594826 (cit. on pp. 339, 341). **SFB876-A1, SFB876-B2, SFB876-C3**

[109] S. Buschjäger, P.-J. Honysz, L. Pfahler, and K. Morik. “Very Fast Streaming Submodular Function Maximization”. In: *Procs. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2021*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 151–166. DOI: https://doi.org/10.1007/978-3-030-86523-8_10. URL: https://2021.ecmlpkdd.org/wp-content/uploads/2021/07/sub_178.pdf (cit. on pp. 74–76, 80, 81). **SFB876-A1**

[110] S. Buschjäger and K. Morik. “Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65-I.1 (Jan. 2018), pp. 209–222. DOI: https://doi.org/10.1109/TCSI.2017.2710627 (cit. on pp. 7, 341, 342).

[111] S. Buschjäger, K. Morik, and M. Schmidt. “Summary Extraction on Data Streams in Embedded Systems”. In: *Procs. of the ECML Workshop on IoT Large Scale Learning From Data Streams 2017*. 2017. URL: http://ceur-ws.org/Vol-1958/IOTSTREAMING3.pdf (cit. on p. 81). **SFB876-A1**

[112] S. Buschjäger, L. Pfahler, J. Buss, K. Morik, and W. Rhode. “On-Site Gamma-Hadron Separation with Deep Learning on FPGAs”. In: *Procs. of the Joint European Conference on Ma-*

chine Learning and Knowledge Discovery in Databases 2020. Springer, 2020. URL: https://link.springer.com/content/pdf/10.1007%5C%2F978-3-030-67667-4_29.pdf (cit. on p. 10). **SFB876-A1, SFB876-C3**

[113] S. Buschjäger et al. "Margin-Maximization in Binarized Neural Networks for Optimizing Bit Error Tolerance". In: *Procs. of the Design, Automation and Test in Europe Conference 2021*. 2021. URL: https://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2021dateyayla.pdf (cit. on pp. 10, 326, 331, 332). **SFB876-A1**

[114] S. Buschjäger et al. *Towards Explainable Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks*. 2020. URL: https : / / www - ai . cs . tu - dortmund . de / PublicPublicationFiles/buschjaeger_etal_2020b.pdf (cit. on p. 331). **SFB876-A1**

[115] J. Buss, C. Bockermann, K. Morik, W. Rhode, and T. Ruhe. "FACT-Tools – Processing High-Volume Telescope Data". In: *Procs. of the Astronomical Data Analysis Software and Systems Conference 2019*. Ed. by M. Molinaro, K. Shortridge, and F. Pasian. Vol. 521. Astronomical Society of the Pacific, 2019, p. 584. URL: http://www.aspbooks.org/a/volumes/article_details/?paper_id=39082 (cit. on p. 339). **SFB876-C3**

[116] J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. "An Improved Approximation for *k*-Median and Positive Correlation in Budgeted Optimization". In: *ACM Transactions on Algorithms* 13.2 (2017), 23:1–23:31. DOI: https://doi.org/10.1145/2981561 (cit. on p. 201).

[117] J. Cai, M. Fürer, and N. Immerman. "An optimal lower bound on the number of variables for graph identifications". In: *Combinatorica* 12.4 (1992), pp. 389–410 (cit. on p. 120).

[118] C. Cangea, P. Veličković, N. Jovanović, T. N. Kipf, and P. Liò. "Towards Sparse Hierarchical Graph Classifiers". In: *Advances in Neural Information Processing Systems 31: Procs. of the 2018 Conference*. 2018 (cit. on p. 135).

[119] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal. "Dynamic instrumentation of production systems". In: *Procs. of the Conference on USENIX Annual Technical Conference 2004*. Berkeley, CA, USA: USENIX Association, 2004, pp. 2–2 (cit. on pp. 17, 18).

[120] R. Caruana, N. Karampatziakis, and A. Yessenalina. "An empirical evaluation of supervised learning in high dimensions". In: *Procs. of the Int. Conference on Machine Learning 2008*. ACM. 2008, pp. 96–103 (cit. on p. 339).

[121] A. Chakrabarti and S. Kale. "Submodular Maximization Meets Streaming: Matchings, Matroids, and More". In: *arXiv: Computing Research Repository* (2013). DOI: arXiv:1309.2038 (cit. on p. 76).

[122] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy. "Machine Learning on Graphs: A Model and Comprehensive Taxonomy". In: *arXiv: Computing Research Repository* (2020). DOI: arXiv:2005.03675 (cit. on p. 116).

[123] M. Charikar, K. Chen, and M. Farach-Colton. "Finding frequent items in data streams". In: *Theoretical Computer Science* 312.1 (2004), pp. 3–15 (cit. on p. 88).

[124] C. Chekuri, S. Gupta, and K. Quanrud. "Streaming algorithms for submodular function maximization". In: *arXiv: Computing Research Repository* (2015). DOI: arXiv:1504.08024 (cit. on p. 76).

[125] F. Chen, J. Deng, Z. Pang, M. Baghaei Nejad, H. Yang, and G. Yang. "Finger angle-based hand gesture recognition for smart infrastructure using wearable wrist-worn camera". In: *Applied Sciences* 8.3 (2018), p. 369 (cit. on p. 61).

[126] J. Chen, J. Zhu, and L. Song. "Stochastic Training of Graph Convolutional Networks with Variance Reduction". In: *Procs. of the Int. Conference on Machine Learning 2018*. 2018 (cit. on pp. 136, 137, 139).

[127] J.-J. Chen, G. von der Brüggen, W.-H. Huang, and C. Liu. "State of the art for scheduling and analyzing self-suspending sporadic real-time tasks". In: *Procs. of the IEEE Int. Conference on*

Embedded and Real-Time Computing Systems and Applications 2017. (invited paper). IEEE, 2017, pp. 1–10. URL: 10.1109/RTCSA.2017.8046321 (cit. on p. 363). **SFB876-B2**

[128] J.-J. Chen et al. “Many suspensions, many problems: a review of self-suspending tasks in real-time systems”. In: *Real-Time Systems* (2018). preprint. URL: https://link.springer.com/article/10.1007%5C%2Fs11241-018-9316-9 (cit. on p. 363). **SFB876-B2**

[129] K. Chen, N. Ueter, G. v. der Brüggen, and J. Chen. “Efficient Computation of Deadline-Miss Probability and Potential Pitfalls”. In: *Procs. of the Design, Automation and Test in Europe Conference 2019*. 2019, pp. 896–901 (cit. on p. 377).

[130] K. Chen. “On Coresets for k-Median and k-Means Clustering in Metric and Euclidean Spaces and Their Applications”. In: *SIAM Journal on Computing* 39.3 (Aug. 2009). Previously appeared in Symposium on Discrete Algorithms 2006, pp. 923–947. URL: http://link.aip.org/link/?SMJ/39/923/1 (cit. on p. 201).

[131] K.-H. Chen and J.-J. Chen. “Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors”. In: *Procs. of the IEEE Int. Symposium on Industrial Embedded Systems 2017*. 2017, pp. 1–8. DOI: https://doi.org/10.1109/SIES.2017.7993392 (cit. on p. 377). **SFB876-B2**

[132] K.-H. Chen et al. “Efficient Realization of Decision Trees for Real-Time Inference (to appear, accepted)”. In: *ACM Transactions on Embedded Computing Systems* (2022) (cit. on p. 349). **SFB876-A1**

[133] L. Chen, Z. Wei, Z. Cui, M. Chen, H. Pan, and Y. Bao. “CMD: Classification-based Memory Deduplication Through Page Access Characteristics”. In: *Procs. of the ACM SIGPLAN/SIGOPS Int. Conference on Virtual Execution Environments 2014*. VEE ’14. ACM, 2014, pp. 65–76 (cit. on p. 307).

[134] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. “Simple and Deep Graph Convolutional Networks”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020 (cit. on pp. 130, 132, 175).

[135] Y. Cheng and G. M. Church. “Biclustering of expression data.” In: *Procs. of the Int. Conference on Intelligent Systems for Molecular Biology 2000*. Vol. 8. 2000, pp. 93–103 (cit. on p. 233).

[136] C. Chevalier and D. Ginsbourger. “Fast Computation of the Multi-Points Expected Improvement with Applications in Batch Selection”. In: *Learning and Intelligent Optimization*. Ed. by G. Nicosia and P. Pardalos. Springer, 2013, pp. 59–69 (cit. on p. 287).

[137] W. L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. J. Hsieh. “Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2019*. SIGKDD. 2019 (cit. on pp. 136, 141).

[138] Y.-D. Chih et al. “An 89TOPS/W and 16.3 TOPS/mm 2 All-Digital SRAM-Based Full-Precision Compute-In Memory Macro in 22nm for Machine-Learning Edge Applications”. In: *Procs. of the IEEE Int. Solid-State Circuits Conference 2021*. Vol. 64. IEEE. 2021, pp. 252–254 (cit. on p. 4).

[139] H. Cho, I. S. Dhillon, Y. Guan, and S. Sra. “Minimum sum-squared residue co-clustering of gene expression data”. In: *Procs. of the SIAM Int. Conference on Data Mining 2004*. SIAM. 2004, pp. 114–125 (cit. on p. 232).

[140] C. Choirat and R. Seri. “Estimation in Discrete Parameter Models”. In: *Statistical Science* 27.2 (2012), pp. 278–293 (cit. on p. 407).

[141] K. L. Clarkson. “Subgradient and sampling algorithms for ℓ_1 regression”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2005*. 2005, pp. 257–266 (cit. on pp. 87, 90).

[142] K. L. Clarkson, P. Drineas, M. Magdon-Ismail, M. W. Mahoney, X. Meng, and D. P. Woodruff. “The Fast Cauchy Transform and Faster Robust Linear Regression”. In: *SIAM Journal on*

Computing 45.3 (2016), pp. 763–810. DOI: https://doi.org/10.1137/140963698 (cit. on pp. 87, 90).

[143] K. L. Clarkson, R. Wang, and D. P. Woodruff. "Dimensionality Reduction for Tukey Regression". In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019, pp. 1262–1271 (cit. on p. 90).

[144] K. L. Clarkson and D. P. Woodruff. "Input Sparsity and Hardness for Robust Subspace Approximation". In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2015*. 2015, pp. 310–329. DOI: https://doi.org/10.1109/FOCS.2015.27 (cit. on pp. 87, 90).

[145] K. L. Clarkson and D. P. Woodruff. "Numerical linear algebra in the streaming model". In: *Procs. of the ACM Symposium on Theory of Computing 2009*. 2009, pp. 205–214 (cit. on pp. 89, 93, 94).

[146] K. L. Clarkson and D. P. Woodruff. "Sketching for *M*-Estimators: A Unified Approach to Robust Regression". In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2015*. 2015, pp. 921–939. DOI: https://doi.org/10.1137/1.9781611973730.63 (cit. on pp. 87, 90).

[147] M. B. Cohen, Y. T. Lee, C. Musco, C. Musco, R. Peng, and A. Sidford. "Uniform Sampling for Matrix Approximation". In: *Procs. of the Conference on Innovations in Theoretical Computer Science 2015*. 2015, pp. 181–190 (cit. on p. 87).

[148] V. Cohen-Addad, K. G. Larsen, D. Saulpic, and C. Schwiegelshohn. "Towards optimal lower bounds for *k*-median and *k*-means coresets". In: *Procs. of the ACM SIGACT Symposium on Theory of Computing 2022*. Ed. by S. Leonardi and A. Gupta. ACM, 2022, pp. 1038–1051. DOI: 10.1145/3519935.3519946. URL: https://doi.org/10.1145/3519935.3519946 (cit. on p. 212).

[149] V. Cohen-Addad, D. Saulpic, and C. Schwiegelshohn. "A new coreset framework for clustering". In: *Procs. of the Symposium on Theory of Computing 2021*. Ed. by S. Khuller and V. Vassilevska Williams. ACM, 2021, pp. 169–182. DOI: https://doi.org/10.1145/3406325.3451022 (cit. on p. 212).

[150] V. Cohen-Addad and C. Schwiegelshohn. "On the Local Structure of Stable Clustering Instances". In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2017*. Ed. by C. Umans. 2017 (cit. on p. 213). **SFB876-A2**

[151] V. Cohen-Addad, C. Schwiegelshohn, and C. Sohler. "Diameter and k-Center in Sliding Windows". In: *Procs. of the Int. Colloquium on Automata, Languages, and Programming 2016*. Ed. by M. Mitzenmacher, Y. Rabani, and D. Sangiorgi. Vol. 55. 2016, 19:1–19:12. DOI: https://doi.org/10.4230/LIPIcs.ICALP.2016.19 (cit. on p. 213). **SFB876-A2**

[152] A. Colin, E. Ruppel, and B. Lucia. "A reconfigurable energy storage architecture for energy-harvesting devices". In: *Procs. of the Int. Conference on Architectural Support for Programming Languages and Operating Systems 2018*. ACM, 2018 (cit. on pp. 53, 57).

[153] W. Cong, R. Forsati, M. Kandemir, and M. Mahdavi. "Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks". In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2020*. 2020 (cit. on p. 137).

[154] G. Cormode and S. Muthukrishnan. "An improved data stream summary: The Count-Min sketch and its applications". In: *Journal of Algorithms* 55 (2004), pp. 29–38 (cit. on p. 88).

[155] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković. "Principal Neighbourhood Aggregation for Graph Nets". In: *Advances in Neural Information Processing Systems 33: Procs. of the 2020 Conference*. 2020 (cit. on pp. 132, 133, 139).

[156] M. Cucuringu, J. Puente, and D. Shue. *Model Selection in Undirected Graphical Models with the Elastic Net*. 2011 (cit. on p. 105).

[157] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. "Calibrating probability with undersampling for unbalanced classification". In: *Procs. of the IEEE Symposium Series on Computational Intelligence 2015*. IEEE. 2015, pp. 159–166. URL: https://www.kaggle.com/mlg-ulb/creditcardfraud (cit. on p. 81).

[158] E. Dallago, A. Barnabei, A. Liberale, P. Malcovati, and G. Venchi. “An Interface Circuit for Low-Voltage Low-Current Energy Harvesting Systems”. In: *IEEE Transactions on Power Electronics* 30.3 (2015) (cit. on p. 53).

[159] E. Dallago, A. Lazzarini Barnabei, A. Liberale, G. Torelli, and G. Venchi. “A 300 mV Low-Power Management System For Energy Harvesting Applications”. In: *IEEE Transactions on Power Electronics* PP.99 (2015) (cit. on p. 53).

[160] J. N. Darroch and D. Ratcliff. “Generalized iterative scaling for log-linear models”. In: *The Annals of Mathematical Statistics* 43.5 (1972), pp. 1470–1480 (cit. on p. 105).

[161] A. Dasgupta, P. Drineas, B. Harb, and R. K. and Michael W. Mahoney. “Sampling algorithms and coresets for ℓ_p-regression”. In: *SIAM Journal on Computing* 38.5 (2009), pp. 2060–2078 (cit. on pp. 87, 90).

[162] D. Dato et al. “Fast ranking with additive ensembles of oblivious and non-oblivious regression trees”. In: *ACM Transactions on Information Systems* (2016) (cit. on p. 340).

[163] D. Davis, B. Edmunds, and M. Udell. “The Sound of APALM Clapping: Faster Nonsmooth Nonconvex Optimization with Stochastic Asynchronous PALM”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. Vol. 29. 2016 (cit. on p. 247).

[164] M. Defferrard, X. Bresson, and P. Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. 2016, pp. 3844–3852 (cit. on p. 136).

[165] N. Del Buono and G. Pio. “Non-negative Matrix Tri-Factorization for co-clustering: An analysis of the block matrix”. In: *Information Sciences* 301 (2015), pp. 13–26 (cit. on pp. 232, 238, 239).

[166] C. Delimitrou and C. Kozyrakis. “Quasar: Resource-efficient and QoS-aware Cluster Management”. In: *Procs. of the Int. Conference on Architectural Support for Programming Languages and Operating Systems 2014*. ACM, 2014, pp. 127–144 (cit. on p. 285).

[167] H. Dell, M. Grohe, and G. Rattan. “Lovász Meets Weisfeiler and Leman”. In: *Procs. of the Int. Colloquium on Automata, Languages, and Programming 2018*. 2018, 40:1–40:14 (cit. on p. 125).

[168] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. “What does classifying more than 10,000 image categories tell us?” In: *Procs. of the European Conference on Computer Vision 2010*. Springer, 2010, pp. 71–84 (cit. on p. 272).

[169] Y. Deng, L. Song, and X. Huang. “Evaluating Memory Compression and Deduplication”. In: *Procs. of the IEEE Int. Conference on Networking, Architecture and Storage 2013*. IEEE Computer Society, 2013, pp. 282–286 (cit. on p. 307).

[170] Y. Deng, A. Kanervisto, J. Ling, and A. M. Rush. “Image-to-Markup Generation with Coarse-to-Fine Attention”. In: *Procs. of the Int. Conference on Machine Learning 2017*. 2017, pp. 980–989 (cit. on p. 162).

[171] E. Denton, J. Weston, M. Paluri, L. Bourdev, and R. Fergus. “User Conditional Hashtag Prediction for Images”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2015*. 2015 (cit. on p. 272).

[172] M. Desnoyers and M. R. Dagenais. “The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux”. In: *Procs. of the Ottawa Linux Symposium 2006*. 2006, pp. 209–224 (cit. on p. 17).

[173] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Procs. of Conference of the North American Chapter of the Association for Computational Linguistics 2019*. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: arXiv:1810.04805 (cit. on pp. 163, 166, 167, 169).

[174] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv: Computing Research Repository* (2018). DOI: arXiv:1810.04805 (cit. on p. 283).

[175] I. S. Dhillon, Y. Guan, and B. Kulis. “Weighted Graph Cuts without Eigenvectors: A Multilevel Approach”. In: 2007, pp. 1944–1957 (cit. on pp. 135, 140).

[176] I. S. Dhillon. “Co-clustering documents and words using bipartite spectral graph partitioning”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2001*. ACM. 2001, pp. 269–274 (cit. on p. 232).

[177] J. L. Díaz et al. “Stochastic Analysis of Periodic Real-Time Systems”. In: *Procs. of the IEEE Real-Time Systems Symposium 2002*. 2002, pp. 289–300 (cit. on p. 363).

[178] C. Ding, T. Li, W. Peng, and H. Park. “Orthogonal nonnegative matrix t-factorizations for clustering”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2006*. ACM. 2006, pp. 126–135 (cit. on p. 232).

[179] S. Ding, L. Lin, G. Wang, and H. Chao. “Deep feature learning with relative distance comparison for person re-identification”. In: *Pattern Recognition* 48.10 (2015), pp. 2993–3003 (cit. on p. 166).

[180] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda. “Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed”. In: *Testbeds and Research Infrastructure. Development of Networks and Communities*. Ed. by T. Korakis, H. Li, P. Tran-Gia, and H.-S. Park. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2012, pp. 302–316 (cit. on p. 35).

[181] W. Dong, L. Yang, R. Gravina, and G. Fortino. “Soft wrist-worn multi-functional sensor array for real-time hand gesture recognition”. In: *IEEE Sensors Journal* (2021) (cit. on p. 61).

[182] B. Douillard, D. Fox, and F. T. Ramos. “A spatio-temporal probabilistic model for multi-sensor object recognition”. In: *Procs. of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems 2007*. 2007, pp. 2402–2408 (cit. on p. 102).

[183] U. Drepper. *What Every Programmer Should Know About Memory*. 2007 (cit. on p. 343).

[184] A. Driemel and A. Krivošija. “Probabilistic embeddings of the Fréchet distance”. In: *Procs. of the Int. Workshop on Approximation and Online Algorithms 2018*. Ed. by L. Epstein and T. Erlebach. Vol. 11312. LNCS. Springer, 2018, pp. 218–237. DOI: https://doi.org/10.1007/978-3-030-04693-4_14 (cit. on p. 212). **SFB876-A2**

[185] A. Driemel, A. Krivošija, and C. Sohler. “Clustering time series under the Fréchet distance”. In: *Procs. of the Symposium on Discrete Algorithms 2016*. Ed. by R. Krauthgamer. SIAM, 2016, pp. 766–785. URL: http://epubs.siam.org/doi/10.1137/1.9781611974331.ch55 (cit. on pp. 197, 201–204). **SFB876-A2**

[186] A. Driemel and I. Psarros. “ANN for Time Series Under the Fréchet Distance”. In: *Procs. of the Int. Symposium on Algorithms and Data Structures 2021*. Ed. by A. Lubiw and M. R. Salavatipour. Vol. 12808. Lecture Notes in Computer Science. Springer, 2021, pp. 315–328. DOI: https://doi.org/10.1007/978-3-030-83508-8_23 (cit. on p. 203).

[187] D. Driggs, J. Tang, M. Davies, and C.-B. Schönlieb. “SPRING: A fast stochastic proximal alternating method for non-smooth non-convex optimization”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv:2002.12266 (cit. on pp. 238, 247).

[188] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. “Relative-Error CUR Matrix Decompositions”. In: *SIAM Journal on Matrix Analysis and Applications* 30.2 (2008), pp. 844–881. DOI: https://doi.org/10.1137/07070471X (cit. on pp. 87, 90, 92).

[189] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. “Sampling algorithms for ℓ_2 regression and applications”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2006*. 2006, pp. 1127–1136 (cit. on pp. 87, 90).

[190] B. Dusza. “Context-Aware Power Consumption Modeling for Energy Efficient Mobile Communication Services”. PhD thesis. TU Dortmund University, 2014. URL: http://www.shaker.de/de/content/catalogue/index.asp?lang=de%5C&ID=8%5C&ISBN=978-3-8440-2683-2 (cit. on p. 431). **SFB876-A4**

[191] B. Dusza, C. Ide, L. Cheng, and C. Wietfeld. “An Accurate Measurement-Based Power Consumption Model for LTE Uplink Transmissions”. In: *Procs. of IEEE INFOCOM 2013*. Turin, Italy: IEEE, Apr. 2013 (cit. on p. 430). **SFB876-A4, SFB876-B4**

[192] B. Dusza, C. Ide, L. Cheng, and C. Wietfeld. “CoPoMo: A Context-Aware Power Consumption Model for LTE User Equipment”. In: *Transactions on Emerging Telecommunications Technologies* 24.6 (Oct. 2013), pp. 615–632. URL: http://onlinelibrary.wiley.com/doi/10.1002/ett.2702/full (cit. on pp. 429, 430, 432). **SFB876-A4, SFB876-B4**

[193] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About A Highly Connected World*. Cambridge University Press, July 2010 (cit. on pp. 116, 144).

[194] A. Easwaran and B. Andersson. “Resource Sharing in Global Fixed-Priority Preemptive Multiprocessor Scheduling”. In: *Procs. of the IEEE Real-Time Systems Symposium 2009*. 2009, pp. 377–386. DOI: http://dx.doi.org/10.1109/RTSS.2009.37 (cit. on p. 368).

[195] I. J. Egielski, J. Huang, and E. Z. Zhang. “Massive Atomics for Massive Parallelism on GPUs”. In: *ACM SIGPLAN Notices* 49.11 (2015), pp. 93–103 (cit. on p. 391).

[196] F. C. Eigler et al. *Architecture of systemtap: a Linux trace/probe tool*. 2005. URL: http://www.cs.ucsb.edu/~grze/papers/profile/eigler05systemtap.bib (cit. on pp. 16, 18).

[197] T. Eiter and H. Mannila. *Computing discrete Fréchet distance*. Tech. rep. CD-TR 94/64. Christian Doppler Laboratory, 1994 (cit. on pp. 199, 211).

[198] J. G. Eldredge and B. L. Hutchings. “Density enhancement of a neural network using FPGAs and run-time reconfiguration”. In: *Procs. of the IEEE Workshop on FPGAs for Custom Computing Machines 1994*. IEEE. 1994, pp. 180–188 (cit. on pp. 253, 255).

[199] P. Emberson, R. Stafford, and R. I. Davis. “Techniques for the synthesis of multiprocessor tasksets”. In: *Procs. of the Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems 2010*. 2010, pp. 6–11 (cit. on p. 368).

[200] T. A. Ensslin, M. Frommert, and F. S. Kitaura. “Information field theory for cosmological perturbation reconstruction and nonlinear signal analysis”. In: *Physical Review D* 80 (2009). DOI: http://dx.doi.org/10.1103/PhysRevD.80.105005 (cit. on p. 160).

[201] D. Eriksson, M. Pearce, R. Gardner Jacob R. and Turner, and M. Poloczek. “Scalable Global Optimization via Local Bayesian Optimization”. In: *Procs. of the Conference on Neural Information Processing Systems 2019*. 2019, pp. 5497–5508 (cit. on p. 96).

[202] Ú. Erlingsson, M. Peinado, S. Peter, M. Budiu, and G. Mainar-Ruiz. “Fay: Extensible Distributed Tracing from Kernels to Clusters”. In: *ACM Transactions on Computer Systems* 30.4 (Nov. 2012), 13:1–13:35 (cit. on p. 18).

[203] H. Esen, M. Adachi, D. Bernardini, A. Bemporad, D. Rost, and J. Knodel. “Control as a Service (CaaS): Cloud-Based Software Architecture for Automotive Control Applications”. In: *Procs. of the Int. Workshop on the Swarm at the Edge of the Cloud 2015*. Swarm at the Edge of the Cloud 2015. New York, NY, USA: Association for Computing Machinery, 2015, pp. 13–18. DOI: https://doi.org/10.1145/2756755.2756758 (cit. on p. 363).

[204] A. Esper, G. Nelissen, V. Nélis, and E. Tovar. “How Realistic is the Mixed-Criticality Real-Time System Model?” In: *Procs. of the Int. Conference on Real Time and Networks Systems 2015*. Real Time and Networks Systems 2015. New York, NY, USA: Association for Computing Machinery, 2015, pp. 139–148. DOI: https://doi.org/10.1145/2834848.2834869 (cit. on p. 372).

[205] B. O. Fagginger Auer and R. H. Bisseling. “A GPU Algorithm for Greedy Graph Matching”. In: *Facing the Multicore - Challenge II - Aspects of New Paradigms and Technologies in Parallel Computing*. 2011 (cit. on p. 135).

[206] R. Falkenberg, J. Drenhaus, B. Sliwa, and C. Wietfeld. “System-in-the-loop Design Space Exploration for Efficient Communication in Large-scale IoT-based Warehouse Systems”. In: *Procs. of the IEEE Int. Systems Conference 2018*. Vancouver, Canada: IEEE, Apr. 2018 (cit. on pp. 424, 426, 427, 429, 430). **SFB876-A4**

[207] R. Falkenberg, B. Sliwa, N. Piatkowski, and C. Wietfeld. “Machine Learning Based Uplink Transmission Power Prediction for LTE and Upcoming 5G Networks using Passive Downlink Indicators”. In: *Procs. of the IEEE Vehicular Technology Conference 2018*. Chicago, USA, Aug. 2018 (cit. on pp. 433–435). **SFB876-A4, SFB876-B4, SFB876-A1**

[208] R. Falkenberg, B. Sliwa, and C. Wietfeld. “Rushing Full Speed with LTE-Advanced is Economical - A Power Consumption Analysis”. In: *Procs. of the IEEE Vehicular Technology Conference 2017*. June 2017, pp. 1–7 (cit. on p. 431). **SFB876-A4**

[209] R. Falkenberg et al. “PhyNetLab: An IoT-based warehouse testbed”. In: *Procs. of the Federated Conference on Computer Science and Information Systems 2017*. Sept. 2017 (cit. on pp. 35, 424, 426, 428). **SFB876-A4**

[210] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. “LIBLINEAR: A library for large linear classification”. In: *Journal of Machine Learning Research* 9 (2008), pp. 1871–1874 (cit. on p. 275).

[211] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. J. V. Gool. “Random Forests for Real Time 3D Face Analysis”. In: *Int. Journal of Computer Vision* 101.3 (2013), pp. 437–458 (cit. on p. 339).

[212] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. “CNP: An FPGA-based processor for convolutional networks”. In: *Procs. of the Int. Conference on Field Programmable Logic and Applications 2009*. IEEE. 2009, pp. 32–37 (cit. on p. 254).

[213] T. Feder and D. Greene. “Optimal algorithms for approximate clustering”. In: *Procs. of the Symposium on Theory of Computing 1988*. ACM, 1988, pp. 434–444 (cit. on p. 200).

[214] U. Feige. “A Threshold of Ln N for Approximating Set Cover”. In: *Journal of the Association for Computing Machinery* 45.4 (July 1998), pp. 634–652. DOI: http://doi.acm.org/10.1145/285055.285059 (cit. on p. 75).

[215] U. Feige, V. S. Mirrokni, and J. Vondrák. “Maximizing non-monotone submodular functions”. In: *SIAM Journal on Computing* 40.4 (2011), pp. 1133–1153 (cit. on p. 76).

[216] D. Feldman, M. Faulkner, and A. Krause. “Scalable Training of Mixture Models via Coresets”. In: *Advances in Neural Information Processing Systems 24: Procs. of the 2011 Conference*. 2011, pp. 2142–2150. URL: http://papers.nips.cc/paper/4363-scalable-training-of-mixture-models-via-coresets (cit. on p. 87).

[217] D. Feldman and M. Langberg. “A unified framework for approximating and clustering data”. In: *Procs. of the ACM Symposium on Theory of Computing 2011*. Ed. by L. Fortnow and S. P. Vadhan. ACM, 2011, pp. 569–578. DOI: http://doi.acm.org/10.1145/1993636.1993712 (cit. on p. 212).

[218] D. Feldman, A. Munteanu, and C. Sohler. “Smallest enclosing ball for probabilistic data”. In: *Procs. of the Symposium on Computational Geometry 2014*. 2014, p. 214 (cit. on pp. 87, 212).

[219] D. Feldman, M. Schmidt, and C. Sohler. “Turning Big Data Into Tiny Data: Constant-Size Coresets for k-Means, PCA, and Projective Clustering”. In: *SIAM Journal of Computing* 49.3 (2020), pp. 601–657 (cit. on pp. 87, 91, 212). **SFB876-A2**

[220] M. Feldman, A. Norouzi-Fard, O. Svensson, and R. Zenklusen. “The one-way communication complexity of submodular maximization with applications to streaming and robustness”. In: *Procs. of the ACM SIGACT Symposium on Theory of Computing 2020*. 2020, pp. 1363–1374 (cit. on pp. 74, 75, 77).

[221] X. Feng, Y. Zhang, and J. Glass. “Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition”. In: *Procs. of the IEEE Int. Conference on Acoustics, Speech and Signal Processing 2014*. IEEE. 2014, pp. 1759–1763 (cit. on p. 260).

[222] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. “Scalable kernels for graphs with continuous attributes”. In: *Advances in Neural Information Processing Systems 26: Procs. of the 2013 Conference*. Erratum available at http://image.diku.dk/aasa/papers/graphkernels_nips_erratum.pdf. 2013, pp. 216–224 (cit. on p. 124).

[223] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. “Do we need hundreds of classifiers to solve real world classification problems?” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 3133–3181 (cit. on p. 339).

[224] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec. “GNNAutoScale: Scalable And Expressive Graph Neural Networks via Historical Embeddings”. In: *Procs. of the Int. Conference on Machine Learning 2021*. 2021 (cit. on pp. 130, 137, 139, 140, 142). **SFB876-A6**

[225] M. Fey and J. E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *Procs. of the ICLR Workshop on Representation Learning on Graphs and Manifolds 2019*. 2019. DOI: arXiv:1903.02428 (cit. on pp. 126, 130, 131, 134). **SFB876-A6,SFB876-B2**

[226] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. “SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels”. In: *Procs. of the IEEE Conference on Computer Vision and Pattern Recognition 2018*. 2018. DOI: arXiv:1711.08920 (cit. on pp. 130, 133). **SFB876-B2, SFB876-A6**

[227] M. Fey, J.-G. Yuen, and F. Weichert. “Hierarchical Inter-Message Passing for Learning on Molecular Graphs”. In: *Procs. of the ICML Graph Representation Learning and Beyond (GRL+) Workhop 2020*. 2020 (cit. on p. 132).

[228] H. Fichtenberger, M. Gillé, M. Schmidt, C. Schwiegelshohn, and C. Sohler. “BICO: Birch meets Coresets for k-means”. In: *Procs. of the European Symposium on Algorithms 2013*. Ed. by H. L. Bodlaender and G. F. Italiano. Springer, 2013. URL: http://link.springer.com/chapter/10.1007%5C%2F978-3-642-40450-4_41 (cit. on p. 213). **SFB876-A2**

[229] R. Fischer, N. Piatkowski, C. Pelletier, G. Webb, F. Petitjean, and K. Morik. “No Cloud on the Horizon: Probabilistic Gap Filling in Satellite Image Series”. In: *Procs. of the IEEE Int. Conference on Data Science and Advanced Analytics 2020*. Ed. by G. Webb, L. Cao, Z. Zhang, V. S. Tseng, G. Williams, and M. Vlachos. Environmental and Geo-spatial Data Analytics. The Institute of Electrical and Electronics Engineers, Inc. (IEEE), Oct. 2020, pp. 546–555. URL: https://ieeexplore.ieee.org/document/9260084 (cit. on p. 101).

[230] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. “An analysis of approximations for maximizing submodular set functions—II”. In: *Polyhedral combinatorics* (1978), pp. 73–87 (cit. on pp. 75, 76).

[231] FIT consortium. *FIT IoT-LAB*. FIT IoT-LAB, 2021. URL: https://iot-lab.info/ (cit. on p. 35).

[232] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer New York, 2013 (cit. on p. 378).

[233] M. Fragkoulis, D. Spinellis, P. Louridas, and A. Bilas. “Relational Access to Unix Kernel Data Structures”. In: *Procs. of the European Conference on Computer Systems 2014*. EuroSys ’14. New York, NY, USA: Association for Computing Machinery, 2014. DOI: https://doi.org/10.1145/2592798.2592802 (cit. on pp. 17, 19).

[234] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. M. Bronstein, and F. Monti. “SIGN: Scalable Inception Graph Neural Networks”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020 (cit. on pp. 130, 137).

[235] P. I. Frazier. “A tutorial on Bayesian optimization”. In: *arXiv: Computing Research Repository* (2018). DOI: arXiv:1807.02811 (cit. on pp. 95, 96).

[236] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. “Optimal Assignment Kernels for Attributed Molecular Graphs”. In: *Procs. of the Int. Conference on Machine Learning 2005*. 2005, pp. 225–232 (cit. on p. 121).

[237] H. Funke and J. Teubner. “Data-Parallel Query Processing on Non-Uniform Data”. In: *Procs. of the VLDB Endowment* (2020) (cit. on pp. 265, 268). **SFB876-A2**

[238] P. Gai, G. Lipari, and M. D. Natale. “Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-Chip”. In: *Procs. of the IEEE Real-Time Systems Symposium 2001*. 2001, pp. 73–83. DOI: http://dx.doi.org/10.1109/REAL.2001.990598 (cit. on pp. 361, 362).

[239] L. Galli and C.-J. Lin. “A Study on Truncated Newton Methods for Linear Classification”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021) (cit. on p. 278).

[240] P. R. Gankidi and J. Thangavelautham. “FPGA architecture for deep learning and its application to planetary robotics”. In: *Procs. of the IEEE Aerospace Conference 2017*. Mar. 2017, pp. 1–9 (cit. on p. 254).

[241] H. Gao and S. Ji. “Graph U-Net”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019 (cit. on p. 135).

[242] Y. Gao et al. “Estimating gpu memory consumption of deep learning models”. In: *Procs. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering 2020*. 2020, pp. 1342–1352. URL: https://www.microsoft.com/en-us/research/uploads/prod/2020/09/dnnmem.pdf (cit. on p. 9).

[243] W. Gaul and M. Schader. “A new algorithm for two-mode clustering”. In: *Data Analysis and Information Systems*. Springer, 1996, pp. 15–23 (cit. on p. 231).

[244] F. Geerts, B. Goethals, and T. Mielikäinen. “Tiling databases”. In: *Procs. of the Int. Conference on Discovery Science 2004*. Springer. 2004, pp. 278–289 (cit. on p. 233).

[245] L. N. Geppert, K. Ickstadt, A. Munteanu, J. Quedenfeld, and C. Sohler. “Random projections for Bayesian regression”. In: *Statistics and Computing* 27.1 (2017), pp. 79–101. DOI: http://doi.org/10.1007/s11222-015-9608-z (cit. on pp. 86, 89, 93, 94). **SFB876-C4**

[246] L. N. Geppert, K. Ickstadt, A. Munteanu, and C. Sohler. “Streaming statistical models via Merge & Reduce”. In: *Int. Journal of Data Science and Analytics* 10.4 (2020), pp. 331–347. DOI: https://doi.org/10.1007/s41060-020-00226-0 (cit. on pp. 87, 88, 90). **SFB876-C4**

[247] L. N. Geppert. “Bayesian and Frequentist Regression Approaches for Very Large Data Sets”. PhD thesis. TU Dortmund University, 2018. DOI: http://dx.doi.org/10.17877/DE290R-19931 (cit. on pp. 89, 94, 95).

[248] L. Gerhorst, B. Herzog, S. Reif, W. Schröder-Preikschat, and T. Hönig. “AnyCall: Fast and Flexible System-Call Aggregation”. In: *Procs. of the Workshop on Programming Languages and Operating Systems 2021*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1–8. DOI: https://doi.org/10.1145/3477113.3487267 (cit. on p. 17).

[249] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter. “NoSQL database systems: a survey and decision guidance”. In: *Computer Science - Research and Development* 32.3-4 (2017), pp. 353–365 (cit. on p. 89).

[250] P. Geurts, D. Ernst, and L. Wehenkel. “Extremely randomized trees”. In: *Machine Learning* 63.1 (2006), pp. 3–42 (cit. on pp. 341, 351).

[251] S. Gidaris, P. Singh, and N. Komodakis. “Unsupervised Representation Learning by Predicting Image Rotations”. In: *Procs. of the Int. Conference on Learning Representations 2018*. 2018, pp. 1–16 (cit. on p. 163).

[252] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. “Neural Message Passing for Quantum Chemistry”. In: *Procs. of the Int. Conference on Machine Learning 2017*. 2017, pp. 1263–1272 (cit. on pp. 116, 125, 126, 130–132).

[253] D. Ginsbourger, J. Janusevskis, and R. Le Riche. *Dealing with asynchronicity in parallel Gaussian Process based global optimization*. Tech. rep. 2011. URL: https://hal.archives-ouvertes.fr/hal-00507632 (cit. on pp. 288, 292).

[254] D. Ginsbourger, R. Le Riche, and L. Carraro. “Kriging is Well-Suited to Parallelize Optimization”. In: *Computational Intelligence in Expensive Optimization Problems*. Springer, 2010, pp. 131–162 (cit. on pp. 287, 288, 290, 292, 301).

[255] X. Glorot, A. Bordes, and Y. Bengio. “Deep sparse rectifier neural networks”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2011*. 2011, pp. 315–323 (cit. on p. 261).

[256] M. X. Goemans and D. P. Williamson. “Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming”. In: *Journal of the Association for Computing Machinery* 42.6 (1995), pp. 1115–1145 (cit. on pp. 145, 147, 148, 153).

[257] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1983 (cit. on p. 150).

[258] R. Gomes and A. Krause. “Budgeted Nonparametric Learning from Data Streams”. In: *Procs. of the Int. Conference on Machine Learning 2010*. Vol. 1. 2010, p. 3 (cit. on p. 76).

[259] A. Gomez, L. Sigrist, M. Magno, L. Benini, and L. Thiele. “Dynamic Energy Burst Scaling for Transiently Powered Systems”. In: *Procs. of the Design, Automation and Test in Europe Conference 2016*. DATE ’16. San Jose, CA, USA: EDA Consortium, 2016, pp. 349–354. URL: http://dl.acm.org/citation.cfm?id=2971808.2971888 (cit. on pp. 49, 57).

[260] A. Gomez. “On-demand communication with the batteryless MiroCard: demo abstract”. In: *Procs. of the Conference on Embedded Networked Sensor Systems 2020*. 2020 (cit. on p. 57).

[261] A. Gomez, L. Sigrist, T. Schalch, L. Benini, and L. Thiele. “Efficient, Long-Term Logging of Rich Data Sensors Using Transient Sensor Nodes”. In: *ACM Transactions on Embedded Computing Systems* 17.1 (2017), 4:1–4:23 (cit. on p. 47).

[262] A. Gomez, A. Tretter, P. A. Hager, P. Sanmugarajah, L. Benini, and L. Thiele. “Data-Flow Driven Partitioning of Machine Learning Applications for Optimal Energy Use in Batteryless Systems”. In: *ACM Transactions on Embedded Computing Systems* (Feb. 2022). DOI: https://doi.org/10.1145/3520135 (cit. on p. 54).

[263] X. Gong and F. Wang. “Three Improvements on KNN-NPR for Traffic Flow Forecasting”. In: *Procs. of the Int. Conference on Intelligent Transportation Systems 2002*. 2002, pp. 736–740 (cit. on p. 102).

[264] T. F. Gonzalez. “Clustering to Minimize the Maximum Intercluster Distance”. In: *Theoretical Computer Science* 38 (1985), pp. 293–306 (cit. on pp. 200, 203).

[265] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (cit. on p. XI).

[266] I. Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27: Procs. of the 2014 Conference*. 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (cit. on p. 250).

[267] M. Gorlatova, A. Wallwater, and G. Zussman. “Networking Low-Power Energy Harvesting Devices: Measurements and Algorithms”. In: *IEEE Transactions on Mobile Computing* 12.9 (2013), pp. 1853–1865 (cit. on p. 47).

[268] A. B. Graf and S. Borer. “Normalization in support vector machines”. In: *Pattern Recognition: Procs. of the German Association for Pattern Recognition Symposium 2001*. Springer. 2001, p. 277 (cit. on p. 81).

[269] R. L. Graham. “Bounds on Multiprocessing Timing Anomalies”. In: *SIAM Journal of Applied Mathematics* 17.2 (1969), pp. 416–429 (cit. on p. 366).

[270] C. Gregg and K. Hazelwood. “Where Is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer”. In: *Procs. of the IEEE Int. Symposium on Performance Analysis of Systems and Software 2011*. IEEE Press, 2011, pp. 134–144 (cit. on p. 380).

[271] M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017 (cit. on p. 117).

[272] M. Grohe. “Word2vec, Node2vec, Graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv:2003.12590 (cit. on p. 116).

[273] M. Grohe, K. Kersting, M. Mladenov, and A. E. Selman. “Dimension Reduction via Colour Refinement”. In: *Procs. of the European Symposium on Algorithms 2014*. 2014. URL: http://link.springer.com/chapter/10.1007/978-3-662-44777-2_42 (cit. on p. 118).

[274] P. Grosjean and S. Urbanek. *R Benchmark 2.5 Suite*. 2008. URL: http://r.research.att.com/benchmarks/R-benchmark-25.R (cit. on p. 315).

[275] S. Gu and Y. Yang. “A Deep Learning Algorithm for the Max-Cut Problem Based on Pointer Network Structure with Supervised Learning and Reinforcement Learning Strategies”. In: *Mathematics* 8.2 (2020) (cit. on p. 145).

[276] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. “Continuous deep Q-learning with model-based acceleration”. In: *Procs. of the Int. Conference on Machine Learning 2016*. 2016, pp. 2829–2838 (cit. on p. 259).

[277] F. Guidi and C. Sacerdoti Coen. “A survey on retrieval of mathematical knowledge”. In: *Procs. of the Int. Conference on Intelligent Computer Mathematics 2015*. Springer, 2015, pp. 296–315 (cit. on p. 161).

[278] C. Guo, W. Luk, and W. Xu. “Non-linear function evaluation reusing matrix-vector multipliers”. In: *Procs. of the IEEE Int. Conference on ASIC 2019*. IEEE, pp. 1–4 (cit. on pp. 251, 256).

[279] C. Guo et al. “Breaking the glass ceiling for embedding-based classifiers for large output spaces”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. Vol. 32. 2019 (cit. on p. 283).

[280] A. Gupta, K. Ni, O. Prakash, X. S. Hu, and H. Amrouch. “Temperature Dependence and Temperature-Aware Sensing in Ferroelectric FET”. In: *Procs. of the IEEE Int. Reliability Physics Symposium 2020*. 2020 (cit. on p. 328).

[281] A. Gupta, R. Krauthgamer, and J. R. Lee. “Bounded Geometries, Fractals, and Low-Distortion Embeddings”. In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2003*. IEEE. 2003, pp. 534–543. DOI: 10.1109/SFCS.2003.1238226 (cit. on p. 201).

[282] U. Gupta et al. “Chasing Carbon: The Elusive Environmental Footprint of Computing”. In: *Procs. of the IEEE Int. Symposium on High-Performance Computer Architecture 2021*. 2021, pp. 854–867. DOI: https://doi.org/10.1109/HPCA51647.2021.00076 (cit. on p. 3).

[283] B. Haasdonk and C. Bahlmann. “Learning with Distance Substitution Kernels”. In: *Pattern Recognition*. Vol. 3175. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 220–227 (cit. on p. 123).

[284] L. Habel, A. Molina, T. Zaksek, K. Kersting, and M. Schreckenberg. “Traffic simulations with empirical data – How to replace missing traffic flows?” In: *Traffic and Granular Flow ’15*. Ed. by V. L. Knoop and W. Daamen. Springer, May 2016, pp. 491–498. DOI: http://dx.doi.org/10.1007/978-3-319-33482-0_62 (cit. on p. 92). **SFB876-B4**

[285] Haci Bayhan, Dietmar Ebel, Timo Erler, Lorenz Kiebler, Kira Schmeltzpfenning, and Robert Schulze Forsthövel. *Logistik IT im Wandel: Einbindung dezentraler IT-Strukturen am Beispiel eines Cyberphysischen Produktionssystems (CPPS)*. 2021. URL: https://leistungszentrum-logistik-it.de/wp-content/uploads/2021/02/Whitepaper_Logistik-IT-im-Wandel.pdf (cit. on p. 45).

[286] F. Hadiji, A. Molina, S. Natarajan, and K. Kersting. "Poisson Dependency Networks: Gradient Boosted Models for Multivariate Count Data". In: *Machine Learning Journal* 100.2 (2015), pp. 477–507. DOI: http://dx.doi.org/10.1007/s10994-015-5506-z (cit. on p. 92). **SFB876-B4**

[287] S. F. Hafstein, R. Chrobok, A. Pottmeier, and M. S. and F. Mazur. "A High-Resolution Cellular Automata Traffic Simulation Model with Application in a Freeway Traffic Information System". In: *Computer-Aided Civil and Infrastructure Engineering* 19.5 (2004), pp. 338–350 (cit. on p. 101).

[288] R. T. Haftka, D. Villanueva, and A. Chaudhuri. "Parallel surrogate-assisted global optimization with expensive functions – a survey". In: *Structural and Multidisciplinary Optimization* 54.1 (2016), pp. 3–13 (cit. on p. 287).

[289] L. A. Hall and D. B. Shmoys. "Jackson's Rule for Single-Machine Scheduling: Making a Good Heuristic Better". In: *Mathematics of Operations Research* 17.1 (1992), pp. 22–35. DOI: https://doi.org/10.1287/moor.17.1.22 (cit. on p. 365).

[290] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst. "Communication Centric Design in Complex Automotive Embedded Systems". In: *Procs. of the Euromicro Conference on Real-Time Systems 2017*. 2017, 10:1–10:20 (cit. on p. 368).

[291] W. L. Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (2020), pp. 1–159 (cit. on pp. 130, 145).

[292] W. L. Hamilton, R. Ying, and J. Leskovec. "Inductive Representation Learning on Large Graphs". In: *arXiv: Computing Research Repository* (2017). DOI: arXiv:1706.02216 (cit. on pp. 125, 126, 130, 132, 136, 141).

[293] J. Han, K. Song, F. Nie, and X. Li. "Bilateral k-Means Algorithm for Fast Co-Clustering." In: *Procs. of the AAAI Conference on Artificial Intelligence 2017*. 2017, pp. 1969–1975 (cit. on p. 232).

[294] S. Har-Peled. "A Simple Algorithm for Maximum Margin Classification, Revisited". In: *arXiv: Computing Research Repository* (2015). DOI: arXiv:1507.01563 (cit. on p. 87).

[295] S. Har-Peled and S. Mazumdar. "On coresets for k-means and k-median clustering". In: *Procs. of the ACM Symposium on Theory of Computing 2004*. ACM, 2004, pp. 291–300 (cit. on pp. 86–88).

[296] S. Har-Peled and B. Raichel. "The Fréchet Distance Revisited and Extended". In: *ACM Transactions on Algorithms* 10.1 (2014). Previously appeared in Procs. of the Int. Symposium on Computational Geometry 2011, 3:1–3:22. DOI: http://doi.acm.org/10.1145/2532646 (cit. on p. 202).

[297] S. Har-Peled, D. Roth, and D. Zimak. "Maximum Margin Coresets for Active and Noise Tolerant Learning". In: *Procs. of the Int. Joint Conference on Artificial Intelligence 2007*. 2007, pp. 836–841 (cit. on p. 87).

[298] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd. Statistics. Springer, 2009 (cit. on p. XI).

[299] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. New York, USA: Springer, 2001 (cit. on p. 184).

[300] B. He et al. "Relational Joins on Graphics Processors". In: *Procs. of the ACM SIGMOD Int. Conference on Management of Data 2008*. Vancouver, BC, Canada, June 2008, pp. 511–524 (cit. on p. 387).

[301] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. M. Kadie. "Dependency Networks for Collaborative Filtering and Data Visualization". In: *Procs. of the Conference on Uncertainty in Artificial Intelligence 2000*. 2000, pp. 264–273. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article_id=31%5C&proceeding_id=16 (cit. on p. 92).

[302] M. Heimel, M. Saecker, H. Pirk, S. Manegold, and V. Markl. “Hardware-Oblivious Parallelism for In-Memory Column-Stores”. In: *Procs. of the VLDB Endowment* 6.9 (2013), pp. 709–720 (cit. on p. 382).

[303] M. Heinrich, A. Munteanu, and C. Sohler. “Asymptotically exact streaming algorithms”. In: *arXiv: Computing Research Repository* (2014). DOI: arXiv:1408.1847 (cit. on p. 90).

[304] H. Hellbrück, M. Pagel, A. Köller, D. Bimschas, D. Pfisterer, and S. Fischer. “Using and Operating Wireless Sensor Network Testbeds with WISEBED”. In: *Procs. of the IFIP Annual Mediterranean Ad Hoc Networking Workshop 2011*. 2011, pp. 171–178 (cit. on p. 35).

[305] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau. “Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning”. In: *Journal of Machine Learning Research* 21.248 (2020), pp. 1–43. URL: http://jmlr.org/papers/v21/20-312.html (cit. on p. 6).

[306] S. Henwood, F. Leduc-Primeau, and Y. Savaria. “Layerwise Noise Maximisation to Train Low-Energy Deep Neural Networks”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1912.10764 (cit. on pp. 326, 327).

[307] D. Herr, Q. Han, S. Lohmann, and T. Ertl. “Visual Clutter Reduction through Hierarchy-based Projection of High-dimensional Labeled Data”. In: *Procs. of the Graphics Interface Conference 2016*. 2016, pp. 109–116 (cit. on p. 216).

[308] S. Hess, W. Duivesteijn, P.-J. Honysz, and K. Morik. “The SpectACl of Nonconvex Clustering: a Spectral Approach to Density-Based Clustering”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2019*. 2019 (cit. on p. 231). **SFB876-C1**

[309] S. Hess and K. Morik. “C-SALT: Mining Class-Specific ALTerations in Boolean Matrix Factorization”. In: *Procs. of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases 2017*. Springer, 2017. URL: https://link.springer.com/content/pdf/10.1007/978-3-319-71249-9_33.pdf (cit. on pp. 228, 238). **SFB876-C1**

[310] S. Hess, K. Morik, and N. Piatkowski. “The PRIMPING routine—Tiling through proximal alternating linearized minimization”. In: *Data Mining and Knowledge Discovery* 31.4 (July 2017), pp. 1090–1131. DOI: https://doi.org/10.1007/s10618-017-0508-z (cit. on pp. 228, 234, 237–239, 246). **SFB876-A1, SFB876-C1**

[311] S. Hess, N. Piatkowski, and K. Morik. “The Trustworthy Pal: Controlling the False Discovery Rate in Boolean Matrix Factorization”. In: *Procs. of the 2018 SIAM Int. Conference on Data Mining 2018*. SIAM. 2018, pp. 405–413. DOI: https://doi.org/10.1137/1.9781611975321.46 (cit. on pp. 228, 237, 238). **SFB876-A1, SFB876-C1**

[312] S. Hess, G. Pio, M. Hochstenbach, and M. Ceci. “BROCCOLI: overlapping and outlier-robust biclustering through proximal stochastic gradient descent”. In: *Data Mining and Knowledge Discovery* (2021), pp. 1–35 (cit. on pp. 228, 238, 244, 246).

[313] J. Hester and J. Sorber. “Flicker: Rapid prototyping for the batteryless internet-of-things”. In: *Procs. of the SenSys Conference 2017*. ACM, 2017 (cit. on p. 57).

[314] S. Hido and H. Kashima. “A Linear-Time Graph Kernel”. In: *Procs. of the IEEE Int. Conference on Data Mining 2009*. 2009, pp. 179–188 (cit. on p. 125).

[315] T. Hirtzlin et al. “Implementing Binarized Neural Networks with Magnetoresistive RAM without Error Correction”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1908.04085 (cit. on p. 327).

[316] T. Hirtzlin et al. “Outstanding Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks”. In: *Procs. of the Int. Conference on Artificial Intelligence Circuits and Systems 2019*. 2019 (cit. on pp. 327, 330, 331).

[317] D. S. Hochbaum and D. B. Shmoys. “A best possible heuristic for the *k*-center problem”. In: *Mathematics of Operations Research* 10.2 (1985), pp. 180–184 (cit. on p. 200).

[318] S. Hochreiter et al. “FABIA: Factor Analysis for Bicluster Acquisition”. In: *Bioinformatics (Oxford, England)* 26 (Apr. 2010), pp. 1520–7 (cit. on p. 239).

[319] W. Hoeffding. “Probability inequalities for sums of bounded random variables”. In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30 (cit. on p. 377).

[320] W. Hu et al. “Open Graph Benchmark: Datasets for Machine Learning on Graphs”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv : 2005 . 00687 (cit. on pp. 129, 141). **SFB876-A6**

[321] Q. Huang, H. He, A. Singh, S. N. Lim, and A. R. Benson. “Combining Label Propagation and Simple Models Out-performs Graph Neural Networks”. In: *Procs. of the Int. Conference on Learning Representations 2021*. 2021 (cit. on pp. 130, 137).

[322] R. Huang, V. Pavlovic, and D. Metaxas. “A New Spatio-Temporal MRF Framework for Video-based Object Segmentation”. In: *Procs. of the Int. Workshop on Machine Learning for Vision-based Motion Analysis 2008*. Marseille, France, 2008 (cit. on p. 102).

[323] W. Huang, T. Zhang, Y. Rong, and J. Huang. “Adaptive Sampling Towards Fast Graph Representation Learning”. In: *Advances in Neural Information Processing Systems 31: Procs. of the 2018 Conference*. 2018 (cit. on p. 136).

[324] W.-H. Huang, J.-J. Chen, H. Zhou, and C. Liu. “PASS: Priority Assignment of Real-Time Tasks with Dynamic Suspending Behavior under Fixed-Priority Scheduling”. In: *Procs. of the Design Automation Conference 2015*. 2015 (cit. on p. 363). **SFB876-B2**

[325] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. “Binarized Neural Networks”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. 2016, pp. 4107–4115 (cit. on pp. 10, 329).

[326] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations”. In: *Journal of Machine Learning Research* 18 (2018) (cit. on p. 253).

[327] J. H. Huggins, T. Campbell, and T. Broderick. “Coresets for Scalable Bayesian Logistic Regression”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. 2016, pp. 4080–4088. URL: http://papers.nips.cc/paper/6486-coresets-for-scalable-bayesian-logistic-regression (cit. on pp. 87, 90, 91).

[328] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Parallel Algorithm Configuration”. In: *Learning and Intelligent Optimization*. Ed. by Y. Hamadi and M. Schoenauer. Lecture Notes in Computer Science 7219. Springer Berlin Heidelberg, 2012, pp. 55–70. URL: http://link.springer.com/chapter/10.1007/978-3-642-34413-8_5 (cit. on p. 287).

[329] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *Learning and Intelligent Optimization*. Ed. by C. A. Coello. Lecture Notes in Computer Science 6683. Springer Berlin Heidelberg, 2011, pp. 507–523. URL: http://link.springer.com/chapter/10.1007/978-3-642-25566-3_40 (cit. on p. 292).

[330] T. I. “Longest Common Extensions with Recompression”. In: *Procs. of the Symposium on Combinatorial Pattern Matching 2017*. Ed. by J. Kärkkäinen, J. Radoszewski, and W. Rytter. Vol. 78. Leibniz Int. Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, 18:1–18:15 (cit. on p. 157).

[331] A. Ihler and D. McAllester. “Particle Belief Propagation”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2009*. Ed. by D. van Dyk and M. Welling. 2009, pp. 256–263 (cit. on p. 407).

[332] A. T. Ihler, J. W. Fischer III, and A. S. Willsky. “Loopy Belief Propagation: Convergence and Effects of Message Errors”. In: *Journal of Machine Learning Research* 6 (Dec. 2005), pp. 905–936 (cit. on p. 413).

[333] P. Indyk. “High-dimensional Computational Geometry”. PhD thesis. Stanford University, 2000 (cit. on p. 201).

[334] Intel Corporation. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. 2016. URL: http://www.intel.de/content/www/de/de/architecture-and-technology/64-ia-32-architectures-optimization-manual.html (cit. on p. 408).

[335] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Procs. of the Int. Conference on Machine Learning 2015*. 2015, pp. 448–456. URL: http://jmlr.org/proceedings/papers/v37/ioffe15.html (cit. on p. 166).

[336] H. Jain, Y. Prabhu, and M. Varma. "Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications". In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2016*. 2016, pp. 935–944 (cit. on p. 284).

[337] K. Jain, M. Mahdian, and A. Saberi. "A New Greedy Approach for Facility Location Problems". In: *Procs. of the Symposium on Theory of Computing 2002*. ACM, 2002, pp. 731–740. DOI: http://doi.acm.org/10.1145/509907.510012 (cit. on p. 200).

[338] J. Janusevskis, R. Le Riche, and D. Ginsbourger. *Parallel Expected Improvements for Global Optimization: Summary, Bounds and Speed-Up*. Tech. rep. 2011, pp. 1–21. URL: https://hal.archives-ouvertes.fr/hal-00613971 (cit. on p. 292).

[339] J. Janusevskis, R. Le Riche, D. Ginsbourger, and R. Girdziusas. "Expected Improvements for the Asynchronous Parallel Global Optimization of Expensive Functions: Potentials and challenges". In: *Learning and Intelligent Optimization*. Springer, 2012, pp. 413–418 (cit. on pp. 288, 292).

[340] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. "What is the best multi-stage architecture for object recognition?" In: *Procs. of the IEEE Int. Conference on Computer Vision 2009*. IEEE Computer Society, 2009, pp. 2146–2153 (cit. on p. 242).

[341] H. Jayakumar, A. Raha, and V. Raghunathan. "QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers". In: *Procs. of the Int. Conference on VLSI Design 2014* (2014) (cit. on p. 53).

[342] A. Jez. "Recompression: A Simple and Powerful Technique for Word Equations". In: *Journal of the Association for Computing Machinery* 63.1 (Feb. 2016) (cit. on pp. 154, 157).

[343] H. Jiang, T. Kathuria, Y. T. Lee, S. Padmanabhan, and Z. Song. "A Faster Interior Point Method for Semidefinite Programming". In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2020*. 2020, pp. 910–918 (cit. on p. 150).

[344] W. Jiang and S. G. Kong. "Block-based neural networks for personalized ECG signal classification". In: *IEEE Transactions on Neural Networks* 18.6 (2007), pp. 1750–1761 (cit. on p. 254).

[345] W. Jiang, S. G. Kong, and G. D. Peterson. "Continuous heartbeat monitoring using evolvable block-based neural networks". In: *Procs. of the IEEE Int. Joint Conference on Neural Networks 2006*. IEEE. 2006, pp. 1950–1957 (cit. on p. 254).

[346] W. Jiang, S. G. Kong, and G. D. Peterson. "ECG signal classification using block-based neural networks". In: *Procs. of the IEEE Int. Joint Conference on Neural Networks 2005*. Vol. 1. IEEE. 2005, pp. 326–331 (cit. on p. 254).

[347] Z. Jiang, H. Liu, B. Fu, and Z. Wu. "Generalized ambiguity decompositions for classification with applications in active learning and unsupervised ensemble pruning". In: *Procs. of the AAAI Conference on Artificial Intelligence 2017*. 2017, pp. 2073–2079 (cit. on p. 340).

[348] D. R. Jones, M. Schonlau, and W. J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13.4 (1998), 455–492. URL: http://link.springer.com/article/10.1023/A:1008306431147 (cit. on p. 286).

[349] P. Joshi, D. Colombi, B. Thors, L. E. Larsson, and C. Törnevik. "Output Power Levels of 4G User Equipment and Implications on Realistic RF EMF Exposure Assessments". In: *IEEE Access* 5 (2017), pp. 4545–4550 (cit. on p. 435).

[350] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. “Bag of Tricks for Efficient Text Classification”. In: *Procs. of the Conference of the European Chapter of the Association for Computational Linguistics 2017*. 2017, pp. 1–55. DOI: arXiv:1511.09249 (cit. on pp. 175, 176).

[351] B. D. Jovanovic and P. S. Levy. “A Look at the Rule of Three”. In: *The American Statistician* 51.2 (1997), pp. 137–139. URL: https://www.tandfonline.com/doi/abs/10.1080/00031305.1997.10473947 (cit. on p. 79).

[352] S. Jung et al. “A crossbar array of magnetoresistive memory devices for in-memory computing”. In: *Nature* 601.7892 (2022), pp. 211–216 (cit. on p. 4).

[353] T. Kalibera, P. Maj, F. Morandat, and J. Vitek. “A Fast Abstract Syntax Tree Interpreter for R”. In: *Procs. of the ACM SIGPLAN/SIGOPS Int. Conference on Virtual Execution Environments 2014*. VEE ’14. ACM, 2014, pp. 89–102 (cit. on p. 315).

[354] R. Kannan, S. Vempala, and D. P. Woodruff. “Principal Component Analysis and Higher Correlations for Distributed Data”. In: *Procs. of the Conference on Learning Theory 2014*. 2014, pp. 1040–1057 (cit. on p. 89).

[355] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. “A local search approximation algorithm for k-means clustering”. In: *Computational Geometry: Theory and Applications* 28.2-3 (2004). Previously appeared in Procs. of the Int. Symposium on Computational Geometry 2002, pp. 89–112 (cit. on p. 184).

[356] S. Karaev, P. Miettinen, and J. Vreeken. “Getting to know the unknown unknowns: Destructive-noise resistant Boolean matrix factorization”. In: *Procs. of the SIAM Int. Conference on Data Mining 2015*. SIAM. 2015, pp. 325–333 (cit. on p. 239).

[357] O. Kariv and S. Hakimi. “An Algorithmic Approach to Network Location Problems. II: The p-Medians”. In: *SIAM Journal on Applied Mathematics* 37.3 (1979), pp. 539–560 (cit. on p. 183).

[358] T. Karnagel, R. Mueller, and G. M. Lohman. “Optimizing GPU-Accelerated Group-By and Aggregation”. In: *Procs. of the Int. Workshop on Accelerating Data Management Systems at the Int. Conference on Very Large Data Bases 2015*. 2015, pp. 13–24 (cit. on pp. 384, 392).

[359] R. M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*. Springer US, 1972, pp. 85–103 (cit. on p. 144).

[360] V. Kartsch, S. Benatti, M. Mancini, M. Magno, and L. Benini. “Smart wearable wristband for EMG based gesture recognition powered by solar energy harvester”. In: *Procs. of the Int. Symposium on Computer Architecture 2018*. IEEE. 2018, pp. 1–5 (cit. on p. 61).

[361] G. Karypis and V. Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392 (cit. on p. 140).

[362] S. Kato, R. Rajkumar, and Y. Ishikawa. “A loadable real-time scheduler suite for multicore platforms”. In: *Technical Report CMU-ECE-TR09-12* (2009) (cit. on p. 374).

[363] L. Kaufman and P. J. Rousseeuw. “Clustering by means of medoids”. In: *Statistical Data Analysis Based on the L_1 Norm and Related Methods*. Ed. by Y. Dodge. North-Holland, 1987, pp. 405–416 (cit. on pp. 182, 188).

[364] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990 (cit. on pp. 215, 218).

[365] L. Kaufman and P. J. Rousseeuw. “Partitioning Around Medoids (Program PAM)”. In: *Finding Groups in Data*. John Wiley&Sons, 1990. Chap. 2, pp. 68–125 (cit. on pp. 182, 188).

[366] J. Kays, A. Seack, T. Smirek, F. Westkamp, and C. Rehtanz. “The Generation of Distribution Grid Models on the Basis of Public Available Data”. In: 32.3 (2017), pp. 2346–2353 (cit. on p. 185).

[367] E. Kazemi, M. Mitrovic, M. Zadimoghaddam, S. Lattanzi, and A. Karbasi. “Submodular Streaming in All Its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019, pp. 3311–3320. URL: http://proceedings.mlr.press/v97/kazemi19a.html (cit. on pp. 76, 77).

[368] S. S. Keerthi, D. DeCoste, and T. Joachims. “A modified finite Newton method for fast solution of large scale linear SVMs.” In: *Journal of Machine Learning Research* 6.3 (2005) (cit. on p. 278).

[369] K. Kersting, B. Ahmadi, and S. Natarajan. “Counting belief propagation”. In: *Procs. of the Conference on Uncertainty in Artificial Intelligence 2009*. 2009, pp. 277–284 (cit. on p. 407).

[370] A. H. Khan. “Lightweight Neural Networks”. In: *arXiv: Computing Research Repository* (2017). DOI: arXiv:1712.05695 (cit. on p. 61).

[371] S. Khandagale, H. Xiao, and R. Babbar. “Bonsai: diverse and shallow trees for extreme multi-label classification”. In: *Machine Learning Journal* 109.11 (2020) (cit. on p. 283).

[372] S. Khot, G. Kindler, E. Mossel, and R. O’Donnell. “Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs?” In: *SIAM Journal on Computing (SICOMP)* 37.1 (2007), pp. 319–357 (cit. on p. 145).

[373] C. Kim et al. “FAST: Fast architecture sensitive tree search on modern CPUs and GPUs”. In: *Procs. of the ACM SIGMOD Int. Conference on Management of Data 2010*. ACM. 2010, pp. 339–350 (cit. on p. 341).

[374] D. Kim, T. Na, S. Yalamanchili, and S. Mukhopadhyay. “DeepTrain: A Programmable Embedded Platform for Training Deep Neural Networks”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2360–2370 (cit. on pp. 251, 253).

[375] T. N. Kipf and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *Procs. of the Int. Conference on Learning Representations 2017*. 2017 (cit. on pp. 125, 130, 132, 136, 141).

[376] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. “Neural Relational Inference for Interacting Systems”. In: *Procs. of the Int. Conference on Machine Learning 2018*. Ed. by J. Dy and A. Krause. Vol. 80. Procs. of Machine Learning Research. PMLR, June 2018, pp. 2688–2697 (cit. on p. 99).

[377] H. Kise, T. Ibaraki, and H. Mine. “Performance Analysis of six Approximation Algorithms for the One-Machine Maximum Lateness Scheduling Problem with Ready Times”. In: *Journal of the Operations Research Society of Japan* 22.3 (1979), pp. 205–224 (cit. on p. 365).

[378] J. Klicpera, A. Bojchevski, and S. Günnemann. “Predict then Propagate: Graph Neural Networks meet Personalized PageRank”. In: *Procs. of the Int. Conference on Learning Representations 2019*. 2019 (cit. on pp. 130, 132, 136, 137).

[379] O. Koch, N. M. Kriege, and L. Humbeck. “Chemical Similarity and Substructure Searches”. In: *Encyclopedia of Bioinformatics and Computational Biology - Volume 2*. Ed. by S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach. Elsevier, 2019, pp. 640–649. DOI: https://doi.org/10.1016/b978-0-12-809633-8.20195-7 (cit. on p. 116).

[380] P. Koch, M. Dreier, M. Maass, M. Böhme, H. Phan, and A. Mertins. “A recurrent neural network for hand gesture recognition based on accelerometer data”. In: *Procs. of the Int. Conference of the IEEE Engineering in Medicine and Biology Society 2019*. IEEE. 2019, pp. 5088–5091 (cit. on p. 61).

[381] S. Koppula et al. “EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM”. In: *Procs. of the Int. Symposium on Microarchitecture 2019*. 2019 (cit. on pp. 327, 329).

[382] A. Kopytov. *SysBench: a system performance benchmark*. 2004. URL: https://github.com/akopytov/sysbench (cit. on p. 28).

[383] I. Korb, H. Kotthaus, and P. Marwedel. “mmapcopy: Efficient Memory Footprint Reduction using Application-Knowledge”. In: *Procs. of the ACM Symposium on Applied Computing 2016*. 2016. DOI: https://dl.acm.org/doi/pdf/10.1145/2851613.2851736 (cit. on pp. 307, 310). **SFB876-A3**

[384] H. Kotthaus and M. Lang. *BenchR: Set of Benchmark of R*. TU Dortmund University. 2018. URL: https://github.com/allr/benchR (cit. on pp. 314, 315).

[385] H. Kotthaus. “Methods for Efficient Resource Utilization in Statistical Machine Learning Algorithms”. PhD thesis. Dortmund: TU Dortmund University, 2018. DOI: http://dx.doi.org/10.17877/DE290R-18928 (cit. on pp. 289, 293, 295–299, 301–303, 307, 309–312, 314, 317, 318, 320–323). **SFB876-A3**

[386] H. Kotthaus, I. Korb, M. Engel, and P. Marwedel. “Dynamic Page Sharing Optimization for the R Language”. In: *Procs. of the Symposium on Dynamic Languages 2014*. DLS ’14. Portland, Oregon, USA: ACM, 2014, pp. 79–90. DOI: https://dl.acm.org/doi/10.1145/2661088.2661094 (cit. on p. 307). **SFB876-A3**

[387] H. Kotthaus, I. Korb, M. Lang, B. Bischl, J. Rahnenführer, and P. Marwedel. “Runtime and Memory Consumption Analyses for Machine Learning R Programs”. In: *Journal of Statistical Computation and Simulation* 85.1 (2014), pp. 14–29. URL: http://www.tandfonline.com/eprint/T3mgYXAWdY4kWuDeSv2A/full (cit. on pp. 306, 323). **SFB876-A3**

[388] H. Kotthaus, J. Richter, A. Lang, M. Lang, and P. Marwedel. *Resource-Aware Scheduling Strategies for Parallel Machine Learning R Programs through RAMBO*. Stanford University, Palo Alto, California, July 2016, p. 195. URL: http://user2016.r-project.org/files/abs-book.pdf (cit. on p. 11).

[389] H. Kotthaus, L. Schönberger, A. Lang, J.-J. Chen, and P. Marwedel. “Can Flexible Multi-Core Scheduling Help to Execute Machine Learning Algorithms Resource-Efficiently?” In: *Procs. of the Int. Workshop on Software and Compilers for Embedded Systems 2019*. SCOPES ’19. ACM, 2019, pp. 59–62. DOI: https://dl.acm.org/doi/10.1145/3323439.3323986 (cit. on pp. 289, 292, 295). **SFB876-A3, SFB876-C5**

[390] H. Kotthaus et al. “RAMBO: Resource-Aware Model-Based Optimization with Scheduling for Heterogeneous Runtimes and a Comparison with Asynchronous Model-Based Optimization”. In: *Procs. of the Int. Conference on Learning and Intelligent Optimization 2017*. 2017, pp. 180–195. URL: https://www.springerprofessional.de/en/rambo-resource-aware-model-based-optimization-with-scheduling-fo/15164982 (cit. on pp. 289, 292). **SFB876-A3**

[391] M. Koyutürk and A. Grama. “PROXIMUS: a framework for analyzing very high dimensional discrete-attributed datasets”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2003*. ACM. 2003, pp. 147–156 (cit. on pp. 232, 233).

[392] S. Kramer, D. Ziegenbein, and A. Hamann. “Real world automotive benchmark for free”. In: *Procs. of the Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems 2015*. 2015 (cit. on p. 368).

[393] A. Krause and D. Golovin. *Submodular function maximization*. 2014. URL: http://www.cs.cmu.edu/afs/.cs.cmu.edu/Web/People/dgolovin/papers/submodular_survey12.pdf (cit. on p. 76).

[394] N. Kriege and P. Mutzel. “Subgraph Matching Kernels for Attributed Graphs”. In: *Procs. of the Int. Conference on Machine Learning 2012*. Omnipress, 2012 (cit. on p. 124).

[395] N. Kriege, P. Giscard, and R. C. Wilson. “On Valid Optimal Assignment Kernels and Applications to Graph Classification”. In: *arXiv: Computing Research Repository* (2016). DOI: arXiv:1606.01141 (cit. on pp. 121, 122). **SFB876-A6**

[396] N. M. Kriege. “Deep Weisfeiler-Lehman Assignment Kernels via Multiple Kernel Learning”. In: *Procs. of the European Symposium on Artificial Neural Networks 2019*. 2019 (cit. on p. 122). **SFB876-A6**

[397] N. M. Kriege, P.-L. Giscard, F. Bause, and R. C. Wilson. “Computing Optimal Assignments in Linear Time for Approximate Graph Matching”. In: *Procs. of the IEEE Int. Conference on Data Mining 2019*. 2019 (cit. on p. 117). **SFB876-A6**

[398] N. M. Kriege, F. D. Johansson, and C. Morris. “A Survey on Graph Kernels”. In: *Applied Network Science* 5.1 (2020), p. 6. DOI: https://doi.org/10.1007/s41109-019-0195-3 (cit. on p. 116). **SFB876-A6**

[399] H. Kriegel, E. Schubert, and A. Zimek. “The (black) art of runtime evaluation: Are we comparing algorithms or implementations?” In: *Knowledge and Information Systems (KAIS)* 52.2 (2017). Online first 2016, paginated 2017, pp. 341–378. DOI: https://doi.org/10.1007/s10115-016-1004-2 (cit. on pp. 192, 223).

[400] R. Krishnakumar. “Kernel Korner: Kprobes – a Kernel Debugger”. In: *Linux Journal* 2005.133 (May 2005), p. 11 (cit. on pp. 17, 18).

[401] A. Krivošija. “On clustering and related problems on curves under the Fréchet distance”. PhD thesis. TU Dortmund University, 2021. DOI: http://dx.doi.org/10.17877/DE290R-22055 (cit. on p. 204).

[402] A. Krivošija and A. Munteanu. “Probabilistic smallest enclosing ball in high dimensions via subgradient sampling”. In: *Procs. of the Int. Symposium on Computational Geometry 2019*. Ed. by G. Barequet and Y. Wang. Vol. 129. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, 47:1–47:14. DOI: https://doi.org/10.4230/LIPIcs.SoCG.2019.47 (cit. on pp. 87, 212). **SFB876-A2, SFB876-C4**

[403] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25: Procs. of the 2012 Conference*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105 (cit. on p. 254).

[404] F. Kschischang, S. Member, B. J. Frey, and H.-a. Loeliger. “Factor Graphs and the Sum-Product Algorithm”. In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 498–519 (cit. on pp. 105, 407, 411, 413).

[405] M. Kumar, R. Ghani, and Z.-S. Mei. “Data mining to predict and prevent errors in health insurance claims processing”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2010*. ACM. 2010, pp. 65–74 (cit. on pp. 201, 212).

[406] S. Kumar, S. Gollakota, and D. Katabi. “A Cloud-Assisted Design for Autonomous Driving”. In: *Procs. of the MCC Workshop on Mobile Cloud Computing 2012*. MCC ’12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 41–46. DOI: https://doi.org/10.1145/2342509.2342519 (cit. on p. 363).

[407] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Procs. of the Int. Conference on Machine Learning 2001*. Ed. by C. E. Brodley and A. P. Danyluk. Morgan Kaufmann, 2001, pp. 282–289 (cit. on pp. 106, 416).

[408] K. P. Lakshmi and M. Subadra. “A survey on FPGA based MLP realization for on-chip learning”. In: *International Journal of Scientific & Engineering Research* 4.1 (2013), pp. 1–9 (cit. on p. 253).

[409] W. H. K. Lam, Y. F. Tang, and M. Tam. “Comparison of two non-parametric models for daily traffic forecasting in Hong Kong”. In: *Journal of Forecasting* 25.3 (2006), pp. 173–192 (cit. on p. 102).

[410] P. Lancaster and M. Tismenetsky. *The Theory of Matrices*. Academic Press, 1985 (cit. on p. 150).

[411] G. N. Lance and W. T. Williams. “A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems”. In: *The Computer Journal* 9.4 (Feb. 1967), pp. 373–380. DOI: https://doi.org/10.1093/comjnl/9.4.373 (cit. on pp. 215, 216).

[412] G. N. Lance and W. T. Williams. “A Generalized Sorting Strategy for Computer Classifications”. In: *Nature* 212.5058 (Oct. 1966), pp. 218–218. DOI: https://doi.org/10.1038/212218a0 (cit. on pp. 215, 216).

[413] A. Lang and E. Schubert. “BETULA: Fast Clustering of Large Data with Improved BIRCH CF-Trees”. In: *Information Systems* (2021). DOI: https://doi.org/10.1016/j.is.2021.101918 (cit. on pp. 216, 219–221). **SFB876-A2**

[414] A. Lang and E. Schubert. “BETULA: Numerically Stable CF-Trees for BIRCH Clustering”. In: *Procs. of the Int. Conference on Similarity Search and Applications 2020*. best paper candidate. 2020, pp. 281–296. DOI: https://doi.org/10.1007/978-3-030-60936-8_22 (cit. on pp. 216, 219, 220). **SFB876-A2**

[415] M. Lang, B. Bischl, and D. Surmann. “batchtools: Tools for R to Work on Batch Systems”. In: *The Journal of Open Source Software* 2.10 (2017) (cit. on p. 294).

[416] M. Langberg and L. J. Schulman. “Universal ϵ-approximators for Integrals”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2010*. 2010, pp. 598–607 (cit. on p. 91).

[417] M. Langhammer and B. Pasca. “Activation Function Architectures for FPGAs”. In: *Procs. of the Int. Conference on Field Programmable Logic and Applications 2017*. IEEE. 2017, pp. 1–6 (cit. on p. 253).

[418] J. Lässig, K. Kersting, and K. Morik. *Computational Sustainability*. Ed. by J. Lässig, K. Kersting, and K. Morik. Springer, 2016. URL: http://link.springer.com/book/10.1007/978-3-319-31858-5 (cit. on p. 6).

[419] L. Lazzeroni and A. Owen. “Plaid models for gene expression data”. In: *Statistica sinica* (2002), pp. 61–86 (cit. on p. 233).

[420] Y. LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998) (cit. on pp. 351, 352).

[421] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on pp. 253, 255).

[422] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48 (cit. on p. 252).

[423] D. D. Lee and H. S. Seung. “Learning the parts of objects by non-negative matrix factorization”. In: *Nature* 401.6755 (1999), pp. 788–791 (cit. on p. 229).

[424] H. G. Lee and N. Chang. “Powering the IoT: Storage-less and converter-less energy harvesting”. In: *Procs. of the Asia and South Pacific Design Automation Conference 2015*. Jan. 2015, pp. 124–129 (cit. on p. 53).

[425] V. Leis, P. Boncz, A. Kemper, and T. Neumann. “Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age”. In: *Procs. of the SIGMOD Int. Conference on the Management of Data 2014*. ACM, 2014, pp. 743–754 (cit. on p. 386).

[426] M. Lewin, D. Livnat, and U. Zwick. “Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems”. In: *Procs. of the Int. Conference on Integer Programming and Combinatorial Optimization 2002*. Ed. by W. J. Cook and A. S. Schulz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 67–82 (cit. on pp. 145, 147).

[427] H. Li and Z. Lin. “Accelerated proximal gradient methods for nonconvex programming”. In: *Advances in Neural Information Processing Systems 28: Procs. of the 2015 Conference*. 2015, pp. 379–387 (cit. on p. 247).

[428] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang. “A High Performance FPGA-based Accelerator for Large-Scale Convolutional Neural Networks”. In: *Procs. of the Int. Conference on Field-Programmable Logic and Applications 2016*. 2016 (cit. on p. 254).

[429] H. L. Li Yi and Nguyen and D. P. Woodruff. “Turnstile streaming algorithms might as well be linear sketches”. In: *Procs. of the Symposium on Theory of Computing 2014*. 2014, pp. 174–183 (cit. on p. 89).

[430] J. Li, J.-J. Chen, K. Agrawal, C. Lu, C. D. Gill, and A. Saifullah. “Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2014*. 2014, pp. 85–96 (cit. on p. 367).

[431] J. Li, T. Luong, and D. Jurafsky. “A Hierarchical Neural Autoencoder for Paragraphs and Documents”. In: *Procs. of the Meeting of the Association for Computational Linguistics and the Int. Joint Conference on Natural Language Processing 2015*. Vol. 1. 2015, pp. 1106–1115 (cit. on p. 251).

[432] M. Li, G. L. Miller, and R. Peng. “Iterative Row Sampling”. In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2013*. 2013, pp. 127–136 (cit. on p. 87).

[433] S. Li and O. Svensson. “Approximating *k*-Median via Pseudo-Approximation”. In: *SIAM Journal on Computing* 45.2 (2016). Previously appeared in the Procs. of the Symposium on Theory of Computing 2013, pp. 530–547. DOI: https://doi.org/10.1137/130938645 (cit. on p. 201).

[434] T. Li. “A general model for clustering binary data”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2005*. ACM. 2005, pp. 188–197 (cit. on p. 232).

[435] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. “Gated Graph Sequence Neural Networks”. In: *Procs. of the Int. Conference on Learning Representation 2016*. 2016 (cit. on p. 135).

[436] Y. Li, T. Li, R. A. Patel, X.-D. Yang, and X. Zhou. “Self-powered gesture recognition with ambient light”. In: *Procs. of the ACM Symposium on User Interface Software and Technology 2018*. 2018, pp. 595–608 (cit. on p. 61).

[437] Y. Li and A. Pedram. “CATERPILLAR: Coarse Grain Reconfigurable Architecture for Accelerating the Training of Deep Neural Networks”. In: *Procs. of the Int. Conference on Application-specific Systems, Architectures and Processors 2017*. IEEE. 2017, pp. 1–10 (cit. on pp. 253, 255).

[438] F. Liang, C. Liu, and N. Wang. “A robust sequential Bayesian method for identification of differentially expressed genes”. In: *Statistica Sinica* (2007), pp. 571–597 (cit. on p. 95).

[439] P. Libuschewski. “Exploration of Cyber-Physical Systems for GPGPU Computer Vision-Based Detection of Biological Viruses”. PhD thesis. Dortmund, Germany: TU Dortmund University, 2017. DOI: http://dx.doi.org/10.17877/DE290R-17952 (cit. on p. 339). **SFB876-B2**

[440] T. Liebig, Z. Xu, M. May, and S. Wrobel. “Pedestrian Quantity Estimation with Trajectory Patterns”. In: *Procs. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2012*. Springer, 2012, pp. 629–643. URL: http://link.springer.com/chapter/10.1007%5C%2F978-3-642-33486-3_40 (cit. on p. 102).

[441] M. Lippi, M. Bertini, and P. Frasconi. “Collective Traffic Forecasting”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag. Vol. 6322. LNCS. Springer, 2010, pp. 259–273 (cit. on p. 101).

[442] C. Liu and J.-J. Chen. “Bursty-interference analysis techniques for analyzing complex real-time task models”. In: *Procs. of the IEEE Real-Time Systems Symposium 2014*. IEEE. 2014, pp. 173–183 (cit. on p. 363).

[443] F. T. Liu, K. M. Ting, and Z. Zhou. “Isolation Forest”. In: *Procs. of the IEEE Int. Conference on Data Mining 2008*. Dec. 2008, pp. 413–422 (cit. on p. 81).

[444] S. P. Lloyd. “Least squares quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137 (cit. on p. 231).

[445] A. Lochmann, F. Bruckner, and O. Spinczyk. “Reproducible Load Tests for Android Systems with Trace-based Benchmarks”. In: *Procs. of the ACM/SPEC on Int. Conference on Performance Engineering Companion 2017*. ICPE ’17 Companion. New York, NY, USA: ACM Press, 2017, pp. 73–76 (cit. on p. 32). **SFB876-A1**

[446] A. Lochmann, H. Schirmeier, H. Borghorst, and O. Spinczyk. “LockDoc: Trace-Based Analysis of Locking in the Linux Kernel”. In: *Procs. of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2019*. 2019 (cit. on p. 33). **SFB876-A1**

[447] B. Long, Z. (Zhang, and P. S. Yu. “Co-Clustering by Block Value Decomposition”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2005*. KDD ’05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 635–640 (cit. on p. 238).

[448] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. “The irace package: Iterated racing for automatic algorithm configuration”. In: *Operations Research Perspectives* 3 (2016), pp. 43–58. DOI: https://doi.org/10.1016/j.orp.2016.09.002 (cit. on p. 286).

[449] Z. Lu, X. Wu, X. Zhu, and J. Bongard. “Ensemble pruning via individual contribution ordering”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2010* (2010), pp. 871–880 (cit. on p. 340).

[450] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. “Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees”. In: *Procs. of the Int. ACM SIGIR Conference on Research and Development in Information Retrieval 2015*. ACM. 2015, pp. 73–82 (cit. on pp. 340, 341).

[451] C. Lucchese, S. Orlando, and R. Perego. “A Unifying Framework for Mining Approximate Top-*k* Binary Patterns”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.12 (2014), pp. 2900–2913 (cit. on p. 239).

[452] C. Lucchese, R. Perego, F. M. Nardini, N. Tonellotto, S. Orlando, and R. Venturini. “Exploiting CPU SIMD extensions to speed-up document scoring with tree ensembles”. In: *Procs. of the Int. ACM SIGIR Conference on Research and Development in Information Retrieval 2016*. 2016 (cit. on p. 340).

[453] M. Lucic, O. Bachem, and A. Krause. “Strong Coresets for Hard and Soft Bregman Clustering with Applications to Exponential Family Mixtures”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2016*. 2016, pp. 1–9 (cit. on p. 87).

[454] Y. Lv, Y. Duan, W. Kang, Z. Li, F.-Y. Wang, et al. “Traffic flow prediction with big data: A deep learning approach.” In: *IEEE Trans. Intelligent Transportation Systems* 16.2 (2015), pp. 865–873 (cit. on p. 260).

[455] Y. Ma and J. Tang. *Deep Learning on Graphs*. Cambridge University Press, 2020 (cit. on pp. 9, 130, 136).

[456] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. “Learning Word Vectors for Sentiment Analysis”. In: *Procs. of the Meeting of the Association for Computational Linguistics: Human Language Technologies 2011*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015 (cit. on pp. 351, 352).

[457] Y. Maeda, H. Hirano, and Y. Kanata. “A learning rule of neural networks via simultaneous perturbation and its hardware implementation”. In: *Neural Networks* 8.2 (1995), pp. 251–259 (cit. on p. 254).

[458] Y. Maeda and T. Tada. “FPGA implementation of a pulse density neural network with learning ability using simultaneous perturbation”. In: *IEEE Transactions on Neural Networks* 14.3 (2003), pp. 688–695 (cit. on p. 254).

[459] S. Mahajan and H. Ramesh. “Derandomizing Approximation Algorithms Based on Semidefinite Programming”. In: *SIAM Journal on Computing* 28.5 (1999), pp. 1641–1663 (cit. on p. 145).

[460] M. Mahdavi, R. Zanibbi, H. Mouch, and C. Viard-gaudin. “ICDAR 2019 CROHME + TFD : Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula

Detection”. In: *Procs. of the IAPR Int. Conference on Document Analysis and Recognition 2019*. 2019 (cit. on p. 162).

[461] M. W. Mahoney. “Randomized Algorithms for Matrices and Data”. In: *Foundations and Trends in Machine Learning* 3.2 (2011), pp. 123–224. DOI: https : // doi . org / 10 . 1561 / 2200000035 (cit. on p. 93).

[462] T. Mai, C. Musco, and A. Rao. “Coresets for Classification - Simplified and Strengthened”. In: *Advances in Neural Information Processing Systems 34: Procs. of the 2021 Conference*. 2021, pp. 11643–11654 (cit. on p. 91).

[463] C. Manning, P. Raghavan, and H. Schütze. “Introduction to information retrieval”. In: *Natural Language Engineering* 16.1 (2010), pp. 100–103 (cit. on p. 273).

[464] B. Mansouri, D. W. Oard, C. L. Giles, and R. Zanibbi. “Tangent-CFT : An Embedding Model for Mathematical Formulas”. In: *Procs. of the ACM SIGIR Int. Conference on Theory of Information Retrieval 2019*. 2019 (cit. on pp. 162, 174, 175).

[465] F. E. Maranzana. “On the Location of Supply Points to Minimize Transportation Costs”. In: *IBM Systems Journal* 2.2 (1963), pp. 129–135 (cit. on p. 184).

[466] J. Marín, D. Vázquez, A. M. López, J. Amores, and B. Leibe. “Random Forests of Local Experts for Pedestrian Detection”. In: *Procs. of the IEEE Int. Conference on Computer Vision 2013*. 2013, pp. 2592–2599 (cit. on p. 339).

[467] S. F. Marinosson, R. Chrobok, A. Pottmeier, J. Wahle, and M. Schreckenberg. “Simulation Framework for the Autobahn Traffic in North Rhine-Westphalia”. In: *Cellular Automata – Procs. of the Int. Conf. on Cellular Automata for Research and Industry 2002*. Springer, 2002, pp. 2977–2980 (cit. on p. 101).

[468] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. “Provably Powerful Graph Networks”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019 (cit. on pp. 127, 132).

[469] M. A. Maruf and A. Azim. “Extending resources for avoiding overloads of mixed-criticality tasks in cyber-physical systems”. In: *IET Cyber-Physical Systems: Theory Applications* 5.1 (2020), pp. 60–70 (cit. on p. 363).

[470] P. Marwedel. *Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. 4th ed. Springer, 2021. URL: https://link.springer.com/book/10.1007/978-3-030-60910-8 (cit. on p. XI). **SFB876-A3**

[471] M. Masoudinejad. “Modeling Energy Supply Unit of Ultra-Low Power Devices with Indoor Photovoltaic Harvesting”. PhD Thesis. TU Dortmund University, 2020. URL: https://eldorado.tu-dortmund.de/handle/2003/39765 (cit. on p. 35). **SFB876-A4**

[472] M. Masoudinejad, A. K. Ramachandran Venkatapathy, J. Emmerich, and A. Riesner. “Procs. of the Int. Conference on Sensor Systems and Software 2016”. In: *Sensor Systems and Software*. Springer, 2016. Chap. 4, pp. 41–52 (cit. on p. 35). **SFB876-A4**

[473] M. Masoudinejad, K. A. Venkatapathy Ramachandran, D. Tondorf, D. Heinrich, R. Falkenberg, and M. Buschhoff. “Machine Learning Based Indoor Localisation using Environmental data in PhyNetLab Warehouse”. In: *Procs. of the SysTech European Conference on Smart Objects, Systems and Technologies 2018*. June 2018 (cit. on p. 45). **SFB876-A4**

[474] B. Matérn. “Spatial Variation: Stochastic Models and their Application to some Problems in Forest Surveys and other Sampling Investigations”. In: *Meddelanden fran Statens Skogsforskningsinstitut* 49.5 (1960), p. 144 (cit. on p. 294).

[475] V. Maurizio. “Double k-means clustering for simultaneous classification of objects and variables”. In: *Advances in Classification and Data Analysis*. Springer, 2001, pp. 43–52 (cit. on p. 232).

[476] D. Maxim and L. Cucu-Grosjean. “Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters”. In: *Procs. of the IEEE Real-Time Systems Symposium 2013*. 2013, pp. 224–235 (cit. on pp. 363, 376).

[477] M. May, D. Hecker, C. Körner, S. Scheider, and D. Schulz. “A Vector-Geometry Based Spatial kNN-Algorithm for Traffic Frequency Predictions”. In: *Procs. of the IEEE Int. Conference on Data Mining 2008*. IEEE Computer Society, 2008, pp. 442–447 (cit. on p. 102).

[478] J. McAuley and J. Leskovec. “Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text”. In: *Procs. of the ACM Conference on Recommender Systems 2013*. Vol. 7. RecSys ’13. New York, NY, USA: ACM, 2013, pp. 165–172 (cit. on pp. 280, 282).

[479] P. McCullagh and J. Nelder. *Generalized linear models*. 2nd ed. Chapman and Hall/CRC, Boca Raton, 1989 (cit. on pp. 90, 92).

[480] B. D. McKay and A. Piperno. “Practical graph isomorphism, II”. In: *Journal of Symbolic Computation* 60 (2014), pp. 94–112. DOI: https://doi.org/10.1016/j.jsc.2013.09.003 (cit. on p. 117).

[481] M. D. McKay, R. J. Beckman, and W. J. Conover. “A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code”. In: *Technometrics* 42.1 (2000), pp. 55–61 (cit. on pp. 294, 301).

[482] L. L. McQuitty. “Elementary Linkage Analysis for Isolating Orthogonal and Oblique Types and Typal Relevancies”. In: *Educational and Psychological Measurement* 17.2 (1957), pp. 207–229. DOI: https://doi.org/10.1177/001316445701700204 (cit. on pp. 216, 217).

[483] N. Megiddo and K. J. Supowit. “On the Complexity of Some Common Geometric Location Problems”. In: *SIAM Journal on Computing* 13.1 (1984), pp. 182–196 (cit. on p. 200).

[484] N. Meinshausen and P. Bühlmann. “High-dimensional graphs and variable selection with the Lasso”. In: *Annals of Statistics* 34.3 (2006), pp. 1436–1462 (cit. on p. 105).

[485] S. Meintrup, A. Munteanu, and D. Rohde. “Random projections and sampling algorithms for clustering of high-dimensional polygonal curves”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019, pp. 12807–12817. URL: https://papers.nips.cc/paper/9443-random-projections-and-sampling-algorithms-for-clustering-of-high-dimensional-polygonal-curves (cit. on pp. 75, 87, 203, 204). **SFB876-A2, SFB876-C4**

[486] A. K. Menon, A. S. Rawat, S. Reddi, and S. Kumar. “Multilabel reductions: what is my loss optimising?” In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. Vol. 32. 2019 (cit. on p. 274).

[487] P. Menon et al. “Relaxed Operator Fusion for In-Memory Databases: Making Compilation, Vectorization, and Prefetching Work Together At Last”. In: *Procs. of the VLDB Endowment* 11.1 (2017) (cit. on p. 386).

[488] S. G. Merchant and G. D. Peterson. “Evolvable block-based neural network design for applications in dynamic environments”. In: *VLSI Design* 2010 (2010), p. 4 (cit. on pp. 253, 254).

[489] D. Merrill. *CUB v1.7.0: CUDA Unbound, a Library of Warp-Wide, Block-Wide, and Device-Wide GPU Parallel Primitives*. 2017 (cit. on p. 393).

[490] R. Michalski, J. Carbonell, and T. Mitchell. *Machine Learning: An artificial intelligence approach*. Kaufman Publishers Inc., 1983 (cit. on p. XI).

[491] P. Miettinen, T. Mielikainen, A. Gionis, G. Das, and H. Mannila. “The discrete basis problem”. In: *Knowledge and Data Engineering, IEEE Transactions on* 20.10 (2008), pp. 1348–1362 (cit. on pp. 233, 235).

[492] P. Miettinen and J. Vreeken. “MDL4BMF: Minimum description length for Boolean matrix factorization”. In: *ACM Transactions on Knowledge Discovery from Data 2014* 8.4 (2014), p. 18 (cit. on p. 239).

[493] T. Mikolov, K. Chen, G. Corrado, and J. Dean. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv: Computing Research Repository* (2013). DOI: arXiv:1301.3781 (cit. on p. 167).

[494] B. Mirkin, P. Arabie, and L. J. Hubert. “Additive two-mode clustering: the error-variance approach revisited”. In: *Journal of classification* 12.2 (1995), pp. 243–263 (cit. on p. 231).

[495] B. Mirzasoleiman, S. Jegelka, and A. Krause. “Streaming Non-Monotone Submodular Maximization: Personalized Video Summarization on the Fly”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2018*. 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17014/15832 (cit. on p. 10).

[496] I. Misra and L. v. d. Maaten. “Self-supervised learning of pretext-invariant representations”. In: *Procs. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2020*. 2020, pp. 6707–6717 (cit. on p. 175).

[497] J. Misra and I. Saha. “Artificial neural networks in hardware: A survey of two decades of progress”. In: *Neurocomputing* 74.1 (2010), pp. 239–255 (cit. on p. 253).

[498] S. Mitra and T. Acharya. “Gesture recognition: A survey”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.3 (2007), pp. 311–324 (cit. on p. 60).

[499] S. Miyamoto, Y. Kaizu, and Y. Endo. “Hierarchical and Non-Hierarchical Medoid Clustering Using Asymmetric Similarity Measures”. In: *Procs. of the Int. Symposium on Soft Computing and Intelligent Systems 2016*. 2016, pp. 400–403 (cit. on p. 216).

[500] P. Moerland and E. Fiesler. *Neural network adaptations to hardware implementations*. Tech. rep. The Idiap Research Institute, Switzerland, 1997 (cit. on p. 253).

[501] A. Molina, A. Munteanu, and K. Kersting. “Core Dependency Networks”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2018*. 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16847 (cit. on pp. 87, 89, 92, 93). **SFB876-B4, SFB876-C4**

[502] S.-W. Moon and S.-G. Kong. “Block-based neural networks”. In: *IEEE Transactions on Neural Networks* 12.2 (2001), pp. 307–317 (cit. on p. 254).

[503] F. Morandat, B. Hill, L. Osvald, and J. Vitek. “Evaluating the design of the R language: objects and functions for data analysis”. In: *Procs. of the European Conference on Object-Oriented Programming 2012*. Springer-Verlag, 2012, pp. 104–131 (cit. on p. 306).

[504] C. Morris, G. Rattan, and P. Mutzel. “Weisfeiler and Leman Go Sparse: Towards Scalable Higher-Order Graph Embeddings”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020 (cit. on pp. 123, 127, 128, 132).

[505] C. Morris, M. Fey, and N. M. Kriege. “The Power of the Weisfeiler-Leman Algorithm for Machine Learning with Graphs”. In: *Procs. of the Int. Joint Conferences on Artifical Intelligence 2021*. 2021 (cit. on p. 128). **SFB876-A6**

[506] C. Morris, K. Kersting, and P. Mutzel. “Glocalized Weisfeiler-Lehman Graph Kernels: Global-Local Feature Maps of Graphs”. In: *Procs. of the IEEE Int. Conference on Data Mining 2017*. 2017, pp. 327–336 (cit. on p. 123). **SFB876-A6**

[507] C. Morris, N. Kriege, K. Kersting, and P. Mutzel. “Faster Kernels for Graphs with Continuous Attributes via Hashing”. In: *Procs. of the IEEE Int. Conference on Data Mining 2016*. 2016, pp. 1095–1100 (cit. on p. 124). **SFB876-A6**

[508] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. “TUDataset: A collection of benchmark datasets for learning with graphs”. In: *Procs. of the Int. Conference on Machine Learning Workshop on Graph Representation Learning and Beyond 2020*. 2020. URL: www.graphlearning.io (cit. on pp. 124, 128). **SFB876-A6**

[509] C. Morris et al. “Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2019*. 2019. DOI: arXiv:1810.02244 (cit. on pp. 117, 127, 130, 132, 137, 139). **SFB876-A6**

[510] S. Muecke, N. Piatkowski, and K. Morik. “Hardware Accelerated Learning at the Edge”. In: *Procs. of Decentralized Machine Learning at the Edge 2019*. Ed. by M. Kamp, D. Paurat, and Y. Krishnamurthy. Springer, 2019. URL: https://dmle.iais.fraunhofer.de/papers/muecke2019hardware.pdf (cit. on p. 7).

[511] A. Munteanu. “On large-scale probabilistic and statistical data analysis”. PhD thesis. TU Dortmund University, 2018. DOI: http://dx.doi.org/10.17877/DE290R-19112 (cit. on pp. 90, 95).

[512] A. Munteanu, A. Nayebi, and M. Poloczek. “A Framework for Bayesian Optimization in Embedded Subspaces”. In: *Procs. of the Int. Conference on Machine Learning 2019*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Procs. of Machine Learning Research. Long Beach, California, USA: PMLR, June 2019, pp. 4752–4761. URL: http://proceedings.mlr.press/v97/nayebi19a.html (cit. on pp. 90, 96). **SFB876-C4**

[513] A. Munteanu, S. Omlor, and C. Peters. “p-Generalized Probit Regression and Scalable Maximum Likelihood Estimation via Sketching and Coresets”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2022*. 2022 (cit. on pp. 89, 90, 95). **SFB876-C4**

[514] A. Munteanu, S. Omlor, Z. Song, and D. P. Woodruff. “Bounding the Width of Neural Networks via Coupled Initialization - A Worst Case Analysis”. In: *Procs. of the Int. Conference on Machine Learning 2022*. 2022 (cit. on p. 90). **SFB876-C4**

[515] A. Munteanu, S. Omlor, and D. P. Woodruff. “Oblivious Sketching for Logistic Regression”. In: *Procs. of the Int. Conference on Machine Learning 2021*. 2021. URL: https://proceedings.mlr.press/v139/munteanu21a.html (cit. on pp. 89–92). **SFB876-C4**

[516] A. Munteanu and C. Schwiegelshohn. “Coresets - Methods and History: A Theoreticians Design Pattern for Approximation and Streaming Algorithms”. In: *KI - Künstliche Intelligenz* 32.1 (2018). KI special issue on ’Algorithmic Challenges and Opportunities of Big Data’, pp. 37–53. DOI: https://doi.org/10.1007/s13218-017-0519-3 (cit. on pp. 10, 86, 87, 200). **SFB876-A2, SFB876-C4**

[517] A. Munteanu, C. Schwiegelshohn, C. Sohler, and D. P. Woodruff. “On Coresets for Logistic Regression”. In: *Advances in Neural Information Processing Systems 31: Procs. of the 2018 Conference*. 2018. URL: http://papers.nips.cc/paper/7891-on-coresets-for-logistic-regression (cit. on pp. 75, 87, 89–91). **SFB876-A2, SFB876-C4**

[518] A. Munteanu and M. Wornowizki. “Correcting statistical models via empirical distribution functions”. In: *Computational Statistics* 31.2 (June 2016), pp. 465–495. URL: http://doi.org/10.1007/s00180-015-0607-5 (cit. on p. 95). **SFB876-C3, SFB876-C4**

[519] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. “Relational Pooling for Graph Representations”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019 (cit. on p. 132).

[520] F. Murtagh. “A Survey of Recent Advances in Hierarchical Clustering Algorithms”. In: *The Computer Journal* 26.4 (Nov. 1983), pp. 354–359. DOI: https://doi.org/10.1093/comjnl/26.4.354 (cit. on p. 222).

[521] F. Murtagh and P. Legendre. “Ward’s Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward’s Criterion?” In: *Journal of Classification* 31.3 (2014), pp. 274–295. DOI: https://doi.org/10.1007/s00357-014-9161-z (cit. on p. 216).

[522] S. Murugan, K. P. Lakshmi, J. Sundar, and K. MathiVathani. “Design and Implementation of Multilayer Perceptron with On-chip Learning in Virtex-E”. In: *AASRI Procedia* 6 (2014), pp. 82–88 (cit. on pp. 253, 262).

[523] S. Muthukrishnan. “Data Streams: Algorithms and Applications”. In: *Foundations and Trends in Theoretical Computer Science* 1.2 (2005) (cit. on pp. 71, 89).

[524] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford, and J. R. Smith. “WISPCam : A Battery-Free RFID Camera”. In: *Procs. of the IEEE Int. Conference on RFID 2015*. 2015 (cit. on p. 53).

[525] A. Nath and E. Taylor. "*k*-Median Clustering Under Discrete Fréchet and Hausdorff Distances". In: *Procs. of the Int. Symposium on Computational Geometry 2020*. Ed. by S. Cabello and D. Z. Chen. Vol. 164. Leibniz Int. Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 58:1–58:15 (cit. on pp. 203, 204).

[526] O. Neugebauer. *Energy Measurement made Simple on Embedded Systems*. Technical Report No. 2 for Collaborative Research Center SFB 876 - Graduate School. 2017. URL: https://sfb876.tu-dortmund.de/auto?self=%5C%24egw1pio6bk (cit. on p. 300).

[527] J. von Neumann. "First Draft of a Report on the EDVAC". In: 1945 (cit. on p. 2).

[528] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. "Propagation Kernels: Efficient Graph Kernels from Propagated Information". In: *Machine Learning* 102.2 (Feb. 2016), pp. 209–245 (cit. on p. 125). **SFB876-A6**

[529] T. Neumann. "Efficiently Compiling Efficient Query Plans for Modern Hardware". In: *Procs. of the VLDB Endowment* 4.9 (2011), pp. 539–550. URL: http://www.vldb.org/pvldb/vol4/p539-neumann.pdf (cit. on pp. 263, 385, 386, 395).

[530] J. Newling and F. Fleuret. "A Sub-Quadratic Exact Medoid Algorithm". In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2017*. 2017, pp. 185–193. URL: http://proceedings.mlr.press/v54/newling17a.html (cit. on p. 184).

[531] A. Y. Ng and M. I. Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes". In: *Advances in Neural Information Processing Systems 14: Procs. of the 2001 Conference*. Ed. by S. B. T. G. Dietterich and Z. Ghahramani. Vol. 14. Cambridge, MA: MIT Press., 2002, pp. 841–848 (cit. on p. 102).

[532] D. Nguyen, N. Ho, and I. Chang. "St-DRC: Stretchable DRAM Refresh Controller with No Parity-overhead Error Correction Scheme for Energy-efficient DNNs". In: *Procs. of the Design Automation Conference 2019*. 2019, pp. 1–6 (cit. on p. 327).

[533] H. Nguyen and T. Maehara. "Graph Homomorphism Convolution". In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020, pp. 7306–7316 (cit. on p. 125).

[534] K. Ni, A. Gupta, O. Prakash, S. Thomann, X. S. Hu, and H. Amrouch. "Impact of Extrinsic Variation Sources on the Device-to-Device Variation in Ferroelectric FET". In: *Procs. of the IEEE Int. Reliability Physics Symposium 2020*. 2020 (cit. on p. 328).

[535] K. Ni et al. "Ferroelectric ternary content-addressable memory for one-shot learning". In: *Nature Electronics* 2.11 (2019), pp. 521–529 (cit. on p. 4).

[536] F. Nie, X. Wang, C. Deng, and H. Huang. "Learning a structured optimal bipartite graph for co-clustering". In: *Advances in Neural Information Processing Systems 30: Procs. of the 2017 Conference*. 2017, pp. 4129–4138 (cit. on p. 232).

[537] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis. "Matching Node Embeddings for Graph Similarity". In: *Procs. of the AAAI Conference on Artificial Intelligence 2017*. 2017, pp. 2429–2435 (cit. on p. 125).

[538] J. Nocedal and S. J. Wright. *Numerical Optimization*. Second. Springer Series in Operations Research and Financial Engineering. Springer, 2006. URL: https://books.google.de/books?id=epc5fX0lqRIC (cit. on p. 105).

[539] N. Noorshams and M. Wainwright. "Stochastic belief propagation: Low-complexity message-passing with guarantees". In: *Procs. of the Allerton Conference on Communication, Control, and Computing 2011*. 2011, pp. 269–276 (cit. on p. 407).

[540] A. Norouzi-Fard, J. Tarnawski, S. Mitrovic, A. Zandieh, A. Mousavifar, and O. Svensson. "Beyond 1/2-Approximation for Submodular Maximization on Massive Data Streams". In: *Procs. of the Int. Conference on Machine Learning 2018*. July 2018, pp. 3829–3838. URL: http://proceedings.mlr.press/v80/norouzi-fard18a.html (cit. on pp. 76, 77).

[541] E. Nurvitadhi et al. “Can FPGAs beat GPUs in accelerating next-generation deep neural networks?” In: *Procs. of the ACM/SIGDA Int. Symposium on Field-Programmable Gate Arrays 2017*. ACM. 2017, pp. 5–14 (cit. on p. 253).

[542] NVIDIA Corp. *CUDA Toolkit Documentation v8.0, CUDA C Programming Guide*. 2016 (cit. on p. 408).

[543] P. E. O’Neil, E. J. O’Neil, and X. Chen. “The star schema benchmark (SSB)”. In: *Polymers for Advanced Technologies* 200.0 (2007), p. 50 (cit. on p. 381).

[544] L. Oettershagen, N. M. Kriege, C. Morris, and P. Mutzel. “Temporal Graph Kernels for Classifying Dissemination Processes”. In: *Procs. of the SIAM Int. Conference on Data Mining 2020*. 2020 (cit. on p. 128). **SFB876-A6**

[545] OmniSci Incorporated. *OmniSciDB*. `https://www.omnisci.com/`. 2019. URL: https://www.omnisci.com/platform/omniscidb (cit. on p. 270).

[546] A. Omondi and J. Rajapakse. *FPGA implementations of neural networks*. Springer, 2006 (cit. on p. 253).

[547] A. V. D. Oord, Y. Li, and O. Vinyals. *Representation Learning with Contrastive Predictive Coding*. Tech. rep. 2018. DOI: arXiv:1807.03748 (cit. on p. 175).

[548] P. Paatero and U. Tapper. “Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values”. In: *Environmetrics* 5.2 (1994), pp. 111–126 (cit. on p. 229).

[549] Y. Pan et al. “A Multilevel Cell STT-MRAM-Based Computing In-Memory Accelerator for Binary Convolutional Neural Network”. In: *IEEE Transactions on Magnetics* 54 (2018), pp. 1–5 (cit. on p. 327).

[550] C. H. Papadimitriou. “Worst-case and probabilistic analysis of a geometric location problem”. In: *SIAM Journal on Computing* 10.3 (1981), pp. 542–557 (cit. on p. 200).

[551] R. Parada and J. Melia-Segui. “Gesture detection using passive RFID tags to enable people-centric IoT applications”. In: *IEEE Communications Magazine* 55.2 (2017), pp. 56–61 (cit. on p. 61).

[552] N. Parikh and S. Boyd. “Proximal Algorithms”. In: *Foundations and Trends in Optimization* 1.3 (Jan. 2014), pp. 127–239. DOI: http://dx.doi.org/10.1561/2400000003 (cit. on p. 234).

[553] H. Park and C. Jun. “A simple and fast algorithm for K-medoids clustering”. In: *Expert Systems With Applications* 36.2 (2009), pp. 3336–3341 (cit. on p. 184).

[554] T. Park and G. Casella. “The Bayesian Lasso”. In: *Journal of the American Statistical Association* 103 (2008), pp. 681–686 (cit. on p. 95).

[555] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019 (cit. on p. 134).

[556] D. Patterson et al. *Carbon Emissions and Large Neural Network Training*. arXiv. 2021. DOI: 10.48550/arXiv.2104.10350 (cit. on p. 6).

[557] J. Paul, J. He, and B. He. “GPL: A GPU-Based Pipelined Query Processing Engine”. In: *Procs. of the ACM SIGMOD Int. Conference on Management of Data 2016*. 2016, pp. 1935–1950 (cit. on p. 386).

[558] K. Paul and S. Rajopadhye. “Back-propagation algorithm achieving 5 GOPs on the Virtex-E”. In: *FPGA Implementations of Neural Networks*. Springer, 2006, pp. 137–165 (cit. on pp. 253, 262).

[559] N. D. Pearce and M. P. Wand. “Penalized Splines and Reproducing Kernel Methods”. In: *The American Statistician* 60.3 (2006), pp. 233–240 (cit. on p. 105).

[560] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988 (cit. on pp. 105, 407).

[561] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 350, 351).

[562] L. Pfahler. “Some Representation Learning Tasks and the Inspection of Their Models”. PhD thesis. TU Dortmund University, 2022 (cit. on p. 160). **SFB876-A1**

[563] L. Pfahler and K. Morik. “Semantic Search in Millions of Equations”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2020*. ACM, 2020. DOI: https://dl.acm.org/doi/pdf/10.1145/3394486.3403056 (cit. on pp. 160, 162, 174, 176). **SFB876-A1**

[564] L. Pfahler, J. Schill, and K. Morik. “The Search for Equations - Learning to Identify Similarities between Mathematical Expressions”. In: *Procs. of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases 2019*. Springer, 2019. URL: https://link.springer.com/chapter/10.1007/978-3-030-46133-1_42 (cit. on pp. 162, 167, 168, 170–172, 174). **SFB876-A1**

[565] J. M. Phillips. “Coresets and sketches”. In: *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, 2017 (cit. on pp. 85, 87).

[566] N. Piatkowski, S. Lee, and K. Morik. “Spatio-temporal random fields: compressible representation and distributed estimation”. In: *Machine Learning* 93.1 (2013), pp. 115–139. URL: http://link.springer.com/article/10.1007%5C%2Fs10994-013-5399-7 (cit. on pp. 101, 112). **SFB876-A1**

[567] N. Piatkowski. “Exponential Families on Resource-Constrained Systems”. PhD thesis. Dortmund: TU Dortmund University, 2018. URL: https://eldorado.tu-dortmund.de/handle/2003/36877 (cit. on pp. 9, 101, 111, 112, 408).

[568] N. Piatkowski. “Hyper-Parameter-Free Generative Modelling with Deep Boltzmann Trees”. In: *Procs. of the European Conference on Machine Learning 2019*. Springer, 2019 (cit. on p. 100).

[569] N. Piatkowski. “iST-MRF: Interactive Spatio-Temporal Probabilistic Models for Sensor Networks”. In: *Procs. of the Int. Workshop at ECML PKDD 2012 on Instant Interactive Data Mining 2012*. 2012 (cit. on p. 105). **SFB876-A1**

[570] N. Piatkowski, S. Lee, and K. Morik. “Integer undirected graphical models for resource-constrained systems”. In: *Neurocomputing* 173.1 (Jan. 2016), pp. 9–23. URL: http://www.sciencedirect.com/science/article/pii/S0925231215010449 (cit. on p. 9). **SFB876-A1, SFB876-C1**

[571] N. Piatkowski, S. Lee, and K. Morik. “Spatio-Temporal Models For Sustainability”. In: *Procs. of the SustKDD Workshop within ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2012*. Ed. by M. Marwah, N. Ramakrishnan, M. Berges, and Z. Kolter. ACM, 2012. URL: http://wan.poly.edu/KDD2012/forms/workshop/SustKDD12/doc/SustKDD12_1.pdf (cit. on p. 105). **SFB876-A1, SFB876-C1**

[572] N. Piatkowski and K. Morik. “Fast Stochastic Quadrature for Approximate Maximum-Likelihood Estimation”. In: *Procs. of the Conference on Uncertainty in Artificial Intelligence 2018*. 2018 (cit. on pp. 105, 407). **SFB876-A1**

[573] N. Piatkowski and K. Morik. “Stochastic Discrete Clenshaw-Curtis Quadrature”. In: *Procs. of the Int. Conference on Machine Learning 2016*. JMLR: W&CP. JMLR.org, June 2016. URL: http://jmlr.org/proceedings/papers/v48/piatkowski16.html (cit. on pp. 105, 407). **SFB876-A1**

[574] N. Piatkowski, L. Sangkyun, and K. Morik. “The Integer Approximation of Undirected Graphical Models”. In: *Procs. of the Int. Conference on Pattern Recognition Applications and Methods 2014*. Ed. by M. De Marsico, A. Tabbone, and A. Fred. SciTePress, 2014, pp. 296–304. URL: http://www-ai.cs.uni-dortmund.de/PublicPublicationFiles/piatkowski_etal_2014a.pdf (cit. on p. 408). **SFB876-A1, SFB876-C1**

[575] N. Piatkowski and F. Schnitzler. “Compressible Reparametrization of Time-Variant Linear Dynamical Systems”. In: *Solving Large Scale Learning Tasks. Challenges and Algorithms -*

Essays Dedicated to Katharina Morik on the Occasion of Her 60th Birthday. 2016, pp. 234–250. DOI: http://dx.doi.org/10.1007/978-3-319-41706-6_12 (cit. on p. 11). **SFB876-A1**

[576] N. Piatkowski, J. Streicher, S. Olaf, and K. Morik. "Open Smartphone Data for Structured Mobility and Utilization Analysis in Ubiquitous Systems". In: *Mining, Modeling and Recommending Things in Social Media*. Ed. by M. Atzmueller, A. Chin, C. Scholz, and C. Trattner. Vol. 8940. Lecture Notes in Computer Science. Springer, 2014. Chap. Open Smartphone Data for Structured Mobility and Utilization Analysis in Ubiquitous Systems, pp. 116–130. URL: http://link.springer.com/chapter/10.1007%5C%2F978-3-319-14723-9_7 (cit. on p. 32). **SFB876-A1**

[577] N. Piatkowski et al. "Generative Machine Learning for Resource-Aware 5G and IoT Systems". In: *Procs. of the IEEE Int. Conference on Communications 2021*. 2021, pp. 1–6. URL: https://doi.org/10.1109/ICCWorkshops50388.2021.9473625 (cit. on p. 101).

[578] K. Pietrzak. "On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems". In: *Journal of Computer and System Sciences* 67.4 (2003), pp. 757–771. URL: http://www.sciencedirect.com/science/article/pii/S0022000003000783 (cit. on p. 205).

[579] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. 3rd. Springer Publishing Company, Incorporated, 2008 (cit. on p. 300).

[580] G. Pio, M. Ceci, D. Malerba, and D. D'Elia. "ComiRNet: a web-based system for the analysis of miRNA-gene regulatory networks". In: *BMC Bioinformatics* 16.S-9 (2015), S7 (cit. on p. 228).

[581] T. Pock and S. Sabach. "Inertial proximal alternating linearized minimization (iPALM) for nonconvex and nonsmooth problems". In: *SIAM journal on imaging sciences* 9.4 (2016), pp. 1756–1787 (cit. on p. 247).

[582] J. Podani. "New combinatorial clustering methods". In: *Numerical syntaxonomy*. Ed. by M. B. Mucina L.and Dale. Dordrecht: Springer Netherlands, 1989, pp. 61–77 (cit. on p. 219).

[583] C. N. Potts. "Analysis of a Heuristic for One Machine Sequencing with Release Dates and Delivery Times". In: *Operations Research* 28.6 (1980), pp. 1436–1441 (cit. on pp. 365, 368).

[584] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. "Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising". In: *Procs. of the World Wide Web Conference 2018*. 2018, pp. 993–1002 (cit. on p. 283).

[585] Y. Prabhu and M. Varma. "Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning". In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2014*. ACM. 2014, pp. 263–272 (cit. on p. 272).

[586] R. Prenger, B. Chen, T. Marlatt, and D. Merl. *Fast map search for compact additive tree ensembles (cate)*. Tech. rep. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2013 (cit. on p. 340).

[587] M. Qaraei, E. Schultheis, P. Gupta, and R. Babbar. "Convex Surrogates for Unbiased Loss Functions in Extreme Classification With Missing Labels". In: *Procs. of the World Wide Web Conference 2021*. 2021, pp. 3711–3720 (cit. on p. 284).

[588] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems 30: Procs. of the 2017 Conference*. 2017 (cit. on pp. 130, 132, 135).

[589] C. Qian, Y. Yu, and Z. H. Zhou. "Pareto ensemble pruning". In: *Procs. of the National Conference on Artificial Intelligence* 4 (2015), pp. 2935–2941. URL: https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/aaai15prun.pdf (cit. on p. 340).

[590] P. Raghavan. "Probabilistic construction of deterministic algorithms: Approximating packing integer programs". In: *Journal of Computer and System Sciences* 37.2 (1988), pp. 130–143 (cit. on pp. 145, 147).

[591] R. Rajkumar. “Real-time synchronization protocols for shared memory multiprocessors”. In: *Procs. of the Int. Conference on Distributed Computing Systems 1990*. 1990, pp. 116–123. DOI: http://dx.doi.org/10.1109/icdcs.1990.89257 (cit. on pp. 361, 362).

[592] R. Rajkumar, L. Sha, and J. P. Lehoczky. “Real-Time Synchronization Protocols for Multiprocessors”. In: *Procs. of the IEEE Real-Time Systems Symposium 1988*. 1988, pp. 259–269 (cit. on pp. 361, 362).

[593] Y. K. Ramadass et al. “A batteryless thermoelectric energy-harvesting interface circuit with 35mV startup voltage”. In: *IEEE J Solid-State Circuits* (2010) (cit. on p. 57).

[594] S. J. Reddi, B. Póczos, and A. J. Smola. “Communication Efficient Coresets for Empirical Loss Minimization”. In: *Procs. of the Conference on Uncertainty in Artificial Intelligence 2015*. 2015, pp. 752–761. URL: http://auai.org/uai2015/proceedings/papers/141.pdf (cit. on p. 87).

[595] A. P. Reynolds, G. Richards, B. de la Iglesia, and V. J. Rayward-Smith. “Clustering Rules: A Comparison of Partitioning and Hierarchical Clustering Algorithms”. In: *Journal of Mathematical Modelling and Algorithms* 5.4 (2006), pp. 475–504 (cit. on p. 184).

[596] J. Richter, H. Kotthaus, B. Bischl, P. Marwedel, J. Rahnenführer, and M. Lang. “Faster Model-Based Optimization through Resource-Aware Scheduling Strategies”. In: *Procs. of the Int. Conference: Learning and Intelligent Optimization 2016*. Vol. 10079. Lecture Notes in Computer Science (LNCS). Springer Int. Publishing, 2016, pp. 267–273. URL: http://link.springer.com/chapter/10.1007/978-3-319-50349-3_22 (cit. on p. 11). **SFB876-A3**

[597] B. Rieck, C. Bock, and K. M. Borgwardt. “A Persistent Weisfeiler-Lehman Procedure for Graph Classification”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019, pp. 5448–5458 (cit. on p. 125).

[598] Robotnik. *Mobile Robot RB-1 Base*. 2021. URL: https://www.robotnik.eu/mobile-robots/rb-1-base-2/ (cit. on p. 374).

[599] M. Roidl et al. “Performance Availability Evaluation of Smart Devices in Materials Handling Systems”. In: *Procs. of the IEEE ICCC Workshops on Internet of Things 2014*. Shanghai, China, Oct. 2014 (cit. on p. 35).

[600] Y. Rong, W. Huang, T. Xu, and J. Huang. “DropEdge: Towards Deep Graph Convolutional Networks on Node Classification”. In: *Procs. of the Int. Conference on Learning Representation 2020*. 2020 (cit. on pp. 130, 136).

[601] ROS. *Robot Operating System (ROS)*. 2021. URL: https://www.ros.org/ (cit. on p. 374).

[602] L. Rosasco, E. De, V. A. Caponnetto, M. Piana, and A. Verri. “Are loss functions all the same”. In: *Neural Computation* 15 (2004), p. 2004 (cit. on p. 332).

[603] K. E. Rosing, E. L. Hillsman, and H. Rosing-Vogelaar. “A Note Comparing Optimal and Heuristic Solutions To the p-Median Problem”. In: *Geographical Analysis* 11.1 (1979), pp. 86–89 (cit. on p. 184).

[604] R. A. Rossi and N. K. Ahmed. “The Network Data Repository with Interactive Graph Analytics and Visualization”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2015*. 2015. URL: http://networkrepository.com (cit. on p. 154).

[605] O. Roustant, D. Ginsbourger, and Y. Deville. “DiceKriging, DiceOptim: Two R packages for the Analysis of Computer Experiments by Kriging-based Metamodeling and Optimization”. In: *Journal of Statistical Software* 51.1 (2012), pp. 1–55 (cit. on p. 294).

[606] A. Sailer, S. Schmidhuber, M. Deubzer, M. Alfranseder, M. Mucha, and J. Mottok. “Optimizing the task allocation step for multi-core processors within AUTOSAR”. In: *Procs. of the Int. Conference on Applied Electronics 2013*. Sept. 2013, pp. 1–6 (cit. on p. 368).

[607] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio. “ROSCH:Real-Time Scheduling Framework for ROS”. In: *Procs. of the IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications 2018*. Aug. 2018, pp. 52–58 (cit. on p. 374).

[608] F. Saki, A. Sehgal, I. M. S. Panahi, and N. Kehtarnavaz. “Smartphone-based real-time classification of noise signals using subband features and random forest classifier”. In: *Procs. of the Int. Conference on Acoustics, Speech and Signal Processing 2016*. 2016, pp. 2204–2208 (cit. on p. 339).

[609] E. Sari, M. Belbahri, and V. P. Nia. “How Does Batch Normalization Help Binary Training?” In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1909.09139 (cit. on p. 329).

[610] R. E. Schapire and Y. Freund. “Boosting: Foundations and algorithms”. In: *Kybernetes* (2012) (cit. on p. 341).

[611] L. Schönberger, W.-H. Huang, G. v. d. Brüggen, K.-H. Chen, and J.-J. Chen. “Schedulability Analysis and Priority Assignment for Segmented Self-Suspending Tasks”. In: *Procs. of the IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications 2018*. Hakodate, Japan, Aug. 2018 (cit. on p. 363). **SFB876-B2**

[612] L. Schönberger et al. “Offloading Safety- and Mission-Critical Tasks via Unreliable Connections”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2020*. 2020 (cit. on p. 373). **SFB876-A1, SFB876-A3, SFB876-A4, SFB876-B4**

[613] E. Schubert. “HACAM: Hierarchical Agglomerative Clustering Around Medoids – and its Limitations”. In: *Procs. of the Lernen.Wissen.Daten.Analysen Workshops: FGWM, KDML, FGWI-BIA, and FGIR 2021*. Ed. by T. Seidl, M. Fromm, and S. Obermeier. Vol. 2993. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pp. 191–204. URL: http://ceur-ws.org/Vol-2993/paper-19.pdf (cit. on pp. 216, 219).

[614] E. Schubert and M. Gertz. “Numerically Stable Parallel Computation of (Co-)Variance”. In: *Procs. of the Int. Conference on Scientific and Statistical Database Management 2018*. SSDBM 2018 best paper award. 2018, 10:1–10:12. DOI: https://doi.org/10.1145/3221269.3223036 (cit. on p. 220).

[615] E. Schubert, H.-P. Kriegel, and A. Zimek. “The (black) art of runtime evaluation: Are we comparing algorithms or implementations?” In: *Knowledge and Information Systems* 52.2 (2017), pp. 341–3778 (cit. on p. 339).

[616] E. Schubert and P. J. Rousseeuw. “Fast and Eager k-Medoids Clustering: O(k) Runtime Improvement of the PAM, CLARA, and CLARANS Algorithms”. In: *Information Systems* 101 (2021), p. 101804. DOI: https://doi.org/10.1016/j.is.2021.101804 (cit. on pp. 184, 188, 189, 191). **SFB876-A2**

[617] E. Schubert and P. J. Rousseeuw. “Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms”. In: *Procs. of the Int. Conference on Similarity Search and Applications 2019*. 2019, pp. 171–187. DOI: https://doi.org/10.1007/978-3-030-32047-8_16 (cit. on pp. 184, 188, 189).

[618] E. Schubert and A. Zimek. “ELKI: A large open-source library for data analysis - ELKI Release 0.7.5 "Heidelberg"”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1902.03616 (cit. on pp. 192, 223).

[619] M. Seeger. *Greedy forward selection in the informative vector machine*. Tech. rep. Technical report, University of California at Berkeley, 2004 (cit. on p. 81).

[620] T. K. Sellis. “Multiple-Query Optimization”. In: *ACM Transactions on Database Systems* 13.1 (1988), pp. 23–52 (cit. on p. 31).

[621] G. Sen, G. Namata, M. Bilgic, and L. Getoor. “Collective Classification in Network Data”. In: *AI Magazine* 29 (2008) (cit. on p. 136).

[622] S. Sengupta, M. Harris, and M. Garland. *Efficient Parallel Scan Algorithms for GPUs*. Tech. Rep. NVR-2008-003. NVIDIA, Santa Clara, CA, 2008 (cit. on pp. 393, 394).

[623] L. Sha, R. Rajkumar, and J. P. Lehoczky. “Priority Inheritance Protocols: An Approach to Real-Time Synchronization”. In: *IEEE Transactions on Computers* 39.9 (1990), pp. 1175–1185. DOI: http://dx.doi.org/10.1109/12.57058 (cit. on p. 361).

[624] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. “Taking the Human Out of the Loop: A Review of Bayesian Optimization”. In: *Procs. of the IEEE* 104.1 (Jan. 2016), pp. 148–175 (cit. on p. 95).

[625] S. Shao and W. Luk. “Customised pearlmutter propagation: A hardware architecture for trust region policy optimisation”. In: *Procs. of the Int. Conference on Field Programmable Logic and Applications 2017*. IEEE. 2017, pp. 1–6 (cit. on p. 254).

[626] P. Sharma and P. Kulkarni. “Singleton: System-wide Page Deduplication in Virtual Environments”. In: *Procs. of the Int. Symposium on High-Performance Parallel and Distributed Computing 2012*. HPDC ’12. ACM, 2012, pp. 15–26 (cit. on p. 307).

[627] N. Shervashidze, P. Schweitzer, E. van Leeuwen, K. Mehlhorn, and K. Borgwardt. “Weisfeiler–Lehman Graph Kernels”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2539–2561 (cit. on pp. 117, 120, 121, 124).

[628] A. Siddiqa, A. Karim, and A. Gani. “Big Data storage technologies: A survey”. In: *Frontiers of Information Technology & Electronic Engineering* 18.8 (2017), pp. 1040–1070 (cit. on p. 89).

[629] L. Sigrist et al. “Environment and Application Testbed for Low-Power Energy Harvesting System Design”. In: *IEEE Transactions on Industrial Electronics* (2020) (cit. on p. 59).

[630] L. Sigrist, R. Ahmed, A. Gomez, and L. Thiele. “Harvesting-aware optimal communication scheme for infrastructure-less sensing”. In: *ACM Transactions on Internet of Things* (2020) (cit. on pp. 54, 57, 59).

[631] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele. “Measurement and Validation of Energy Harvesting IoT Devices”. In: *Procs. of the Design, Automation and Test in Europe Conference 2017*. 2017, pp. 1159–1164 (cit. on pp. 48, 50, 58).

[632] L. Sigrist, A. Gomez, and L. Thiele. *Long-Term Tracing of Indoor Solar Harvesting*. Aug. 2019. DOI: https://doi.org/10.5281/zenodo.3363925 (cit. on pp. 47, 49, 51).

[633] M. Simonovsky and N. Komodakis. “Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs”. In: *Procs. of the Conference on Computer Vision and Pattern Recognition 2017*. 2017 (cit. on pp. 116, 135).

[634] P. H. A. Sneath. “The Application of Computers to Taxonomy”. In: *Journal of General Microbiology* 17.1 (Aug. 1957), pp. 201–226 (cit. on pp. 216, 217).

[635] J. Snoek, H. Larochelle, and R. P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. In: *Advances in Neural Information Processing Systems 25: Procs. of the 2012 Conference*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf (cit. on p. 288).

[636] C. Sohler and D. P. Woodruff. “Strong Coresets for *k*-Median and Subspace Approximation: Goodbye Dimension”. In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2018*. Ed. by M. Thorup. IEEE Computer Society, 2018, pp. 802–813. DOI: https://doi.org/10.1109/FOCS.2018.00081 (cit. on p. 212). **SFB876-A2**

[637] C. Sohler and D. P. Woodruff. “Subspace embeddings for the L_1-norm with applications”. In: *Procs. of the ACM Symposium on Theory of Computing 2011*. Ed. by L. Fortnow and S. P. Vadhan. ACM, 2011, pp. 755–764 (cit. on pp. 87, 90, 95). **SFB876-C4**

[638] R. Sokal and P. Sneath. *Principles of Numerical Taxonomy*. Books in biology. W. H. Freeman, 1963 (cit. on pp. 216, 217).

[639] J. Sompolski, M. Zukowski, and P. A. Boncz. “Vectorization vs. Compilation in Query Execution”. In: *Procs. of the Int. Workshop on Data Management on New Hardware 2011*. Athens, Greece, June 2011, pp. 33–40 (cit. on p. 386).

[640] K. Song, X. Yao, F. Nie, X. Li, and M. Xu. “Weighted Bilateral K-means Algorithm for Fast Co-clustering and Fast Spectral Clustering”. In: *Pattern Recognition* (2020), p. 107560 (cit. on p. 232).

[641] S. Song, D. Yan, and Y. Xie. “Design of control system based on hand gesture recognition”. In: *Procs. of the IEEE Int. Conference on Networking, Sensing and Control 2018*. IEEE. 2018, pp. 1–4 (cit. on p. 61).

[642] D. Sontag and T. Jaakkola. “Tree Block Coordinate Descent for MAP in Graphical Models”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2009*. Vol. 8. 2009, pp. 544–551 (cit. on p. 407).

[643] J. Spencer. *Ten Lectures on the Probabilistic Method*. 2nd Edition. Society for Industrial and Applied Mathematics, 1994 (cit. on pp. 145, 147).

[644] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. “AspectC++: An Aspect-Oriented Extension to C++”. In: *Procs. of the Int. Conference on Technology of Object-Oriented Languages and Systems 2002*. Sydney, Australia, Feb. 2002, pp. 53–60 (cit. on p. 40).

[645] O. Spinczyk and D. Lohmann. “The Design and Implementation of AspectC++”. In: *Knowledge-Based Systems, Special Issue on Techniques to Produce Intelligent Secure Software* 20.7 (2007), pp. 636–651 (cit. on p. 40).

[646] B. K. Stöcker, T. Schäfer, P. Mutzel, J. Köster, N. M. Kriege, and S. Rahmann. “Protein Complex Similarity Based on Weisfeiler-Lehman Labeling”. In: *Procs. of the Int. Conference on Similarity Search and Applications 2019*. Ed. by G. Amato, C. Gennaro, V. Oria, and M. Radovanovic. Cham: Springer Int. Publishing, 2019, pp. 308–322 (cit. on pp. 116, 117). **SFB876-A6, SFB876-C1**

[647] J. Stokes et al. “A Deep Learning Approach to Antibiotic Discovery”. In: *Cell* 180 (Feb. 2020), 688–702.e13 (cit. on p. 116).

[648] M. Stolpe, K. Bhaduri, K. Das, and K. Morik. “Anomaly Detection in Vertically Partitioned Data by Distributed Core Vector Machines”. In: *Procs. of the European Conference on Machine Learning and Knowledge Discovery in Databases 2013*. Ed. by H. Blockeel, K. Kersting, S. Nijssen, and F. Železný. Springer, 2013, pp. 321–336 (cit. on p. 89). **SFB876-B3**

[649] J. Streicher. *Data Modeling of Ubiquitous System Software*. Tech. rep. 7. TU Dortmund University, Aug. 2014 (cit. on p. 32). **SFB876-A1**

[650] E. Strubell, A. Ganesh, and A. McCallum. “Energy and Policy Considerations for Modern Deep Learning Research”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2020, The Innovative Applications of Artificial Intelligence Conference 2020, The AAAI Symposium on Educational Advances in Artificial Intelligence 2020*. AAAI Press, 2020, pp. 13693–13696. URL: https://aaai.org/ojs/index.php/AAAI/article/view/7123 (cit. on p. 6).

[651] X. Sun, X. Peng, P. Chen, R. Liu, J. Seo, and S. Yu. “Binary neural network with 16 Mb RRAM macro chip for classification and online training”. In: *Procs. of the Asia and South Pacific Design Automation Conference 2018*. 2018, pp. 574–579 (cit. on p. 327).

[652] X. Sun et al. “Low-VDD Operation of SRAM Synaptic Array for Implementing Ternary Neural Network”. In: *Procs. of IEEE Transactions on Very Large Scale Integration Systems 2017* 25.10 (2017), pp. 2962–2965 (cit. on p. 326).

[653] Y. Sun, Y. Zhang, and G. Xue. “SmartWheelTag: Flexible and Battery-less User Interface for Drivers”. In: *Procs. of the IEEE Vehicular Networking Conference 2018*. IEEE. 2018, pp. 1–2 (cit. on p. 61).

[654] C. Sutton and A. McCallum. “An Introduction to Conditional Random Fields for Relational Learning”. In: *Introduction to Statistical Relational Learning*. Ed. by L. Getoor and B. Taskar. MIT Press, 2007. URL: http://www.cs.umass.edu/~mccallum/papers/crf-tutorial.pdf (cit. on p. 106).

[655] C. Sutton and A. McCallum. “An Introduction to Conditional Random Fields”. In: *Foundations and Trends in Machine Learning* 4.4 (2012), pp. 267–373 (cit. on pp. 410, 416).

[656] Synopsys, Inc. *https://www.synopsys.com/silicon/tcad.html*. 2022 (cit. on p. 328).

[657] B. Tanasa, U. D. Bordoloi, P. Eles, and Z. Peng. “Probabilistic Response Time and Joint Analysis of Periodic Tasks”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2015*. 2015, pp. 235–246 (cit. on p. 363).

[658] M. B. Teitz and P. Bart. “Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph”. In: *Operations Research* 16.5 (1968), pp. 955–961 (cit. on p. 184).

[659] M. Thorup. “Quick *k*-Median, *k*-Center, and Facility Location for Sparse Graphs”. In: *SIAM Journal on Computing* 34.2 (2005), pp. 405–432 (cit. on p. 201).

[660] R. Tibshirani. “Regression Shrinkage and Selection via the Lasso”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 58.1 (1996), pp. 267–288 (cit. on p. 105).

[661] R. M. Timoney. *OpenMath LaTeX to OpenMath Converter*. Tech. rep. 1999, pp. 1–9. URL: https://www.maths.tcd.ie/%7B~%7Drichardt/openmath/ml2om.pdf (cit. on p. 162).

[662] S. Tobuschat, R. Ernst, A. Hamann, and D. Ziegenbein. “System-level timing feasibility test for cyber-physical automotive systems”. In: *Procs. of the IEEE Symposium on Industrial Embedded Systems 2016*. May 2016, pp. 1–10 (cit. on p. 368).

[663] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. M. Borgwardt. “Wasserstein Weisfeiler-Lehman Graph Kernels”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019, pp. 6436–6446 (cit. on p. 123).

[664] E. Tolochinsky and D. Feldman. “Coresets For Monotonic Functions with Applications to Deep Learning”. In: *arXiv: Computing Research Repository* (2018). DOI: arXiv:1802.07382 (cit. on p. 87).

[665] A. Tozawa, M. Tatsubori, T. Onodera, and Y. Minamide. “Copy-on-write in the PHP Language”. In: *Procs. of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages 2009*. POPL ’09. ACM, 2009, pp. 200–212 (cit. on p. 308).

[666] P. Tözün and H. Kotthaus. “Scheduling Data-Intensive Tasks on Heterogeneous Many Cores”. In: *IEEE Data Engineering Bulletin* 42.1 (2019), pp. 61–72. URL: http://sites.computer.org/debull/A19mar/p61.pdf (cit. on pp. 285, 287, 289). **SFB876-A3, SFB876-C5**

[667] D.-S. Tran, N.-H. Ho, H.-J. Yang, E.-T. Baek, S.-H. Kim, and G. Lee. “Real-time hand gesture spotting and recognition using RGB-D camera and 3D convolutional neural network”. In: *Applied Sciences* 10.2 (2020), p. 722 (cit. on p. 61).

[668] M. Trentzsch et al. “A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs”. In: *Procs. of the IEEE Int. Electron Devices Meeting 2016*. IEEE. 2016, pp. 11–5 (cit. on p. 328).

[669] T. H. Trinh, M.-t. Luong, Q. V. Le, and G. Brain. “Selfie : Self-supervised Pretraining for Image Embedding”. In: (2019) (cit. on p. 163).

[670] H. Truong et al. “Capband: Battery-free successive capacitance sensing wristband for hand gesture recognition”. In: *Procs. of the ACM Conference on Embedded Networked Sensor Systems 2018*. 2018, pp. 54–67 (cit. on p. 61).

[671] S. Tschiatschek, P. Reinprecht, M. Mücke, and F. Pernkopf. “Bayesian Network Classifiers with Reduced Precision Parameters”. In: *Procs. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2012*. 2012 (cit. on p. 407).

[672] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei. “RANA: Towards Efficient Neural Acceleration with Refresh-Optimized Embedded DRAM”. In: *Procs. of the Int. Symposium on Computer Architecture 2018*. 2018, pp. 340–352 (cit. on p. 327).

[673] A. Turing. “Computing Machinery and Intelligence”. In: *Mind*. New Series 59.236 (1950), pp. 433–460. URL: http://www.jstor.org/stable/2251299 (cit. on p. 5).

[674] H. Turner, T. Bailey, and W. Krzanowski. “Improved biclustering of microarray data demonstrated through systematic performance tests”. In: *Computational Statistics & Data Analysis* 48.2 (2005), pp. 235–254 (cit. on p. 233).

[675] M. Tzoufras, M. Gajek, and A. Walker. “Magnetoresistive RAM for error resilient XNOR-Nets”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1905.10927 (cit. on p. 327).

[676] E. Ustinova and V. Lempitsky. “Learning Deep Embeddings with Histogram Loss”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. 2016 (cit. on p. 168).

[677] O. Vaidya, S. Gandhe, A. Sharma, A. Bhate, V. Bhosale, and R. Mahale. “Design and Development of Hand Gesture based Communication Device for Deaf and Mute People”. In: *Procs. of the IEEE Bombay Section Signature Conference 2020*. IEEE. 2020, pp. 102–106 (cit. on p. 61).

[678] S. Valat, M. Pérache, and W. Jalby. “Introducing Kernel-level Page Reuse for High Performance Computing”. In: *Procs. of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness 2013*. MSPC ’13. ACM, 2013, 3:1–3:9 (cit. on p. 307).

[679] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger. “Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA?” In: *Procs. of the IEEE Int. Symposium on Field-Programmable Custom Computing Machines 2012*. IEEE. 2012, pp. 232–239 (cit. on p. 340).

[680] V. N. Vapnik. *The Nature of Statistical Learning Theory*. New York: Springer, 1995 (cit. on p. 10).

[681] A. Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Procs. of the 2017 Conference*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on pp. 163, 165, 166).

[682] S. A. Vavasis. “On the complexity of nonnegative matrix factorization”. In: *SIAM journal on optimization* 20.3 (2009), pp. 1364–1377 (cit. on p. 229).

[683] P. Veličković, G. Cucurull, A. Casanova, A. Romero, and Y. B. Liò. “Graph Attention Networks”. In: *Procs. of the Int. Conference on Learning Representation 2018*. 2018 (cit. on pp. 130, 132, 136).

[684] S. I. Venieris and C.-S. Bouganis. “fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs”. In: *Procs. of the Int. Symposium on Field-Programmable Custom Computing Machines 2016*. 2016 (cit. on p. 254).

[685] J.-P. Vert. “The optimal assignment kernel is not positive definite”. In: *arXiv: Computing Research Repository* (2008). URL: arXiv:0801.4061 (cit. on p. 121).

[686] A. F. Vincent et al. “Spin-Transfer Torque Magnetic Memory as a Stochastic Memristive Synapse for Neuromorphic Systems”. In: *Transactions on Biomedical Circuits and Systems* 9 (2015), pp. 166–174 (cit. on p. 327).

[687] O. Vinyals, S. Bengio, and M. Kudlur. “Order Matters: Sequence to Sequence for Sets”. In: *Procs. of the Int. Conference on Learning Representation 2016*. 2016 (cit. on p. 135).

[688] J. S. Vitter. “Random sampling with a reservoir”. In: *ACM Transactions on Mathematical Software (TOMS)* 11.1 (1985), pp. 37–57 (cit. on p. 76).

[689] M. Wahib and N. Maruyama. “Scalable Kernel Fusion for Memory-Bound GPU Applications”. In: *Procs. of the Int. Conference for High Performance Computing, Networking, Storage and Analysis 2014*. IEEE Press, 2014, pp. 191–202 (cit. on p. 388).

[690] M. Wainwright, T. Jaakkola, and A. Willsky. “A new class of upper bounds on the log partition function”. In: *IEEE Transactions on Information Theory* 51.7 (2005), pp. 2313–2335 (cit. on p. 105).

[691] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. “Tree-reweighted belief propagation algorithms and approximate ML estimation by pseudo-moment matching”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2003*. 2003 (cit. on p. 407).

[692] M. J. Wainwright and M. I. Jordan. “Graphical Models, Exponential Families, and Variational Inference”. In: *Foundations and Trends in Machine Learning* 1.1–2 (2008), pp. 1–305. DOI: http://dx.doi.org/10.1561/2200000001 (cit. on pp. 103, 105, 110, 408, 411).

[693] A. X. Wang, C. Tran, N. Desai, D. Lobell, and S. Ermon. “Deep Transfer Learning for Crop Yield Prediction with Remote Sensing Data”. In: *Procs. of the ACM SIGCAS Conference on Computing and Sustainable Societies 2018*. COMPASS ’18. New York, NY, USA: ACM, 2018, 50:1–50:5. DOI: http://doi.acm.org/10.1145/3209811.3212707 (cit. on p. 162).

[694] H. Wang, F. Nie, H. Huang, and F. Makedon. “Fast nonnegative matrix tri-factorization for large-scale data co-clustering”. In: *Procs. of the Int. Joint Conference on Artificial Intelligence 2011*. 2011, p. 1553 (cit. on p. 232).

[695] J. Wang, T. Jebara, and S.-F. Chang. “Semi-supervised learning using greedy max-cut”. In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 771–800 (cit. on p. 145).

[696] M. Wang et al. “Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs”. In: *Int. Conference on Learning Representations Workshop on Representation Learning on Graphs and Manifolds* (2019). URL: https://arxiv.org/abs/1909.01315 (cit. on p. 126).

[697] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. “Dynamic Graph CNN for Learning on Point Clouds”. In: *ACM Transactions on Graphics (TOG)* (2019) (cit. on pp. 130, 132, 135).

[698] Y. Wang et al. “Storage-less and Converter-less Photovoltaic Energy Harvesting with Maximum Power Point Tracking for Internet of Things”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* PP.99 (2015) (cit. on p. 53).

[699] S. Weber, A. Gelman, D. Lee, M. Betancourt, A. Vehtari, and A. Racine-Poon. “Bayesian aggregation of average data: An application in drug development”. In: *Annals of Applied Statistics* (2018) (cit. on p. 94).

[700] B. Weisfeiler. *On Construction and Identification of Graphs*. Lecture Notes in Mathematics, Vol. 558. Springer, 1976 (cit. on p. 118).

[701] B. Weisfeiler and A. A. Lehman. “A Reduction of a Graph to a Canonical Form and an Algebra Arising During this Reduction”. In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968) (cit. on pp. 132, 137).

[702] G. Werner-Allen, P. Swieskowski, and M. Welsh. “MoteLab: A Wireless Sensor Network Testbed”. In: *Procs. of the Int. Symposium on Information Processing in Sensor Networks 2005*. IPSN ’05. IEEE Press, 2005. URL: http://dl.acm.org/citation.cfm?id=1147685.1147769 (cit. on p. 35).

[703] J. Whittaker, S. Garside, and K. Lindveld. “Tracking and predicting a network traffic process”. In: *Int. Journal of Forecasting* 13.1 (Mar. 1997), pp. 51–61. URL: http://www.sciencedirect.com/science/article/B6V92-3SWXN18-6/2/97477ccea59869520e0f7a0fcd1c19bc (cit. on p. 101).

[704] A. Wieder and B. B. Brandenburg. “Efficient partitioning of sporadic real-time tasks with shared resources and spin locks”. In: *Procs. of the Int. Symposium on Industrial Embedded Systems 2013*. 2013, pp. 49–58. DOI: http://dx.doi.org/10.1109/SIES.2013.6601470 (cit. on p. 368).

[705] B. Williams and L. Hoel. “Modeling and Forecasting Vehicular Traffic Flow as a Seasonal ARIMA Process: Theoretical Basis and Empirical Results.” In: *Journal of Transportation Engineering* 129.6 (2003), pp. 664–672 (cit. on p. 101).

[706] R. Winkelmann. *Econometric Analysis of Count Data*. 5th ed. Springer, 2008 (cit. on pp. 92, 93).

[707] D. Wishart. “256. Note: An Algorithm for Hierarchical Classifications”. In: *Biometrics* 25.1 (1969), pp. 165–170. URL: http://www.jstor.org/stable/2528688 (cit. on pp. 216, 218).

[708] D. P. Woodruff. “Sketching as a Tool for Numerical Linear Algebra”. In: *Foundations and Trends in Theoretical Computer Science* 10.1-2 (2014), pp. 1–157 (cit. on p. 89).

[709] D. P. Woodruff and Q. Zhang. “Subspace Embeddings and ℓ_p-Regression Using Exponential Random Variables”. In: *Procs. of the Conference on Learning Theory 2013*. 2013, pp. 546–567 (cit. on pp. 87, 89, 90, 95).

[710] F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger. “Simplifying Graph Convolutional Networks”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019 (cit. on pp. 130, 136, 137).

[711] H. Wu, G. Diamos, J. Wang, S. Cadambi, S. Yalamanchili, and S. Chakradhar. “Optimizing Data Warehousing Applications for GPUs Using Kernel Fusion/Fission”. In: *Procs. of the Parallel and Distributed Processing Symposium Workshops and PhD Forum 2012*. IEEE, 2012, pp. 2433–2442 (cit. on p. 384).

[712] J. Xie and J. Yang. “A survey of join processing in data streams”. In: *Data Streams: Models and Algorithms*. Ed. by C. C. Aggarwal. Springer, Jan. 2007, pp. 209–236 (cit. on p. 20).

[713] J. Xu et al. “Stacked sparse autoencoder (SSAE) for nuclei detection on breast cancer histopathology images”. In: *IEEE transactions on medical imaging* 35.1 (2016), pp. 119–130 (cit. on p. 260).

[714] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. “How Powerful are Graph Neural Networks?” In: *Procs. of the Int. Conference on Learning Representation 2019*. 2019 (cit. on pp. 117, 127, 130, 132, 133, 137, 139).

[715] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. “Representation Learning on Graphs with Jumping Knowledge Networks”. In: *Procs. of the Int. Conference on Machine Learning 2018*. 2018 (cit. on p. 130).

[716] J. Yang, H. Wang, W. Wang, and P. Yu. “An Improved Biclustering Method for Analyzing Gene Expression Profiles”. In: *Int. Journal on Artificial Intelligence Tools* 14 (Oct. 2005), pp. 771–790 (cit. on p. 239).

[717] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann. “Bit Error Tolerance of a CIFAR-10 Binarized Convolutional Neural Network Processor”. In: *Procs. of the Int. Symposium on Computer Architecture 2018*. 2018, pp. 1–5 (cit. on p. 326).

[718] Z. Yang, W. Cohen, and R. Salakhutdinov. “Revisiting Semi-supervised Learning with Graph Embeddings”. In: *Procs. of the Int. Conference on Machine Learning 2016*. 2016 (cit. on p. 136).

[719] W. Yao, A. S. Bandeira, and S. Villar. “Experimental performance of graph neural networks on random instances of max-cut”. In: *Procs. of the SPIE Optical Engineering + Application Conference, Wavelets and Sparsity XVIII, 2019*. Ed. by D. V. D. Ville, M. Papadakis, and Y. M. Lu. Vol. 11138. Int. Society for Optics and Photonics. SPIE, 2019, pp. 242–251 (cit. on p. 145).

[720] M. Yayla et al. “FeFET-based Binarized Neural Networks Under Temperature-dependent Bit Errors”. In: *IEEE Transactions on Computers* (2021). DOI: 10.1109/TC.2021.3104736. URL: https://ieeexplore.ieee.org/document/9513530 (cit. on p. 328). **SFB876-A1**

[721] T. Ye, H. Zhou, W. Y. Zou, B. Gao, and R. Zhang. “RapidScorer: Fast tree ensemble evaluation by maximizing compactness in data level parallelization”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2018*. 2018 (cit. on p. 340).

[722] Y. Ye, K. A. Ross, and N. Vesdapunt. “Scalable Aggregation on Multicore Processors”. In: *Procs. of the Int. Workshop on Data Management on New Hardware 2011*. 2011, pp. 1–11 (cit. on p. 394).

[723] Z. Yin and R. Collins. “Belief Propagation in a 3D Spatio-temporal MRF for Moving Object Detection”. In: *IEEE Computer Vision and Pattern Recognition (CVPR)* (June 2007) (cit. on p. 102).

[724] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. “Hierarchical Graph Representation Learning with Differentiable Pooling”. In: *Advances in Neural Information Processing Systems 31: Procs. of the 2018 Conference*. 2018 (cit. on pp. 126, 135). **SFB876-A6**

[725] J. Yoo and S. Choi. “Orthogonal nonnegative matrix tri-factorization for co-clustering: multiplicative updates on Stiefel manifolds”. In: *Information processing & management* 46.5 (2010), pp. 559–570 (cit. on pp. 232, 239).

[726] L. Yu, J. Shen, J. Li, and A. Lerer. “Scalable Graph Neural Networks for Heterogeneous Graphs”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv : 2011 . 09679 (cit. on pp. 130, 137).

[727] S. Yu et al. “Binary neural network with 16 Mb RRAM macro chip for classification and online training”. In: *Procs. of the Int. Electron Devices Meeting 2016*. 2016, pp. 16.2.1–16.2.4 (cit. on p. 327).

[728] Y. Yuan, R. Lee, and X. Zhang. “The Yin and Yang of processing data warehousing queries on GPU devices”. In: *Procs. of the VLDB Endowment* 6.10 (2013), pp. 817–828 (cit. on pp. 384, 401).

[729] R. Zanibbi, G. Topi, M. Kohlhase, and K. Davila. “NTCIR-12 MathIR Task Overview”. In: *Procs. of the NTCIR Conference on Evaluation of Information Access Technologies 2016*. Tokyo, Japan, 2016 (cit. on pp. 162, 172).

[730] F. Zeidler. *Beitrag zur Selbststeuerung cyberphysischer Produktionssysteme in der auftragsbezogenen Fertigung*. Praxiswissen Service, 2019 (cit. on p. 45).

[731] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. “GraphSAINT: Graph Sampling Based Inductive Learning Method”. In: *Procs. of the Int. Conference on Learning Representation 2020*. 2020 (cit. on pp. 136, 141).

[732] H. Zeng et al. “Deep Graph Neural Networks with Shallow Subgraph Samplers”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv:2012.01.380 (cit. on p. 137).

[733] H. Zha, X. He, C. Ding, H. Simon, and M. Gu. “Bipartite graph partitioning and data clustering”. In: *Procs. of the Int. Conference on Information and Knowledge Management 2001*. ACM. 2001, pp. 25–32 (cit. on p. 232).

[734] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. “Optimizing FPGA-based accelerator design for deep convolutional neural networks”. In: *Procs. of the ACM/SIGDA Int. Symposium on Field-Programmable Gate Arrays 2015*. ACM. 2015, pp. 161–170 (cit. on p. 254).

[735] K. Zhang et al. “Hetero-DB: Next Generation High-Performance Database Systems by Best Utilizing Heterogeneous Computing and Storage Resources”. In: *Journal of Computer Science and Technology* 30.4 (2015), pp. 657–678 (cit. on p. 384).

[736] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. “An End-to-End Deep Learning Architecture for Graph Classification”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2018*. 2018 (cit. on pp. 126, 135).

[737] T. Zhang, R. Ramakrishnan, and M. Livny. “Birch: A new data clustering algorithm and its applications”. In: *Data Mining and Knowledge Discovery* 1.2 (June 1997), pp. 141–182. URL: http://www.ece.nwu.edu/~harsha/Clustering/newkbspaper.ps (cit. on pp. 215, 219, 220).

[738] T. Zhang, R. Ramakrishnan, and M. Livny. “BIRCH: An Efficient Data Clustering Method for Very Large Databases”. In: *Procs. of the ACM SIGMOD Int. Conference on Management of Data 1996*. 1996, pp. 103–114. URL: http://www.ece.nwu.edu/~harsha/Clustering/sigmodpaper.ps (cit. on pp. 215, 219, 220).

[739] Y. Zhang, S. Burer, and W. N. Street. “Ensemble pruning via semi-definite programming”. In: *Journal of Machine Learning Research* 7 (2006), pp. 1315–1338. URL: http://www.jmlr.org/papers/volume7/zhang06a/zhang06a.pdf (cit. on p. 340).

[740] Z.-Y. Zhang, T. Li, C. Ding, X.-W. Ren, and X.-S. Zhang. “Binary matrix factorization for analyzing gene expression data”. In: *Data Mining and Knowledge Discovery* 20.1 (2010), pp. 28–52 (cit. on p. 232).

[741] Z.-Y. Zhang, Y. Wang, and Y.-Y. Ahn. “Overlapping community detection in complex networks using symmetric binary matrix factorization”. In: *Physical Review E* 87.6 (2013), p. 062803 (cit. on p. 232).

[742] Z. Zhang, C. Ding, T. Li, and X. Zhang. “Binary matrix factorization with applications”. In: *Procs. of the IEEE Int. Conference on Data Mining 2007*. IEEE. 2007, pp. 391–400 (cit. on p. 232).

[743] F. Zhao and N. Park. “Using Geographically Weighted Regression Models to Estimate Annual Average Daily Traffic”. In: *Journal of the Transportation Research Board* 1879.12 (2004), pp. 99–107 (cit. on p. 102).

[744] W. Zhao et al. “An FPGA-based Framework for Training Convolutional Neural Networks”. In: *Procs. of the Int. Conference on Application-specific Systems, Architectures and Processors 2016*. 2016 (cit. on pp. 254, 255).

[745] W. Zhong and R. Zanibbi. “Structural similarity search for formulas using leaf-root paths in operator subtrees”. In: *Procs. of the European Conference on Information Retrieval 2019*. Springer, 2019, pp. 116–129 (cit. on pp. 161, 162).

[746] J. Ziv and A. Lempel. “A universal algorithm for sequential data compression”. In: *IEEE Transactions on Information Theory* 23.3 (1977), pp. 337–343 (cit. on p. 157).

[747] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu. “Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019 (cit. on p. 136).

[748] H. Zou and T. Hastie. “Regularization and variable selection via the Elastic Net”. In: *Journal of the Royal Statistical Society B* 67 (2005), pp. 301–320 (cit. on p. 105).

[749] M. Zukowski, P. A. Boncz, N. Nes, and S. Héman. “MonetDB/X100–A DBMS In The CPU Cache”. In: *IEEE Data Engineering Bulletin* 28.2 (2005), pp. 17–22 (cit. on p. 385).

[750] U. Zwick. “Analyzing the MAX 2-SAT and MAX DI-CUT approximation algorithms of Feige and Goemans”. In: (2000) (cit. on p. 145).

Index

https://doi.org/10.1515/9783110785944-011

List of Contributors

Editors

Morik, Katharina, Prof. Dr. TU Dortmund University, Department of Computer Science, Head of the chair for Artificial Intelligence & Speaker of the Collaborative Research Center 876 on Providing Information by Resource-Constrained Data Analysis, katharina.morik@tu-dortmund.de

Marwedel, Peter, Prof. Dr. TU Dortmund University, Department of Computer Science, CRC 876, peter.marwedel@tu-dortmund.de

Contributors

Amrouch, Hussam, Jun.-Prof. Dr.-Ing. Universität Stuttgart, amrouch@iti.uni-stuttgart.de

Babbar, Rohit, Ph.D. Aalto University, Assistant Professor Department of Computer Science, rohit.babbar@aalto.fi

Bertram, Nico, M. Sc. TU Dortmund University, Department of Computer Science, CRC 876, nico.bertram@tu-dortmund.de

Borchert, Christoph, Dr.-Ing. Osnabrück University, Institute of Computer Science, CRC 876, christoph.borchert@uni-osnabrueck.de

Buschhoff, Markus, Dr.-Ing. TU Dortmund University, Department of Computer Science, Design Automation for Embedded Systems Group, CRC 876, markus.buschhoff@tu-dortmund.de

Buschjäger, Sebastian, M. Sc. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, sebastian.buschjaeger@tu-dortmund.de

Chen, Jian-Jia, Prof. Dr. TU Dortmund University, Department of Computer Sciences, Head of the chair for Design Automation for Embedded Systems, CRC 876, jian-jia.chen@cs.uni-dortmund.de

Chen, Kuan-Hsun, Dr.-Ing. University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science, CRC 876, k.h.chen@utwente.nl

Ellert, Jonas, M. Sc. TU Dortmund University, Department of Computer Science, Chair 11 Algorithm Engineering, CRC 876, jonas.ellert@tu-dortmund.de

Falkenberg, Robert, M. Sc. TU Dortmund University, Communications Networks Institute, CRC 876, robert.falkenberg@tu-dortmund.de

Fey, Matthias, M. Sc. TU Dortmund University, Department of Computer Science, Computer Graphics Lab, CRC 876, matthias.fey@tu-dortmund.de

Fischer, Johannes, Prof. Dr. TU Dortmund University, Department of Computer Science, Chair 11 Algorithm Engineering, CRC 876, johannes.fischer@cs.tu-dortmund.de

https://doi.org/10.1515/9783110785944-012

Funke, Henning, M. Sc. TU Dortmund University, Department of Computer Science, Databases and Information Systems Group, CRC 876, henning.funke@cs.tu-dortmund.de

Gómez, Andrés, Ph.D. University of St.Gallen, Interaction- and Communication-based Systems Group, andres.gomez@unisg.ch

Guo, Ce, Professor Imperial College London, Faculty of Engineering, Department of Computing, c.guo@imperial.ac.uk

Hess, Sibylle, Dr. Eindhoven University of Technology, Assistant Professor, Mathematics and Computer Science, Data Mining, CRC 876, s.c.hess@tue.nl

Kotthaus, Helena, Dr. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, helena.kotthaus@cs.tu-dortmund.de

Kriege, Nils M., Prof. Dr. University of Vienna, Faculty of Computer Science, Data Mining and Machine Learning (DM), CRC 876, nils.kriege@univie.ac.at

Krivošija, Amer, Dr. TU Dortmund University, Department of Statistics, Mathematical Statistics with Applications in Biometrics, Projekt FAIR, CRC 876, amer.krivosija@tu-dortmund.de

Lang, Andreas, M. Sc. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, andreas.lang@tu-dortmund.de

Lenssen, Lars, M. Sc. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, lars.lenssen@tu-dortmund.de

Lochmann, Alexander, M. Sc. TU Dortmund University, Department of Computer Science, Embedded System Software Group, CRC 876, alexander.lochmann@tu-dortmund.de

Luk, Wayne, Prof. Dr. Imperial College London, Faculty of Engineering, Department of Computing, w.luk@imperial.ac.uk

Masoudinejad, Mojtaba, Dr.-Ing. TU Dortmund University, Department of Computer Science, Design Automation for Embedded Systems Group, CRC 876, mojtaba.masoudinejad@tu-dortmund.de

Mayer, Simon, Prof. Dr. University of St. Gallen, Interaction- and Communication-based Systems Group, simon.mayer@unisg.ch

Morris, Christopher, Dr. RWTH Aachen, Faculty of Electrical Engineering and Information Technology, CRC 876, christopher.morris@tu-dortmund.de

Munteanu, Alexander, Dr. TU Dortmund University, Dortmund Data Science Center, Faculties of Statistics and Computer Science, CRC 876, alexander.munteanu@tu-dortmund.de

Pfahler, Lukas, M. Sc. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, lukas.pfahler@tu-dortmund.de

Piatkowski, Nico, Dr. Fraunhofer Institute for Intelligent Analysis and Information Systems, CRC 876, nico.piatkowski@iais.fraunhofer.de

Schubert, Erich, Prof. Dr. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, erich.schubert@cs.tu-dortmund.de

Schultheis, Erik, M. Sc. Aalto University, Department of Computer Science, erik.schultheis@aalto.fi

Shi, Junjie, M. Sc. TU Dortmund University, Department of Computer Science, Design Automation for Embedded Systems Group, CRC 876, junjie.shi@tu-dortmund.de

Sliwa, Benjamin, Dr.-Ing. TU Dortmund University, Senior researcher Communications Networks Institute, CRC 876, benjamin.sliwa@tu-dortmund.de

Spinczyk, Olaf, Prof. Dr.-Ing. Osnabrück University, Department for Mathematics and Computer Science, Leader of the Embedded Software Systems Group, CRC 876, olaf@uos.de

Streicher, Jochen, Dipl.-Inf. TU Dortmund University, Department of Computer Science, Embedded System Software Group, CRC 876, jochen.streicher@tu-dortmund.de

Suter, Lars, M.A. University of St.Gallen, Interaction- and Communication-based Systems Group, larskai.suter@student.unisg.ch

Teubner, Jens, Prof. Dr. TU Dortmund University, Department of Computer Science, Databases and Information Systems Group, CRC 876, jens.teubner@cs.tu-dortmund.de

Weichert, Frank, Priv.-Doz. Dr. TU Dortmund University, Department of Computer Science, Intelligent Sensing in Computer Graphics, CRC 876, frank.weichert@tu-dortmund.de

Yayla, Mikail, M. Sc. TU Dortmund University, Department of Computer Science, Design Automation for Embedded Systems Group, CRC 876, mikail.yayla@tu-dortmund.de

Technical Editors

Becker, Andreas, Dr. TU Dortmund University, Department of Computer Science, CRC 876, andreas3.becker@tu-dortmund.de

Buß, Jens, Dr. TU Dortmund University, Department of Computer Science, Managing Director of the Collaborative Research Center 876, jens.buss@tu-dortmund.de

Acknowledgment

Part of the work on this book is the result of research of the Collaborative Research Center 876 "Providing Information by Resource-Constrained Analysis", which was funded from 2011–2022 by the Deutsche Forschungsgemeinschaft (DFG) under DFG project number 124020371, see: https://gepris.dfg.de/gepris/projekt/124020371?language=en.