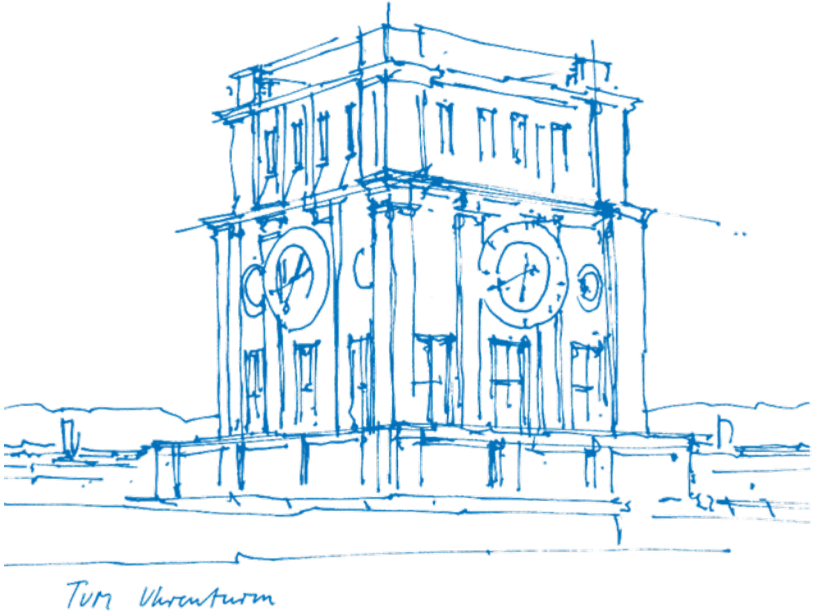
Advances in Data-Driven Solver Selection for Sparse Linear Matrices

SIAM Conference on Parallel Processing for Scientific Computing (PP24)

Hayden Liu Weng   Felix Dietrich   Hans Joachim Bungartz

Technical University of Munich
School of Computation, Information and Technology
Chair of Scientific Computing

Baltimore, March 8, 2024



Outline

Challenges in HPC

Many challenges in attaining peak performance:

- hardware and compilers
- system topology
- heterogeneous systems
- available libraries, algorithms, and implementations keep increasing

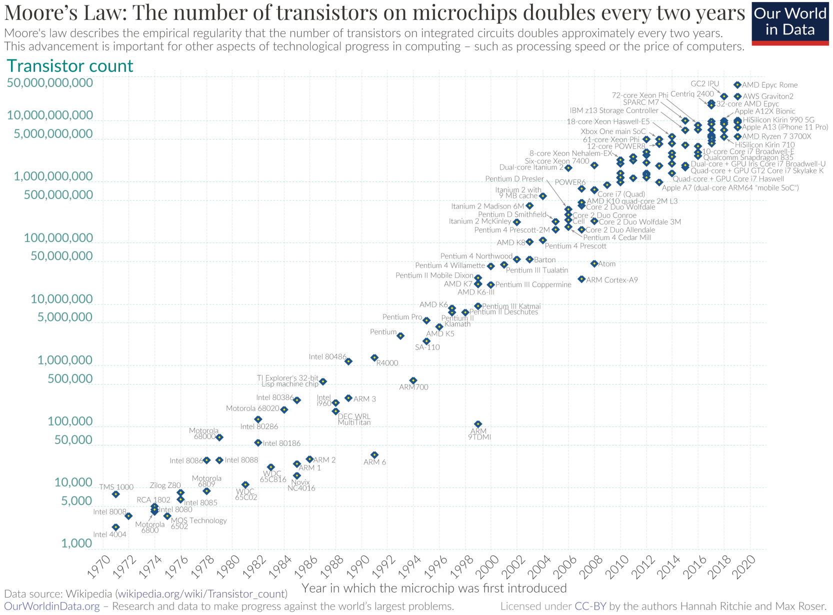


Figure: Transistor count over time. From `https://ourworldindata.org/moores-law`

Challenges in HPC

Many challenges in attaining ~~peak~~ optimal performance:

- hardware and compilers
- system topology
- heterogeneous systems
- available libraries, algorithms, and implementations keep increasing

Main issues & potential trade-offs:

- tuning cannot be done entirely *a priori*
- architecture-aware tuning vs. portability
- ease of use vs. customization
- tuning can be (very) expensive
- when to tune

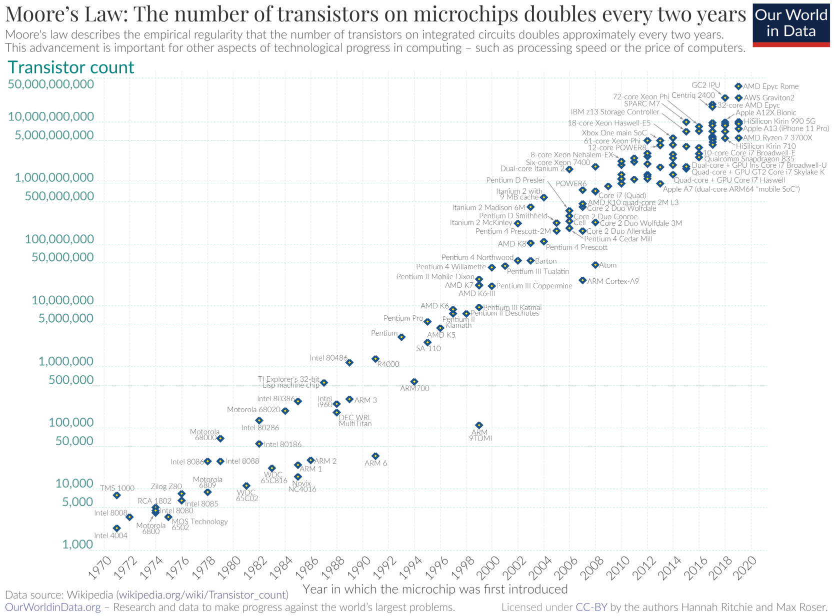


Figure: Transistor count over time. From `https://ourworldindata.org/moores-law`

HPC application development

Three main roles can be identified:

- field/application expert: develops the PDE/model
- algorithms expert: implements and tunes numerical schemes
- optimization expert: optimizes the code (intrinsics)

HPC experts cannot (generally) be expected to fulfill all of the above roles
$\Rightarrow$ The decisions should try to be decoupled as much as possible using abstractions, code generation, and numerical libraries

HPC application development

Three main roles can be identified:

- field/application expert: develops the PDE/model
- algorithms expert: implements and tunes numerical schemes
- optimization expert: optimizes the code (intrinsics)

HPC experts cannot (generally) be expected to fulfill all of the above roles
$\Rightarrow$ The decisions should try to be decoupled as much as possible using abstractions, code generation, and numerical libraries
$\Rightarrow$ Traditionally, HPC Centers mainly offer support regarding this last aspect

HPC application development

Three main roles can be identified:

- field/application expert: develops the PDE/model
- algorithms expert: implements and tunes numerical schemes
- optimization expert: optimizes the code (intrinsics)

HPC experts cannot (generally) be expected to fulfill all of the above roles
$\Rightarrow$ The decisions should try to be decoupled as much as possible using abstractions, code generation, and numerical libraries
$\Rightarrow$ Traditionally, HPC Centers mainly offer support regarding this last aspect
$\Rightarrow$ HPC development should be more approachable [to the domain expert], with special focus on algorithmic aspects!

Self-Adapting Numerical Software (SANS) [1]

Beyond the initial mathematical model, successful management of complex computational environments involves:

- Algorithmic decisions
- Management of the parallel environment
- Processor-specific tuning of kernels

Meaning that the following elements are necessary: Numerical Components + Analysis Modules + Intelligent switch

Examples: ATLAS, PHiPAC, FFTW, . . .

Outline

The Thirteen Berkeley Dwarfs

Main Algorithms/Challenges in Computing [2]:

- Dense Linear Algebra
- Sparse Linear Algebra
- Spectral Methods
- N-body problems
- Structured grids
- Unstructured grids
- MapReduce (Monte Carlo)
- Combinational Logic
- Graph Traversal
- Dynamic Programming
- Back-track and Branch & Bound
- Graphical Models
- Finite State Machines

The Thirteen Berkeley Dwarfs

Main Algorithms/Challenges in Computing [2]:

- Dense Linear Algebra
- Sparse Linear Algebra
- Spectral Methods
- N-body problems
- Structured grids
- Unstructured grids
- MapReduce (Monte Carlo)
- Combinational Logic
- Graph Traversal
- Dynamic Programming
- Back-track and Branch & Bound
- Graphical Models
- Finite State Machines

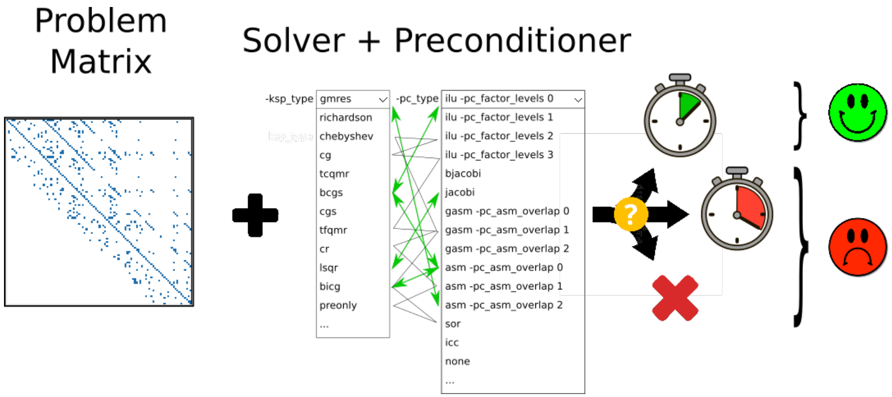
Sparse Linear Algebra: The Solver Selection Problem

We want to solve:

$$A\mathbf{x} = b$$

or, rather:

$$QAP^{-1}(P\mathbf{x}) = Qb$$



We have different choices to make

- direct and iterative solvers
- preconditioning (incl. permutation, scaling, . . .)
- other internal settings (GMRES(r), ILU(k), ASM(k), . . .)
- data structures

To make our lives easier, we use PETSc

Sparse iterative linear solvers

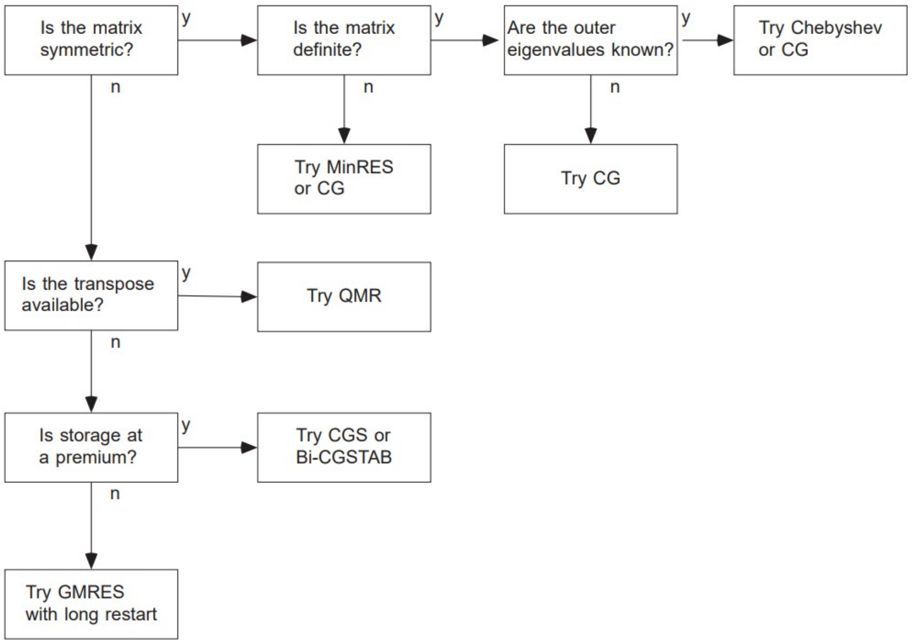


Figure: Flowchart of iterative methods. From [3]

Direct methods have high memory requirements, but iterative methods tend to have trouble with (very) ill-conditioned problems

General guidelines exist

... but no method is the absolute best
... and solvers might be unstable, stagnate, or diverge

Related research

Black-box classifiers & beyond:

- Heuristics – Self-Adapting Large-scale Solver Architecture (SALSA) [4]
- Embeddings – Yeom et al [5]
- Black-box ML – Lighthouse [6]
- Using Neural Networks [7]
- Using Graph-based Machine Learning [8]

Related research

Black-box classifiers & beyond:

- Heuristics – Self-Adapting Large-scale Solver Architecture (SALSA) [4]
- Embeddings – Yeom et al [5]
- Black-box ML – Lighthouse [6]
- Using Neural Networks [7]
- Using Graph-based Machine Learning [8]

Main takeaways:

- A defined set of matrix properties will be the input features
- Regression rather than classification helps contemplate relative performance
- Misclassification or ROC curves don't necessarily convey the impact of a wrong choice
- Embedding can alleviate the impact of unbalanced and limited data
- Preconditioned accuracy might (strongly) differ from actual solver accuracy

Outline

Overview

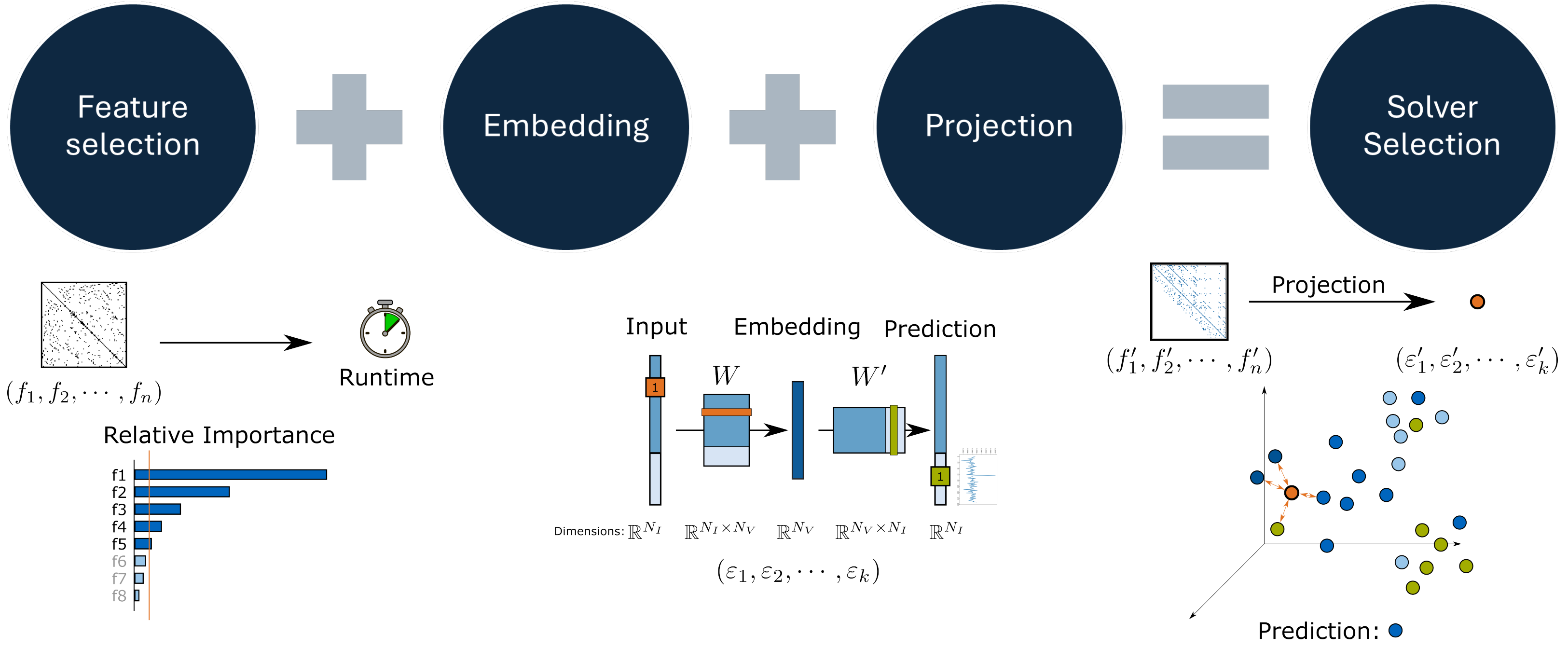


Figure: Overview of the solver selection pipeline

Feature selection

Define a **regression** problem with solver **runtime** as target

- Solve the regression problem with **Gradient Boosting**
- Recover the relative feature importance from the model
- define a cut-off tolerance for features to be included in the final analysis

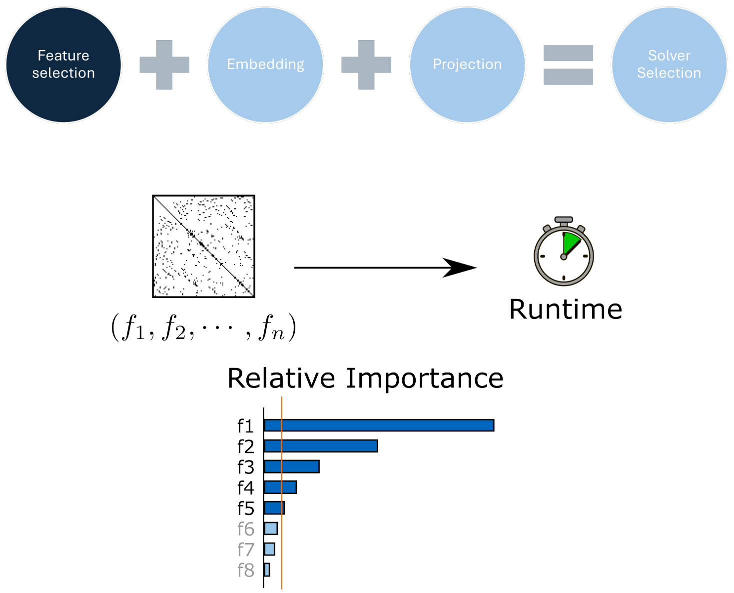


Figure: Selection via regression analysis

Embedding

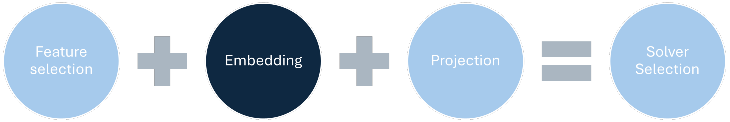
We use *word2vec*'s **skip-gram** model with **negative sampling** based on $(matrix, solver) \to \{good, bad\}$ labeled pairs.

- Consider both matrices and solvers as part of the corpus
- Set 'good' solvers in a matrix's *context*
- Use 'bad' solvers as negative samples
- Hidden layer values will correspond to the embeddings

N_I : Number of Matrices + Solvers

N_V : Number of Embedding Dimensions

$$A \to (\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_k)$$



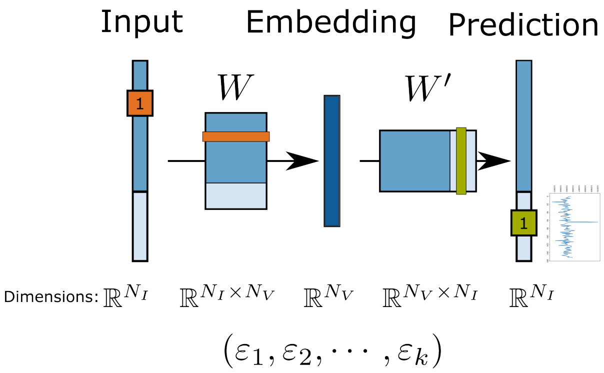
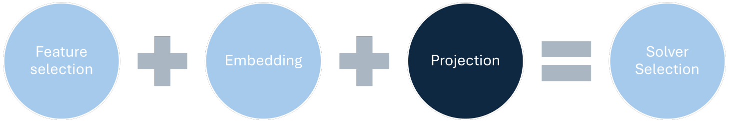


Figure: Embedding using *Word2Vec*

Projection

Test matrices will be out-of-sample, so they don't have an encoding



$\Rightarrow$ Use the training sample features to find a *sparse* **linear combination**

$\Rightarrow$ We do this via **LASSO regression** restricting to positive coefficients

$\Rightarrow$ Use this coefficient vector in the embedding space

$$\min_{\alpha} \|\mathbf{f}' - \mathbf{F}\alpha\|_2^2$$
$$\text{s. t.} \quad \|\alpha\|_1 \leq \delta, \quad \alpha_i \geq 0$$

$\mathbf{f}'$: test sample features
$\mathbf{F}$: training sample set features
δ : target sparsity

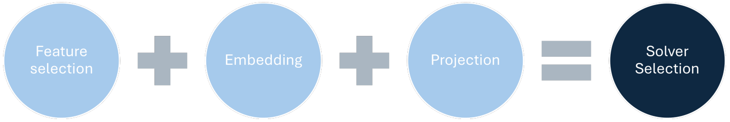
$$\mathbf{F}\alpha \rightarrow \mathbf{E}\alpha$$

$$\mathbf{f}' = (f_1', f_2', \cdots, f_n') \rightarrow (\varepsilon_1', \varepsilon_2', \cdots, \varepsilon_k')$$

Solver Selection

Now that the test sample has an embedding, it suffices to use existing information to predict suitable solvers

We can use e. g., **k-Nearest Neighbors** to determine the preferred solver for the test matrix



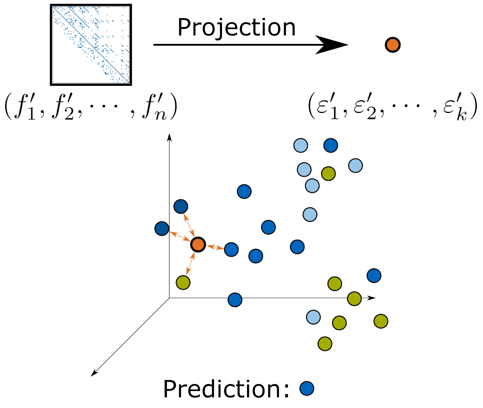


Figure: Prediction based on k-Nearest Neighbors

Outline

Experimental Setup

Performance/runtime measurement data

- SuperMUC-NG Phase 1
  - Intel Skylake Xeon Platinum 8174 processors
  - 48 cores and 96 GB memory per node
- Theta KNL: GNN-dataset from [8]
  - Intel Xeon Phi 7230 processors
  - 64 cores and 96 GB memory per node
- Blue Gene/Q: lighthouse-dataset from [6]
  - Intel Westmere Xeon X5650 processors
  - 12 cores and 72 GB memory per node

Prediction experiments Workstation

- Intel i7-10700 CPU
- NVIDIA Quadro 5000 GPU

Convergence on selected matrices

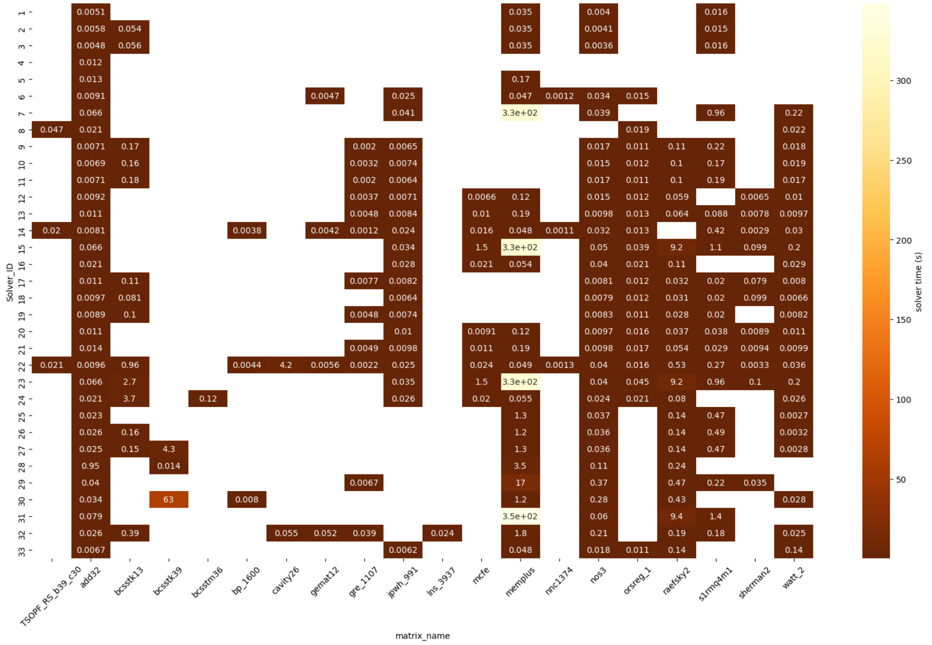


Figure: runtimes on 1 node of SuperMUC for different solvers

Different matrices are resolved effectively by different solvers

Convergence on selected matrices

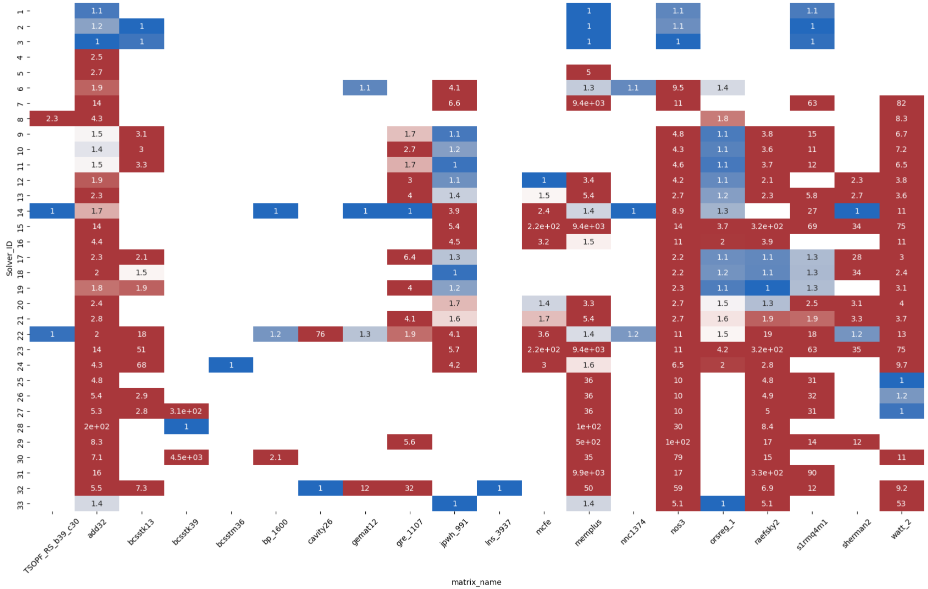


Figure: runtimes on 1 node of SuperMUC for different solvers

Different matrices are resolved effectively by different solvers

Even in cases where many/all solvers converge, runtime can vary (very) wildly

Scaling on selected matrices

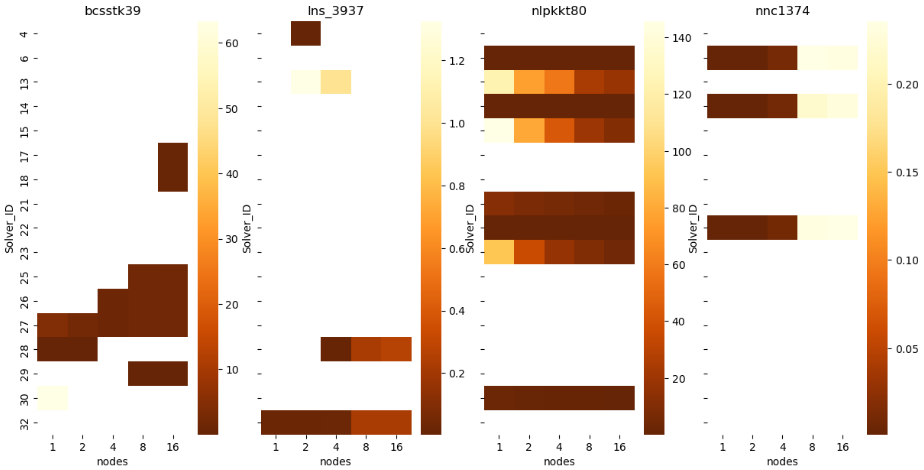


Figure: runtimes on varying numbers of nodes of SuperMUC for different solvers

Scaling on selected matrices

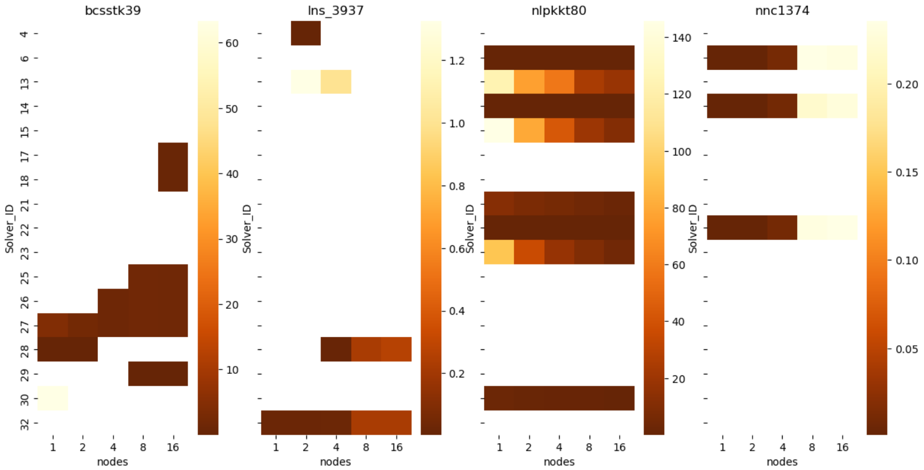


Figure: runtimes on varying numbers of nodes of SuperMUC for different solvers

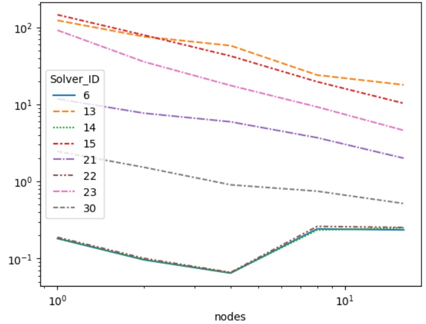


Figure: runtimes vs. node count for nlpkkt80

Experiments: Feature selection

Top features ($R^2 = 0.72$):

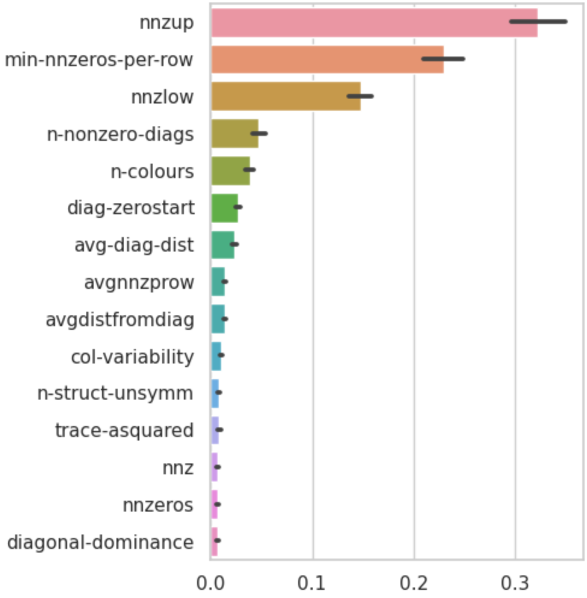


Figure: feature importance

Observations:

- tends to overemphasize size-dependent features since problem size $\propto$ runtime
- features have very distinct ranges (from binary features to some potentially reaching `MaxValue`)

Experiments: Embedding

the SuiteSparse matrix performance data from both Lighthouse [6] (left) and Tang et al [8](right) is embedded into a **20-dimensional** space. Points are colored by best solver

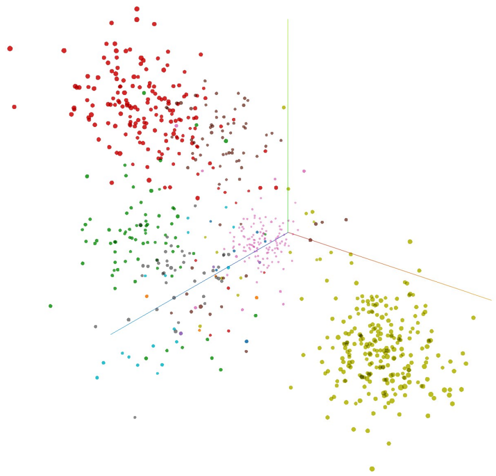


Figure: PCA of embedding for runtime data from Lighthouse [6]

Figure: PCA of the embedding for runtime data from Tang et al [8]

Experiments: Projection and Prediction

Applying the projection requires a regularization parameter which can be predetermined or tuned

Observations:

- the optimal regularization parameter varies strongly based on the data
- normalization or scaling of the properties makes a significant difference between the combination found
- problems of interest will in practice be much larger than the test data

Prediction accuracy:
40%

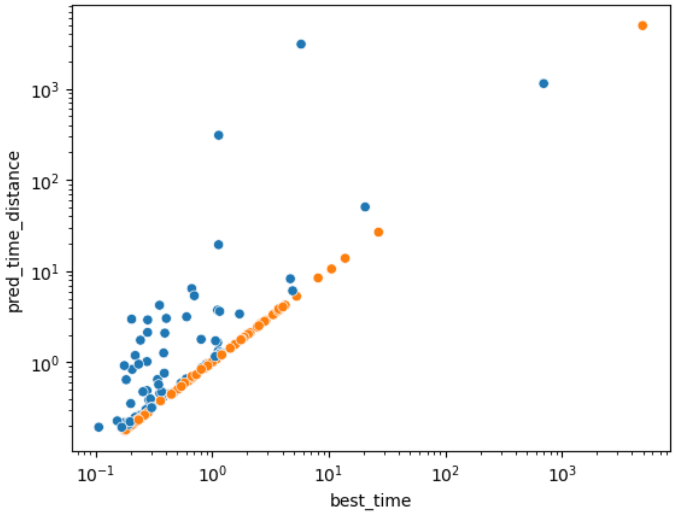


Figure: predicted vs. best runtime

Experiments: Projection and Prediction

Applying the projection requires a regularization parameter which can be predetermined or tuned

Observations:

- the optimal regularization parameter varies strongly based on the data
- normalization or scaling of the properties makes a significant difference between the combination found
- problems of interest will in practice be much larger than the test data

Prediction accuracy:

40% (Misclassification error)

62.5 ± 1.6% (Normalized Discounted Cumulative Gain)

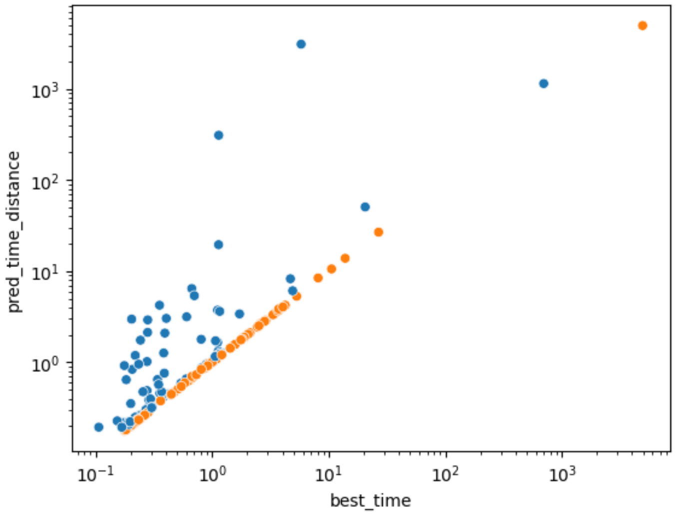


Figure: predicted vs. best runtime

Experiments: Projection and Prediction

Applying the projection requires a regularization parameter which can be predetermined or tuned

Observations:

- the optimal regularization parameter varies strongly based on the data
- normalization or scaling of the properties makes a significant difference between the combination found
- problems of interest will in practice be much larger than the test data

Bonus: Prediction for `ACTIVSg70k_AC` matrices:

- recommends using: BiCGSTAB with Hypre/BoomerAMG
- accurately captures configurations actually solving the problem
- completes in 0.152 s (vs. **0.142 s** for best configuration tried: GMRES + BoomerAMG)

Prediction accuracy:

40% (Misclassification error)

62.5 ± 1.6% (Normalized Discounted Cumulative Gain)

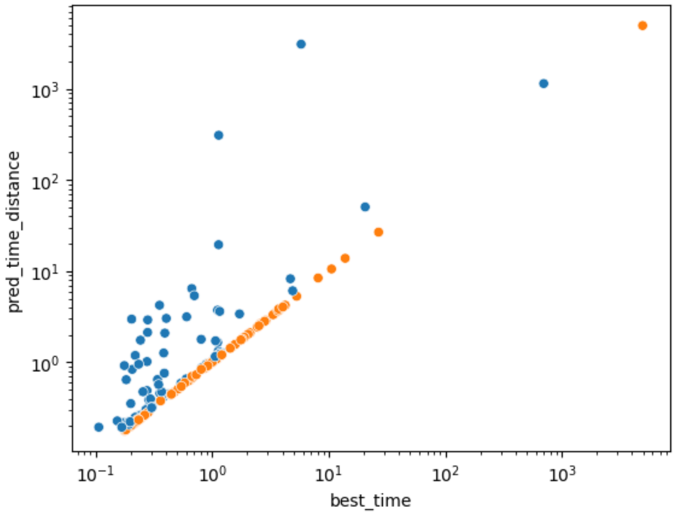


Figure: predicted vs. best runtime

Conclusions & Next steps

- While accuracy is not currently great, recommendations generally in line with common knowledge
- Results can be improved if data is less varied (e.g., sticking to a more restricted set of problems)
- There's lots of tuning possible to enhance the framework: (hyper)parameters, scalings, metrics, models, . . .

Further directions to explore: Now also consider how this all changes with different hardware...

... and add GPUs into the mix...

... and mixed precision...

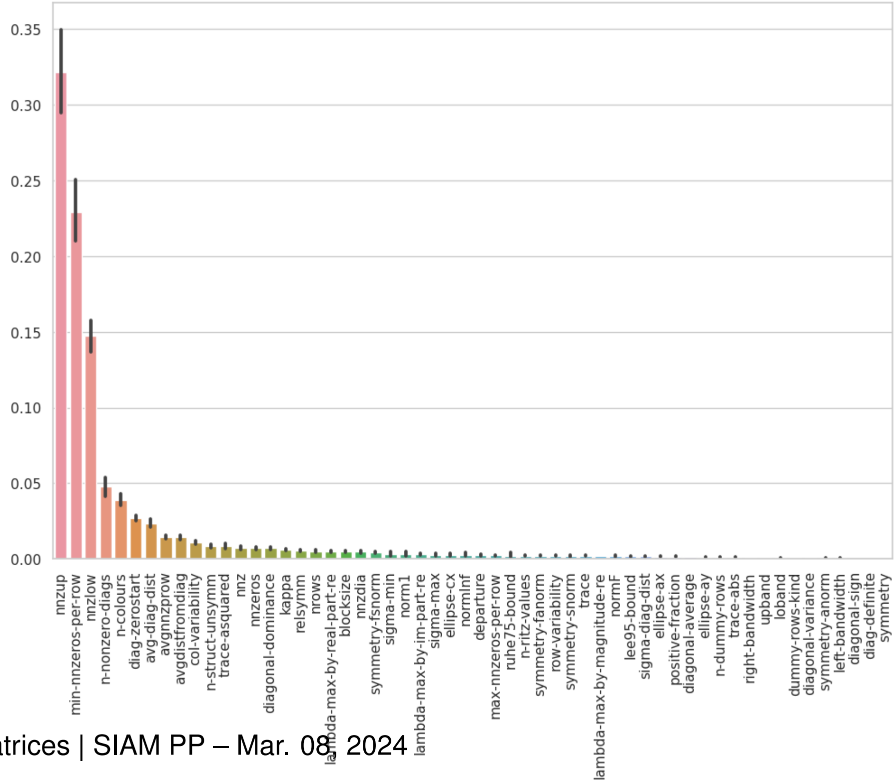
... and batched solves...

... and so much more...

Property list & highlighted features

avgnnzprow	right-bandwidth
avgdistfromdiag	symmetry
n-dummy-rows	blocksize
max-nnzeros-per-row	diag-definite
lambda-max-by-magnitude-im	lambda-max-by-magnitude-re
ellipse-cy	nnzup
ruhe75-bound	avg-diag-dist
nnz	left-bandwidth
lambda-min-by-magnitude-im	lambda-min-by-magnitude-re
norm1	sigma-min
upband	n-struct-unsymm
colours	diagonal-average
diagonal-dominance	dummy-rows
ritz-values-r	symmetry-snorm
symmetry-fanorm	symmetry-fsnorm
lambda-max-by-real-part-im	lambda-max-by-real-part-re
lambda-max-by-im-part-re	lambda-max-by-im-part-im
col-variability	trace-abs
ritz-values-c	nnzeros
diag-zerostart	loband
positive-fraction	trace
min-nnzeros-per-row	diagonal-sign
row-variability	nrows
colour-offsets	n-colours
relsymm	diagonal-variance
departure	nnzlow
n-nonzero-diags	sigma-max
dummy-rows-kind	kappa
n-ritz-values	colour-set-sizes
sigma-diag-dist	symmetry-anorm
ellipse-ax	ellipse-ay
ellipse-cx	lee95-bound
normInf	normF
nnzdia	trace-asquared

Figure: Full feature set. Taken from [9]



References I

- Jack Dongarra, George Bosilca, Zizhong Chen, Victor Eijkhout, Graham E Fagg, Erika Fuentes, Julien Langou, Piotr Luszczek, Jelena Pjesivac-Grbovic, Keith Seymour, et al.
  Self-adapting numerical software (sans) effort.
  IBM Journal of Research and Development, 50(2.3):223–238, 2006.
- Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al.
  The landscape of parallel computing research: A view from berkeley.
  2006.
- Richard Barrett, Michael Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst.
  Templates for the solution of linear systems: building blocks for iterative methods.
  SIAM, 1994.
- Salsa Project.
  Salsa: Self-adapting large-scale solver architecture.

References II

- Jae-Seung Yeom, Jayaraman J Thiagarajan, Abhinav Bhatele, Greg Bronevetsky, and Tzanio Kolev.
  Data-driven performance modeling of linear solvers for sparse matrices.
  In 2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), pages 32–42. IEEE, 2016.
- Kanika Sood, Boyana Norris, and Elizabeth Jessup.
  Lighthouse: A taxonomy-based solver selection tool.
  In Proceedings of the 2nd International Workshop on Software Engineering for Parallel Systems, pages 66–70, 2015.
- Yannick Funk, Markus Götz, and Hartwig Anzt.
  Prediction of optimal solvers for sparse linear systems using deep learning.
  In Proceedings of the 2022 SIAM Conference on Parallel Processing for Scientific Computing, pages 14–24. Society for Industrial and Applied Mathematics, 2022.
- Ziyuan Tang, Hong Zhang, and Jie Chen.
  Graph neural networks for selection of preconditioners and krylov solvers.
  In NeurIPS 2022 Workshop: New Frontiers in Graph Learning, 2022.

References III

Kanika Sood.
Iterative Solver Selection Techniques for Sparse Linear Systems.
PhD thesis, University of Oregon, 2019.