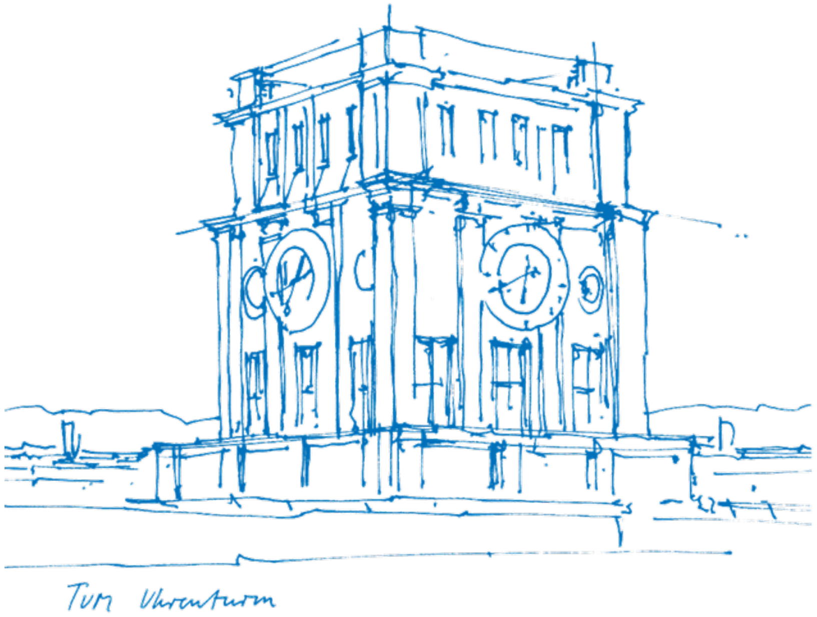
Data-Driven Solver and Preconditioner Selection for Sparse Linear Matrices

International Conference On Preconditioning Techniques For Scientific and Industrial Applications

Hayden Liu Weng Felix Dietrich Hans Joachim Bungartz

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

Atlanta, June 10, 2024



Outline

Scientific Computing application development

Three main roles can be identified:

- field/application expert
- algorithms expert
- optimization expert

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

Scientific Computing application development

Three main roles can be identified:

- field/application expert
- algorithms expert
- optimization expert

One person alone cannot (generally) be expected to fulfill all of the above roles
⇒ The decisions should be decoupled as much as possible using abstractions, code generation, and numerical libraries
⇒ Traditionally, Computing Centers mainly offer support regarding this last aspect

Scientific Computing application development

Three main roles can be identified:

- field/application expert
- algorithms expert
- optimization expert

One person alone cannot (generally) be expected to fulfill all of the above roles
⇒ The decisions should be decoupled as much as possible using abstractions, code generation, and numerical libraries
⇒ Traditionally, Computing Centers mainly offer support regarding this last aspect
⇒ Development should be more approachable *specially* to newcomers, 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, . . .

[1] J. Dongarra, G. Bosilca, Z. Chen, et al., “Self-adapting numerical software (sans) effort,” IBM Journal of Research and Development, vol. 50, no. 2.3, pp. 223–238, 2006.

Outline

Sparse Linear Algebra: The Solver/Preconditioner Selection Problem

We want to solve:

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

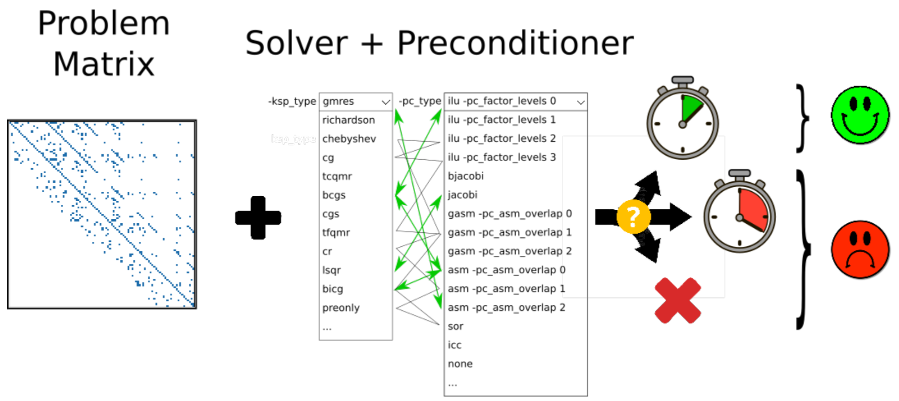
or, rather:

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

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, e.g., PETSc



Quick example:

Properties:

- Num Rows: 4960
- Nonzeros: 19848
- Pattern Symmetry: 100%
- Numeric Symmetry: 30.5%
- Positive Definite: No

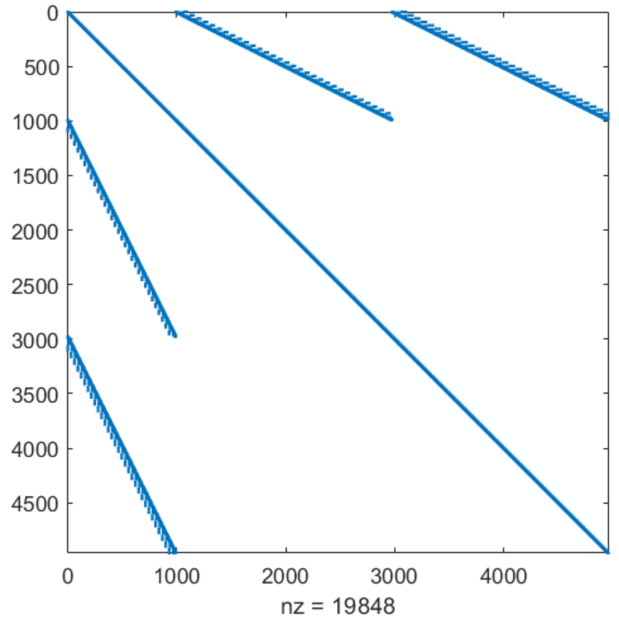


Figure: Sparsity pattern

Quick example: add32

Properties:

- Num Rows: 4960
- Nonzeros: 19848
- Pattern Symmetry: 100%
- Numeric Symmetry: 30.5%
- Positive Definite: No
- Kind: Circuit Simulation Problem
- Cond Num: 136.68

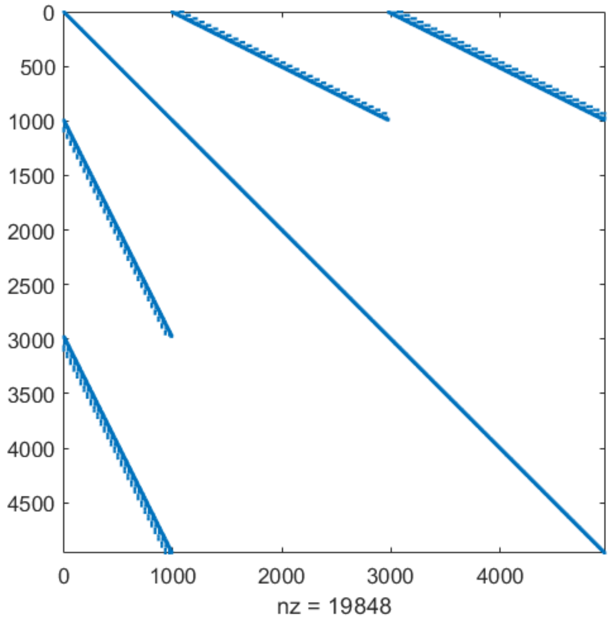


Figure: Sparsity pattern for `add32`

Quick example:

Properties:

- Num Rows: 3937
- Nonzeros: 25407
- Pattern Symmetry: 85%
- Numeric Symmetry: 0.1%
- Positive Definite: No

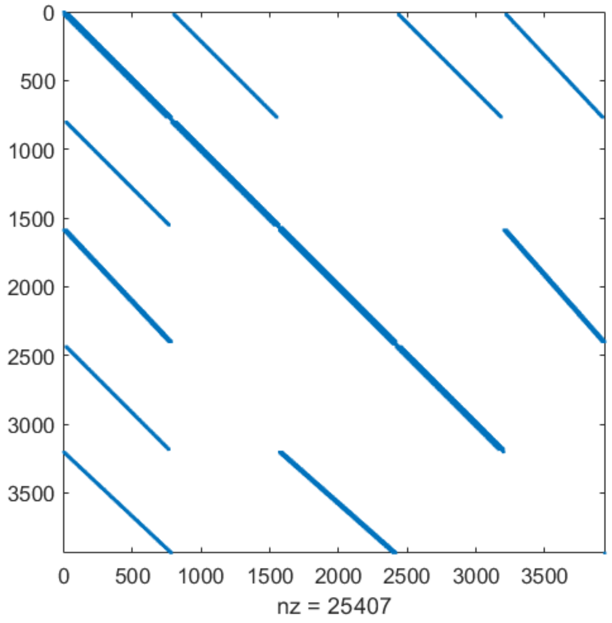


Figure: Sparsity pattern

Quick example: lns_3937

Properties:

- Num Rows: 3937
- Nonzeros: 25407
- Pattern Symmetry: 85%
- Numeric Symmetry: 0.1%
- Positive Definite: No
- Kind: Computational Fluid Dynamics Problem
- Cond Num: 1.9987×10^{16}

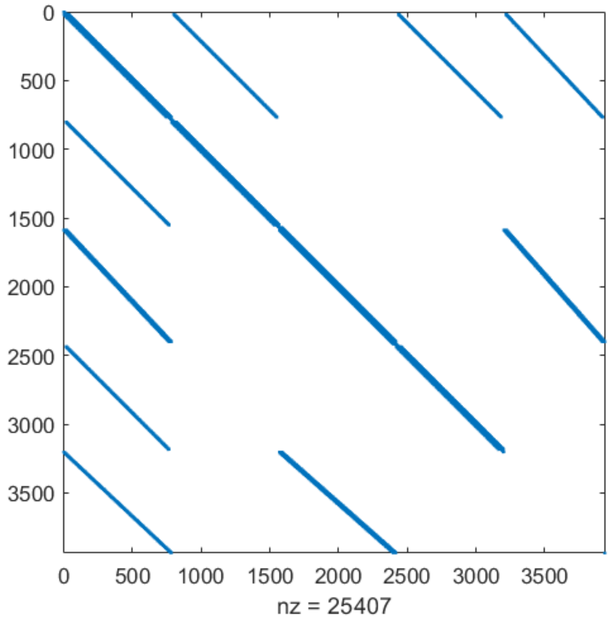


Figure: Sparsity pattern for `lns_3937`

Sparse iterative linear solvers

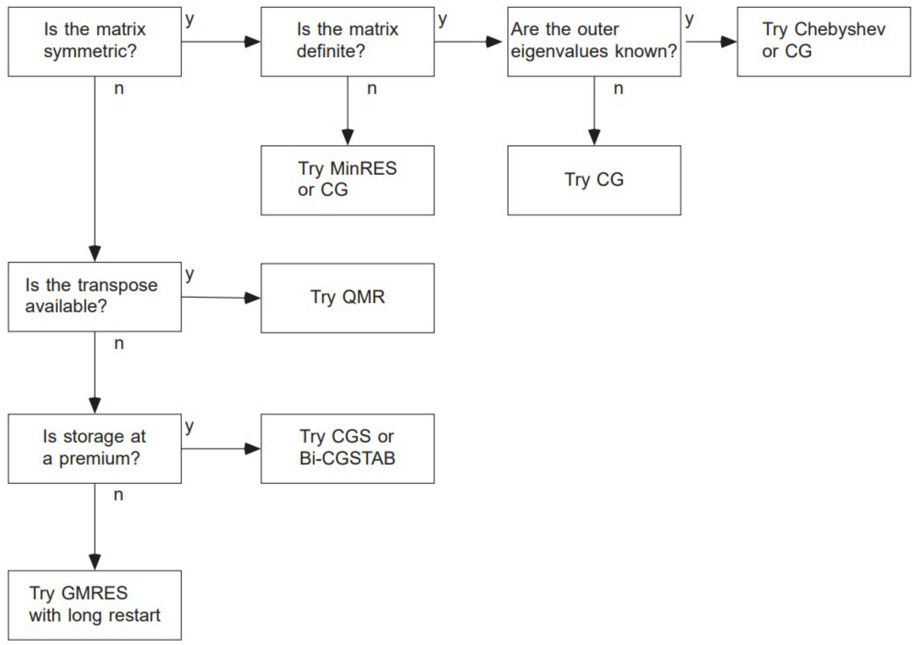


Figure: Flowchart of iterative methods. From Barrett et al. [2]

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) [3]
- Top N Recommender Systems – George et al. [4]
- Embeddings – Yeom et al. [5]
- Black-box ML – Lighthouse [6]
- Neural Networks – Funk [7]
- Graph Neural Networks – Tang et al. [8]

Related research

Black-box classifiers & beyond:

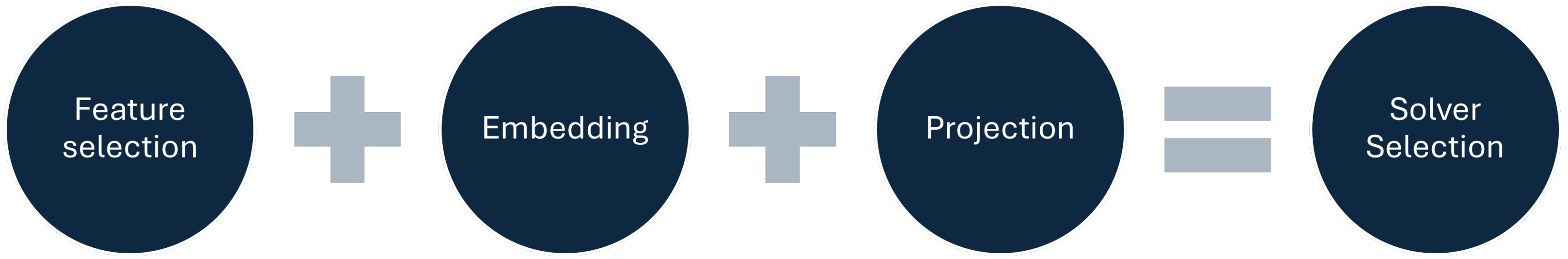
- Heuristics – Self-Adapting Large-scale Solver Architecture (SALSA) [3]
- Top N Recommender Systems – George et al. [4]
- Embeddings – Yeom et al. [5]
- Black-box ML – Lighthouse [6]
- Neural Networks – Funk [7]
- Graph Neural Networks – Tang et al. [8]

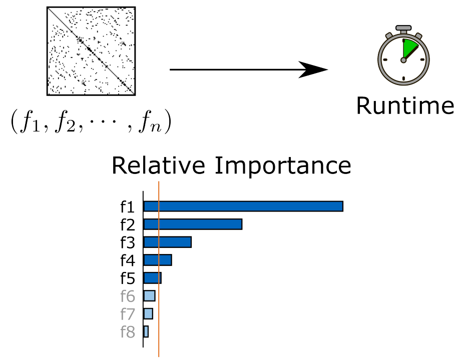
Main takeaways:

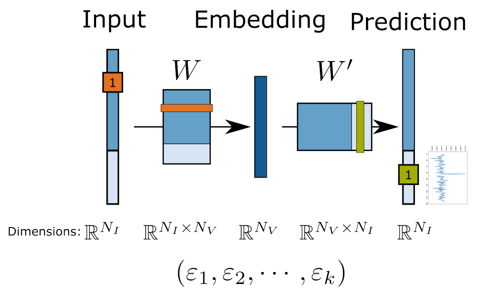
- the input features are a predefined set of matrix properties
- Relative performance is more important than raw classification
- Misclassification or ROC curves don't necessarily convey the impact of a wrong choice
- Embedding can alleviate the impact of unbalanced and limited data
- Preconditioned accuracy might (strongly) differ from actual solver accuracy

Outline

Overview






Input
Embedding
Prediction
W
W'
1
1
Dimensions: $\mathbb{R}^{N_I}$
$\mathbb{R}^{N_I \times N_V}$
$\mathbb{R}^{N_V}$
$\mathbb{R}^{N_V \times N_I}$
$\mathbb{R}^{N_I}$
$(\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_k)$

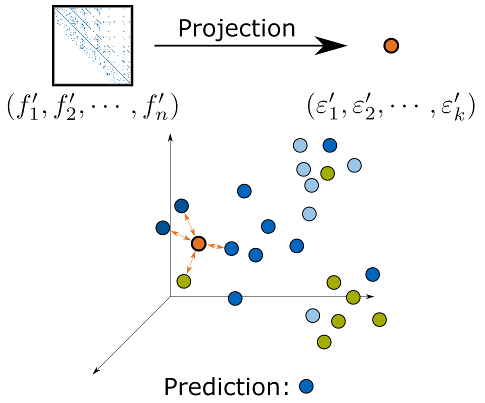
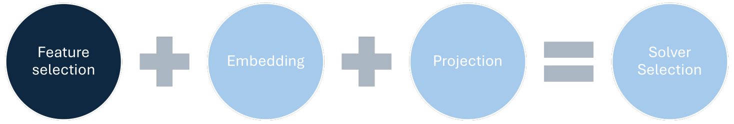

Projection
$(f'_1, f'_2, \cdots, f'_n)$
$(\varepsilon'_1, \varepsilon'_2, \cdots, \varepsilon'_k)$
Prediction: ●

Figure: Overview of the solver selection pipeline

Feature selection: Reduce required computations

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



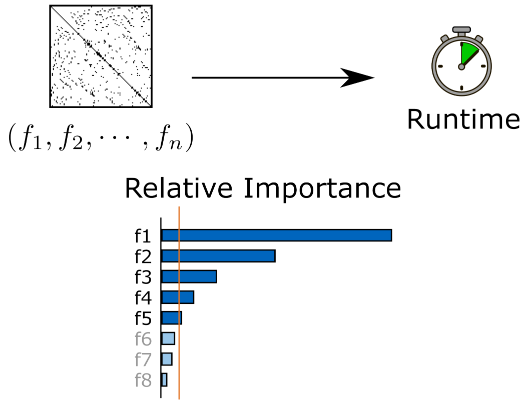
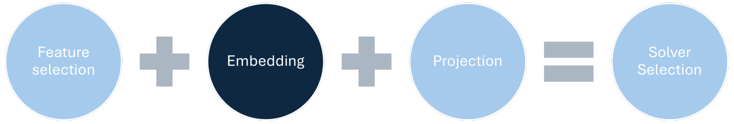


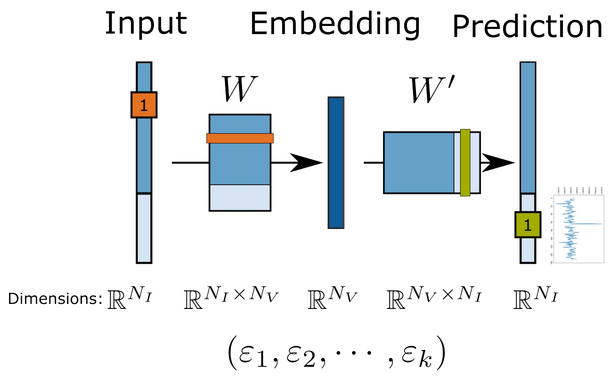
Figure: Selection via regression analysis

Embedding: Cluster matrices together

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





Input
Embedding
Prediction
W
W'
1
1
Dimensions: $\mathbb{R}^{N_I}$ $\mathbb{R}^{N_I \times N_V}$ $\mathbb{R}^{N_V}$ $\mathbb{R}^{N_V \times N_I}$ $\mathbb{R}^{N_I}$
$(\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_k)$

Figure: Embedding using *Word2Vec*

N_I :	Number of Matrices + Solvers
N_V :	Number of Embedding Dimensions

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

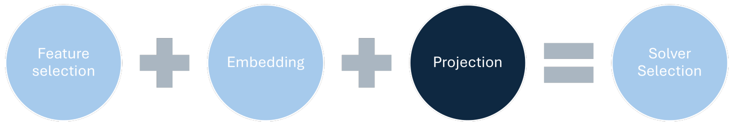
Projection: Add the new data

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

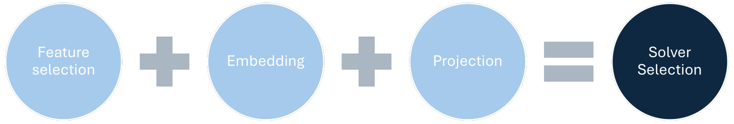
$$\mathbf{F}\alpha \rightarrow \mathbf{E}\alpha$$

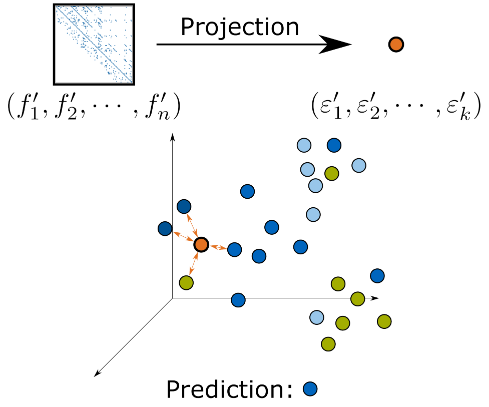
$$\mathbf{f}' = (f_1', f_2', \cdots, f_n') \rightarrow (\varepsilon_1', \varepsilon_2', \cdots, \varepsilon_k')$$

Solver and Preconditioner 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





Projection
$(f'_1, f'_2, \cdots, f'_n)$
$(\varepsilon'_1, \varepsilon'_2, \cdots, \varepsilon'_k)$
Prediction:

Figure: Prediction based on k-Nearest Neighbors

Outline

System Setup & Datasets

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 provided by H. Zhang [8]
  - Intel Xeon Phi 7230 processors
  - 64 cores and 96 GB memory per node
- Blue Gene/Q: Lighthouse-dataset from Sood et al. [6]
  - Intel Westmere Xeon X5650 processors
  - 12 cores and 72 GB memory per node

Prediction experiments Workstation with:

- Intel i7-10700 CPU
- NVIDIA Quadro 5000 GPU

Convergence on selected matrices

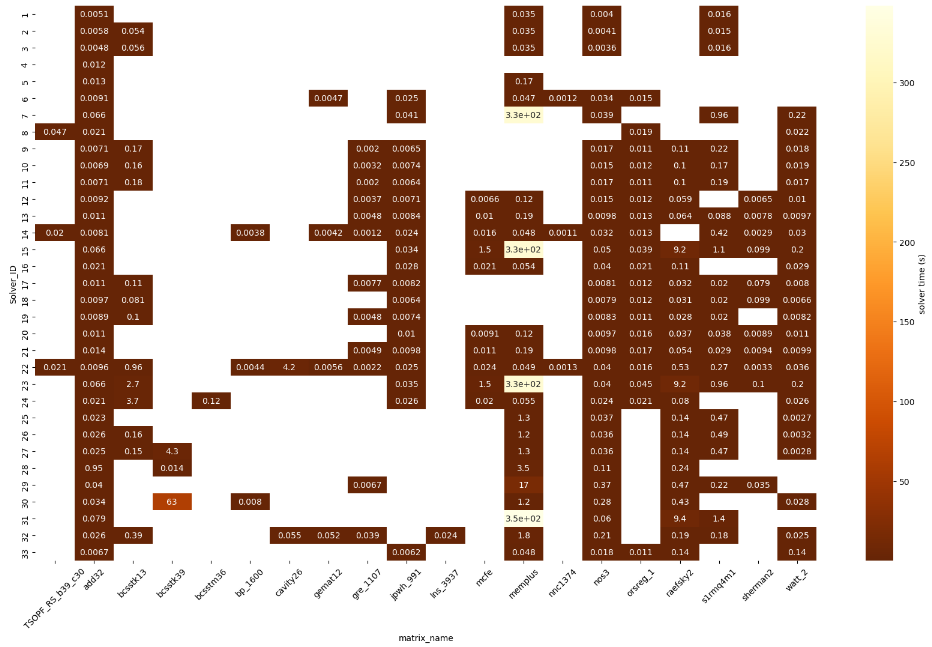


Figure: runtimes on 1 node of SuperMUC for different solvers

Different matrices are resolved effectively by different solvers

Convergence on selected matrices

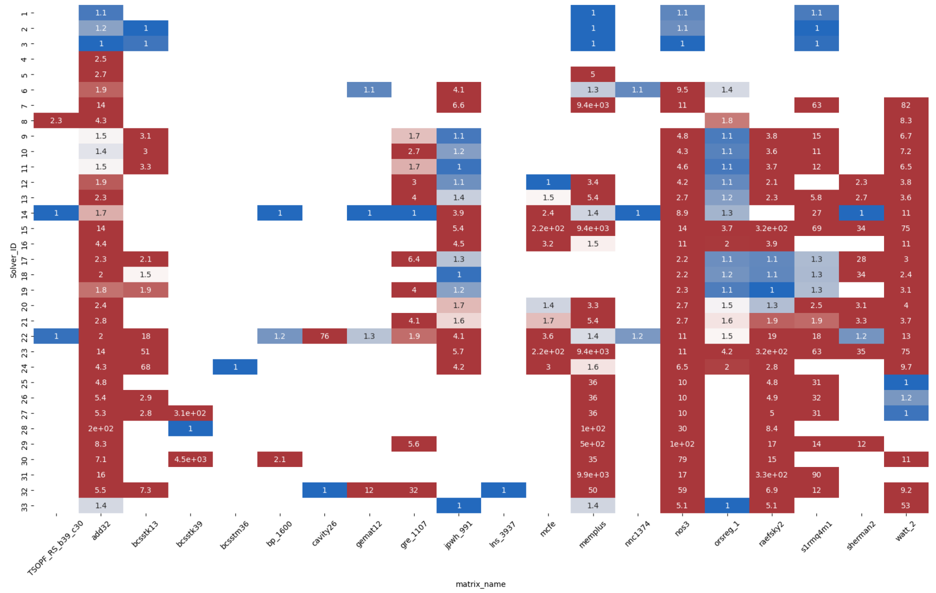


Figure: runtimes on 1 node of SuperMUC for different solvers

Different matrices are resolved effectively by different solvers

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

Experiments: Feature selection

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

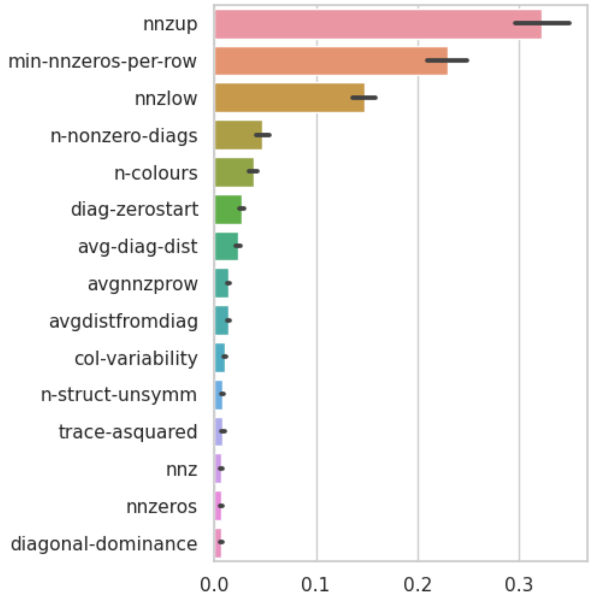


Figure: feature importance

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

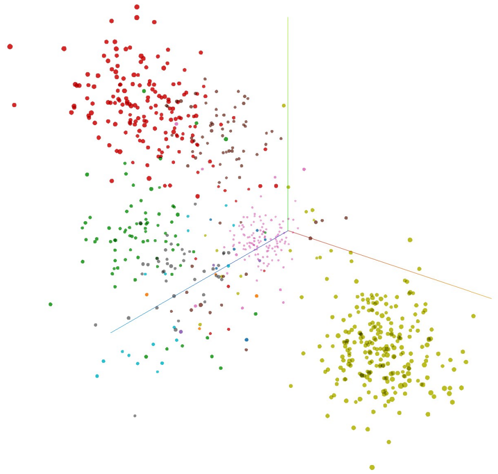


Figure: PCA of embedding for performance data for Lighthouse-dataset

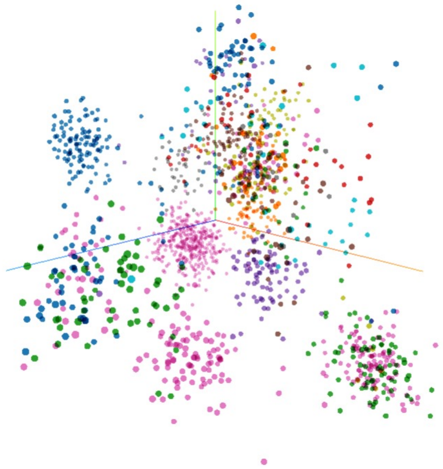


Figure: PCA of embedding for performance data for GNN-dataset

Experiments: Projection and Prediction

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

Observations:

- the optimal regularization parameter varies strongly
- normalization of the properties makes a significant difference between the combination found
- problems of interest will in practice be much larger than the training datasets

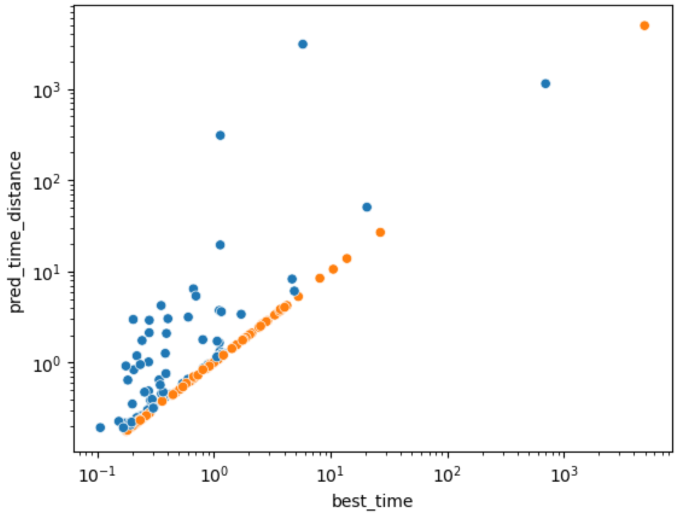
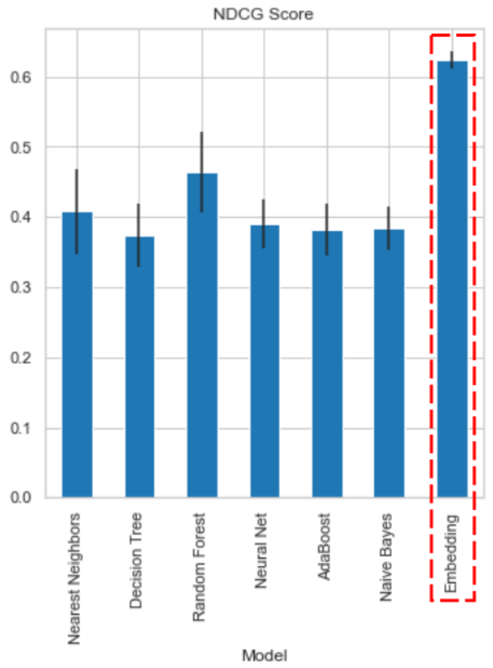


Figure: predicted vs. best runtime

Experiments: Comparison vs. Simple Classifiers



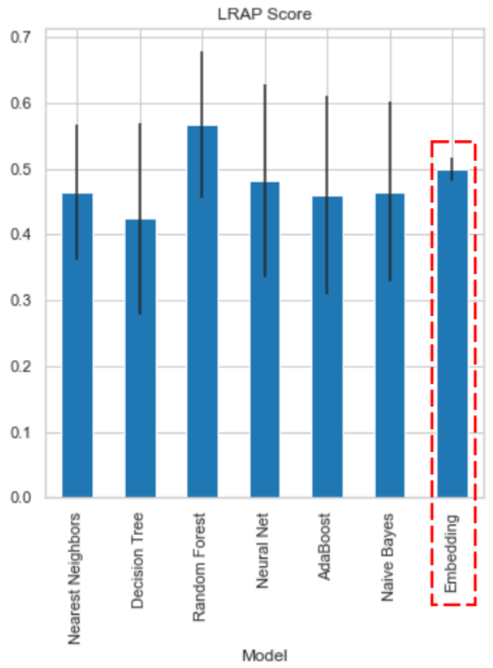


Figure: Score Comparison across different ML methods

Conclusions & Next steps

- NDCG accuracy is ca. 15% better, LRAP is similar
- Variance is however lower than in classical ML methods
- Still many tuning possible to enhance the framework: (hyper)parameters, scaling, metrics, models, . . .
- While building the Embedding is costly, once trained prediction is relatively inexpensive

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...

References I

[1] J. Dongarra, G. Bosilca, Z. Chen, et al., “Self-adapting numerical software (sans) effort,” IBM Journal of Research and Development, vol. 50, no. 2.3, pp. 223–238, 2006.

[2] R. Barrett, M. Berry, T. F. Chan, et al., Templates for the solution of linear systems: building blocks for iterative methods. SIAM, 1994.

[3] Salsa Project., Salsa: Self-adapting large-scale solver architecture, [Online]. Available: `http://icl.cs.utk.edu/salsa/`.

[4] T. George, A. Gupta, and V. Sarin, “A recommendation system for preconditioned iterative solvers,” in 2008 Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 803–808.

[5] J.-S. Yeom, J. J. Thia-garajan, A. Bhatele, G. Bronevetsky, and T. 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 Syster IEEE, 2016, pp. 32–42.

[6] K. Sood, B. Norris, and E. Jessup, “Lighthouse: A taxonomy-based solver selection tool,” in Proceedings of the 2nd International Workshop on Software Engineering for Parallel Systems, 2015, pp. 66–70.

References II

[7] Y. Funk, M. Götz, and H. 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, Society for Industrial and Applied Mathematics, 2022, pp. 14–24.

[8] Z. Tang, H. Zhang, and J. Chen, “Graph neural networks for selection of preconditioners and krylov solvers,” in NeurIPS 2022 Workshop: New Frontiers in Graph Learning, 2022.

[9] K. Sood, “Iterative solver selection techniques for sparse linear systems,” Ph.D. dissertation, University of Oregon, 2019.

LRAP and NDCG scores

For $y_{\text{true}} \in \{0,1\}^{n_{\text{labels}} \times n_{\text{samples}}}$ and $\hat{y}$ prediction probabilities,

LRAP:

$$LRAP(y_{\text{true}}, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}} \left(\frac{1}{\|y_i\|_0} \sum_{j : y_{ij}=1} \frac{|\mathscr{L}_{ij}|}{\text{rank}_{ij}} \right)$$

$\mathscr{L}_{ij} := \{k : y_{ik} = 1, \hat{y}_{ik} > \hat{y}_{ij}\}$
$\text{rank}_{ij} := \big|\{k : \hat{y}_{ik} > \hat{y}_{ij}\big|\}$

NDCG:

$$\text{NDCG} = \frac{\text{DCG}(y_{\text{true}}, \hat{y})}{\text{DCG}(y_{\text{true}}, y_{\text{true}})}$$

where

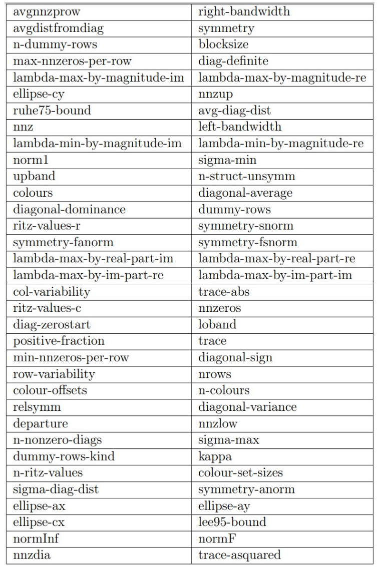
$$\text{DCG}(y_{\text{true}}, \hat{y}) = \sum_i \frac{y_{\hat{\sigma}(i)}}{\log(1+i)}$$

and $y_{\hat{\sigma}(i)}$ is some ranking function for y_{true} and $\hat{y}$.

Both metrics focus on the ranking of “good” classes

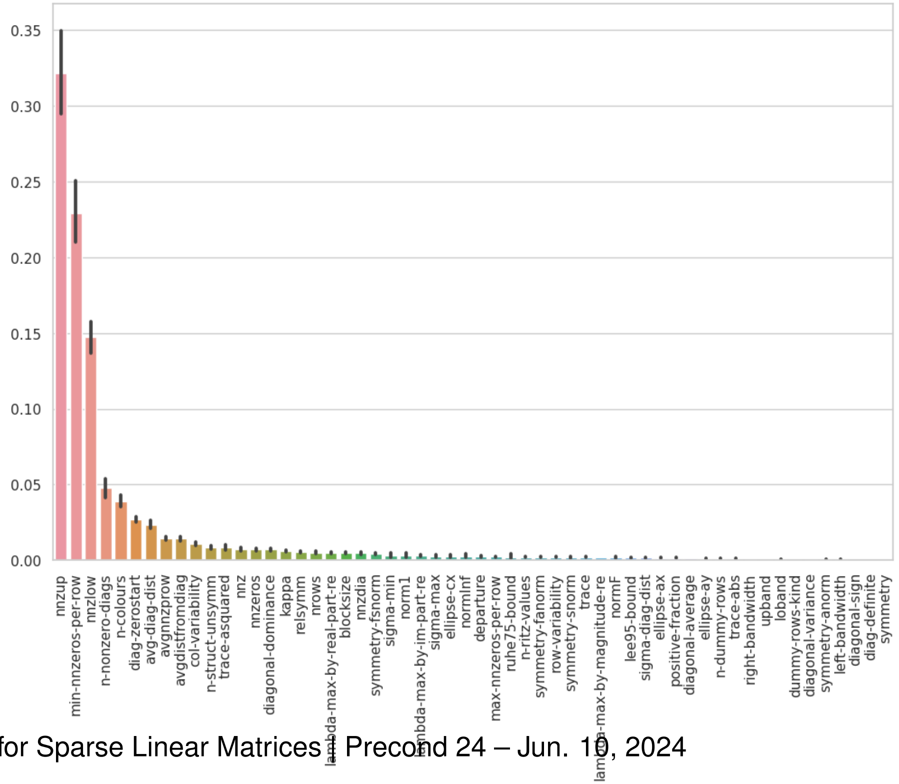
LRAP penalizes more heavily for low rankings of good classes

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

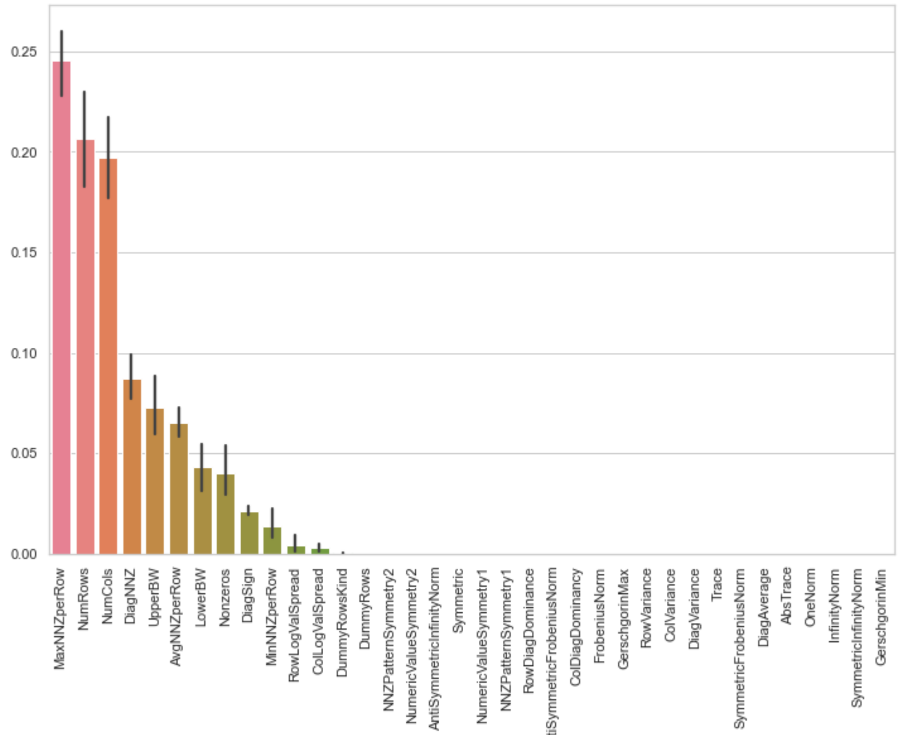
Figure: Full feature set. Taken from [9]



Property list without kappa & highlighted features

NumRows	DummyRows
NumCols	DummyRowsKind
RowVariance	NumericValueSymmetry1
ColVariance	NNZPatternSymmetry1
DiagVariance	NumericValueSymmetry2
Nonzeros	NNZPatternSymmetry2
FrobeniusNorm	RowDiagDominance
SymmetricFrobeniusNorm	ColDiagDominancy
AntiSymmetricFrobeniusNorm	DiagAverage
OneNorm	DiagSign
InfinityNorm	DiagNNZ
SymmetricInfinityNorm	LowerBW
AntiSymmetricInfinityNorm	UpperBW
MaxNNZperRow	RowLogValSpread
Trace	ColLogValSpread
AbsTrace	Symmetric
MinNNZperRow	GerschgorinMax
AvgNNZperRow	GerschgorinMin

Table: Alternative feature set without kappa



Scaling on selected matrices

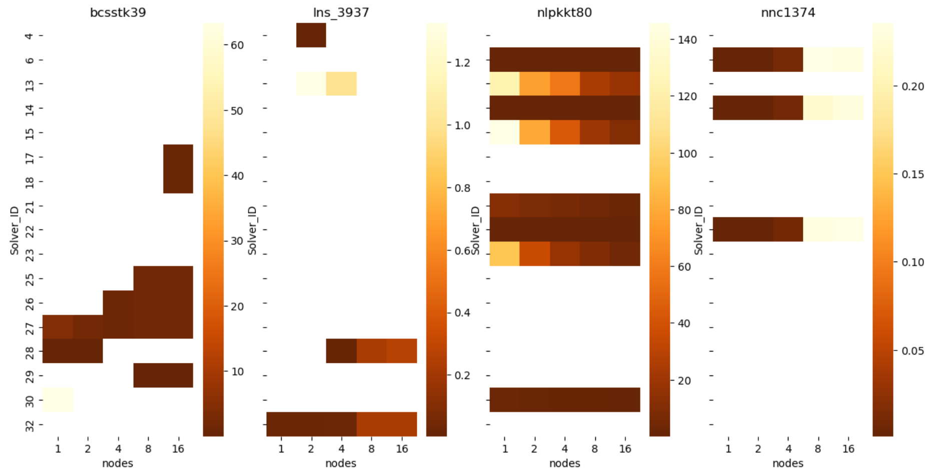


Figure: runtimes on varying numbers of nodes of SuperMUC for different solvers

Scaling on selected matrices

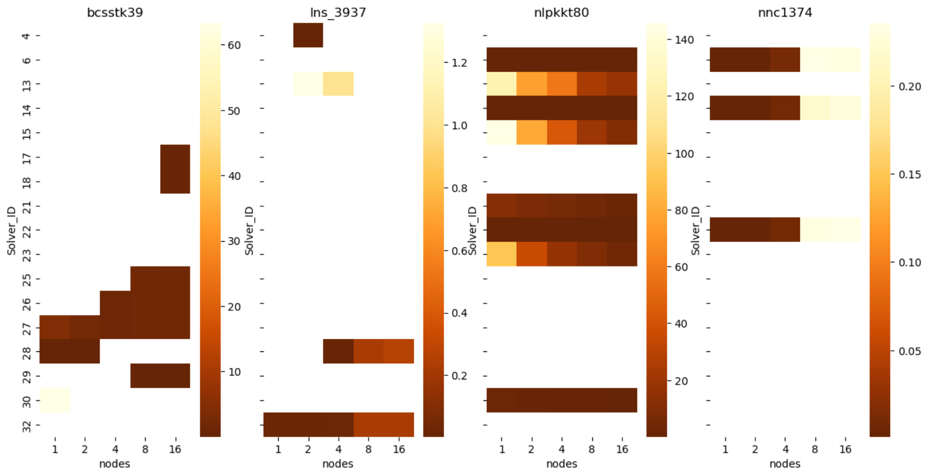


Figure: runtimes on varying numbers of nodes of SuperMUC for different solvers

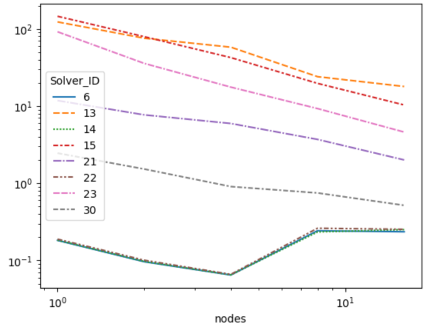


Figure: runtimes vs. node count for nlpkkt80