Utilizing Software Features for Algorithm Selection

Damir Pulatov, Lars Kotthoff

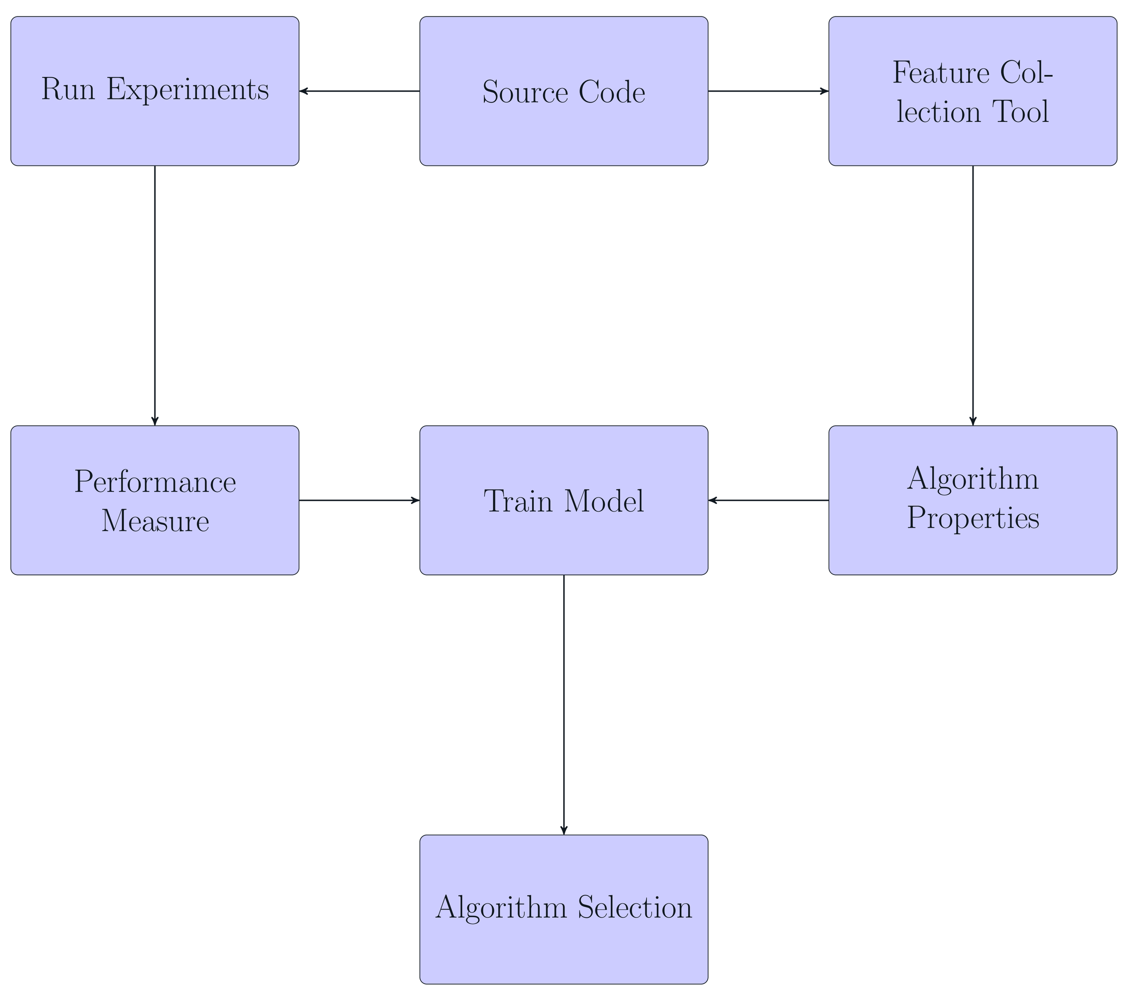
Department of Computer Science, University of Wyoming

dpulatov@uwyo.edu,larsko@uwyo.edu

Motivation

- Goal – investigate whether algorithm selection can be improved if we utilize algorithm features along with instance features
- This iteration of the project uses static algorithm (software) features collected automatically
- Advantage – the number of performance models is constant no matter how many algorithms are used in scenario

Setup



- Performed experiments on seven ASlib scenarios (SAT11-INDU, OPENML-WEKA-2017, etc) as well as scenarios not currently in ASlib (SAT-2018)
- Created SAT18-EXP scenario using main track results with 400 benchmark instances from SAT 2018 competition. Converted data into ASlib format using scripts from COSEAL's aslib-spec repository
- SAT18-EXP has all solvers that participated in the competition, except for varisat since it was written in Rust (software metrics tool does not work)
- Obtained SAT18-EXP's instance features with SATzilla's feature collection tool[1]
- Modified scenarios due to lack of source code, ambiguity in solvers, lack of ability to take into the account parameter settings, and repeated runs
- Automatically collected algorithmic features for solvers written in C++ and Java such as cyclomatic complexity (average and total), maxindent complexity (average and total), lines of code (average and total), size in bytes (average and total), and number of files[2]
- Collected algorithmic features by selecting more relevant pieces of code (e.g., ignored code responsible for parallelism and certificate generation whenever possible)

Setup (cont.)

- Trained all models on Teton High-Performance Computing cluster[3]
- Combined software and instance features by constructing $n \times m$ dataframe, where n is the number of instances times number of solvers, and m is the number of instance and software features
- Utilized server scripts from aslib-r[4] for tuning hyperparameters for individual models. Tuning for combined models was done similarly (e.g., nested cross-validation and so on).

Results

- Combined model is a Random Forest regression model that utilizes both instance and software features
- Individual model is the standard model that uses instance features only
- Models with pair regression method available in LLAMA[5] were also used to see if combined regression model performs better than a slightly modified individual model
- **mcp** and **par10** gaps show the normalized fraction of the gap closed by different methods
- A value of 0 corresponds to the single best solver and a value of 1 to the virtual best. Negative values indicate performance worse than the single best solver
- OPENML was grayed out for par10 table since this metric does not make sense for the scenario

mcp gap

	combined regression	individual pair regression	individual regression
TSP−LION2015	**0**	−0.17	−0.11
SAT18−EXP	0.59	0.6	**0.61**
SAT11−RAND	0.88	**0.92**	0.91
SAT11−INDU	**0.41**	0.39	0.31
SAT11−HAND	0.58	**0.67**	0.64
SAT03−16_INDU	0.46	**0.5**	0.44
OPENML−WEKA−2017	−0.42	**0.22**	−0.51
GRAPHS−2015	0.45	**0.62**	0.56

par10 gap

	combined regression	individual pair regression	individual regression
TSP−LION2015	**0**	−0.42	−0.22
SAT18−EXP	0.29	0.29	**0.32**
SAT11−RAND	0.46	**0.49**	0.48
SAT11−INDU	**0.13**	0.12	0.05
SAT11−HAND	0.24	**0.28**	**0.28**
SAT03−16_INDU	**0.13**	**0.13**	0.12
OPENML−WEKA−2017			
GRAPHS−2015	0.12	**0.19**	0.15

(Color scale: −1.0 −0.5 0.0 0.5 1.0)

Summary

- Building algorithm selection models with current static features produces mixed and inconsistent results
- Some scenarios (OPENML) are improved, some stay about same (SAT01-16_INDU), and others worsen (SAT11-RAND)
- Performing pair regression with instance features gives a much larger improvement on some scenarios compared to combined regression model

Future Work

- Build pair regression models that use both software and instance features to see if they perform any better (currently running experiments)
- Perform feature selection (forward and backward) to find out which software features will be filtered out
- Investigate better static algorithmic features (a lot of minisat hacked solvers have very similar values).
- Take into the account data structures and Object-Oriented properties
- Collect dynamic algorithmic features that characterize only the parts of software that were executed during runtime (stack trace)
- Find a way to automatically analyze more relevant pieces of source code related to computation (e.g., ignore code used for GUIs and so on)
- Add feature costs for algorithm properties

References

[1] L Xu et al. "SATzilla2012: Improved algorithm selection based on cost-sensitive classification models". In: *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions* (Jan. 2012), pp. 55–58.

[2] *Metrix++ is a tool to collect and analyse code metrics.* URL: https://metrixplusplus.github.io/home.html.

[3] *Advanced Research Computing Center (2018) Teton Computing Environment, Intel x86_64 cluster. University of Wyoming, Laramie, WY.* URL: https://doi.org/10.15786/M2FY47.

[4] Bernd Bischl et al. "ASlib: A benchmark library for algorithm selection". In: *Artif. Intell.* 237 (2016), pp. 41–58. DOI: 10.1016/j.artint.2016.04.003. URL: https://doi.org/10.1016/j.artint.2016.04.003.

[5] Lars Kotthoff. *LLAMA: Leveraging Learning to Automatically Manage Algorithms.* Tech. rep. arXiv:1306.1031. arXiv, June 2013. URL: http://arxiv.org/abs/1306.1031.