OASC-2017: *Zilla Submission

Chris Cameron CCHRIS13@CS.UBC.CA
Holger H. Hoos HOOS@CS.UBC.CA
Kevin Leyton-Brown KEVINLB@CS.UBC.CA
Department of Computer Science, University of British Columbia, Canada

Frank Hutter FH@CS.UNI-FREIBURG.DE
Department of Computer Science, University of Freiburg, Germany

Editors: Marius Lindauer, Jan N. van Rijn and Lars Kotthoff

Abstract

*Zilla is a model-based approach for algorithm selection and the most recent iteration of the well-known SATzilla project. The new *Zilla system has increased flexibility for the user and is configurable to run with many machine learning models and alternatives for various pre/post processing steps (e.g., presolver selection, feature completion prediction, and solver subset selection). The main additions to our *Zilla pipeline are automated procedures for feature group selection, hyper-parameter tuning, and solver subsampling prior to model building. We submit two versions for the competition that are equivalent except for the choice of per-instance machine learning model. For our first submission, we use a weighted pairwise random forest classifier. For our second submission, we test an experimental approach that offline, builds a weighted pairwise random forest classifier and online, finds the nearest instances based on the average path lengths across trees and optimizes a schedule over those instances.

1. Introduction

Our lab has been actively working on automated methods for constructing portfolio-based algorithm selectors. Since 2007, our portfolio-based SAT solver, SATzilla (Xu et al., 2008), has won many medals in the SAT competitions and SAT challenges (Le Berre et al., 2015). It was also the winner of the 2015 ICON algorithm selection challenge (Kotthoff et al., 2017).

First, we provide an overview of the pipeline which both of our submissions share and give special attention to new functionality. Second, we describe the machine learning models used for our two submissions. Finally, we describe the computational resources used to train our systems.

2. Overview of system

We introduce the pipeline for our submissions by enumerating the most important steps in the *offline* training phase and *online* execution phase. We pay special attention to new functionality and components that have not been discussed in the literature (marked with boldface). See Xu et al. (2008) for a detailed explanation of the core ideas underlying *Zilla.

Note that there was a critical bug in our submission within the solver sub-sampling procedure that substantially degraded performance. We highlight that bug when we describe that procedure below.

2.1. Key Offline Steps

1. Maximize/Minimize: *Zilla minimizes performance and performance measures are negated if the scenario has a maximization objective.
2. **Solver sub-sampling before model building**: Start with empty set of solvers and greedily add the solver to the set that minimizes the marginal improvement in VBS performance. The VBS (*Virtual Best Solver*) represents the oracle solver that always picks the best solver to run. Continue until performance stagnates and the fraction of improvement drops below a fixed constant. **Bug in submission: constant was misspecified due to accidental rounding when written to file, causing only two solvers to be selected for every scenario.
3. Cheap Features: Choose a set of "cheap" features that will always be computed. (by default, features within the cheapest feature group on average)
4. Build Feature Completion Model: Build a binary random forest prediction model from cheap features to predict whether features are computable within a fixed cutoff time (10% of algorithm cutoff time).
5. Split instances: Randomly partition the training and validation set with 2/3, 1/3 split.
6. Backup Solver: Select a static backup solver that has the best average performance across all training instances without complete features (i.e., instances where all features were not successfully computed before static feature cutoff time).
7. Model building: Build the machine learning model for per-instance selection on all training instances that have complete features.
8. **Presolving**: Find schedule of presolvers to be run prior to feature computation to prevent expensive features from being computed on easy instances. Automatic greedy presolver scheduling (Streeter and Golovin, 2009) is applied for a grid of presolving cutoffs. Select the presolver cutoff based on performance of running *Zilla online on the validation set.
9. Build entire pipeline again over the union of training and validation instances with chosen presolving schedule.
10. **Automatic Feature Group Selection**: Sort the feature groups in order of average runtime. If a feature group in the sorted list has dependencies outside of its preceding feature groups, move through the list and re-index the feature group to the nearest position where its dependencies are satisfied. For all sets of features represented by all prefixes of the sorted feature groups, run steps 4 until 9 for 10 CV folds. Pick feature groups with best 10-fold CV performance.
11. **Hyper-parameter tuning**: For log-scaled grid of random forest hyper-parameter settings, run steps 4 until 9 for 10 CV folds and select the best hyper-parameters based on 10-fold CV.

While the addition of feature group selection and hyper-parameter tuning can be expensive, two of our changes have substantially reduced training time and have never been observed to degrade performance. First, solver sub-sampling before model building substantially reduces training time especially when using random forests (where training time scales quadratically with the number of solvers). Second, a separate model is no longer built conditioned on each presolving schedule. Instead, presolving schedules are evaluated with the same unconditioned model.

2.2. Key Online Steps

1. Execute a statically defined presolving schedule. Terminate if presolvers solve instance.
2. Compute cheap features.
3. Predict whether features will run successfully from cheap features. If the features were predicted to not be computable, run the backup solver.
4. If the features were predicted to be computable, the features are retrieved for the instance.
5. If the features have successfully been computed within the feature cutoff time, impute any missing features to the mean, and query the model for a solver/schedule to run.
6. Otherwise, run the backup solver.

3. Model for Submission 1

As in (Xu et al., 2012), *Zilla uses a pairwise cost-sensitive random forest classifier. It is pairwise because between every pair of algorithms there is a random forest classifier. Each pairwise model outputs a vote for one of two solvers and the solver is selected with the most votes aggregated over all pairwise models. Gini impurity is minimized at every split in each decision tree, where each instance is weighted by the difference in running times between the solvers. Bootstrap sampling of instances is used for every tree.

The parameters `split-min` and `split-features` are exposed to hyper-parameter tuning (`split-min` parameterizes the minimum number of instances in a splittable node and `split-features` parameterizes how many features to randomly sample for each split).

4. Model for Submission 2

Inspired by Lindauer et al. (2016), we attempt an exploratory dynamic scheduling approach while exploiting the use of our high-performance random forests. A pairwise cost-sensitive random forest is built from the training data offline just as in Submission 1 and for a test instance online:

1. For all training instances and all trees across the pairwise random forests, find the distance between the leaf of the test instance and the leaf of the training instance.
2. The leaf of an instance within a particular tree is found by traversing down the tree with the instance features.
3. Distance between leaves is measured as path length to the nearest common ancestor node.

4. Select the k training instances which have the smallest distance to the test instance averaged over all trees among the pairwise random forests.
5. Use the approach by Streeter and Golovin (2009) to greedily find a schedule optimized over those k *nearest* instances.
6. k is exposed to hyper-parameter tuning.

We find this method often performs better and we use it in submission 2 in cases where it performs better than submission 1 based on 10-fold CV performance. (Note that scheduling approaches are not useful for the solution quality objectives). One potential drawback of this system in practice is the online cost of finding the close-by instances and building a schedule. The competition settings are to our advantage since the performance metric does not count the runtime cost from querying the model online.

5. Resources

Approximately 50 CPU days of computation were used for both submissions and all experiments were performed with the ada UBC cluster (21 nodes, each with 32 Intel(R) Xeon(R) 2.10GHz cores). When considering submission 2, ≈ 250 CPU days were spent exploring the *Zilla design space for high-performance configurations.

Acknowledgments

Special thanks to Alexandre Fréchette who has done the majority of software development on *Zilla, to Lin Xu who was the lead on SATzilla between 2007 and 2012, and to Rachel Han for her help with preparing the submission.

References

L. Kotthoff, B. Hurley, and B. O'Sullivan. The ICON Challenge on Algorithm Selection. *AI Magazine*, 38(2):91–93, 2017.

D. Le Berre, O. Roussel, and L. Simon. The international SAT competitions web page. www.satcompetition.org, 2015. Accessed: 2015-11-16.

M. Lindauer, R. Bergdoll, and F. Hutter. An empirical study of per-instance algorithm scheduling. In *International Conference on Learning and Intelligent Optimization*, pages 253–259. Springer, 2016.

M. Streeter and D. Golovin. An online algorithm for maximizing submodular functions. In *Advances in Neural Information Processing Systems*, pages 1577–1584, 2009.

L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

L. Xu, F. Hutter, J. Shen, H. H. Hoos, and K. Leyton-Brown. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. In *Proceedings of SAT Challenge*, pages 57–58, 2012.