Simple Algorithm Portfolio for SAT

Mladen Nikolić · Filip Marić · Predrag Janičić

Abstract The importance of algorithm portfolio techniques for SAT has long been noted, and a number of very successful systems have been devised, including the most successful one — SATzilla. However, all these systems are quite complex (to understand, reimplement, or modify). In this paper we present an algorithm portfolio for SAT that is extremely simple, but in the same time so efficient that it outperforms SATzilla. For a new SAT instance to be solved, our portfolio finds its k-nearest neighbors from the training set and invokes a solver that performs the best for those instances. The main distinguishing feature of our algorithm portfolio is the locality of the selection procedure — the selection of a SAT solver is based only on few instances similar to the input one. An open source tool that implements our approach is publicly available.

1 Introduction

Solving time for a SAT instance can significantly vary for different solvers. Therefore, for many SAT instances, availability of different solvers may be beneficial. This observation leads to algorithm portfolios which, among several available solvers, select one that is expected to perform best on a given instance. This selection is based on data about the performance of available solvers on a large training set of instances. The problem of algorithm portfolio is not limited only

M. Nikolić
Faculty of Mathematics, University of Blegrade, Belgrade, Serbia
E-mail: nikolic@matf.bg.ac.rs
Tel.: +381648650064

F. Marić
Faculty of Mathematics, University of Blegrade, Belgrade, Serbia
E-mail: filip@matf.bg.ac.rs

P. Janičić
Faculty of Mathematics, University of Blegrade, Belgrade, Serbia
E-mail: janicic@matf.bg.ac.rs

to the SAT problem, but can be considered in general (Huberman, Lukose, & Hogg, 1997; Gomes & Selman, 2001; Horvitz, Ruan, Gomes, Kautz, Selman, & Chickering, 2001).

There are a number of approaches to algorithm portfolios for SAT, the most successful one being SATzilla (Xu, Hutter, Hoos, & Leyton-Brown, 2008) that regularly wins in various categories of SAT Competitions[1]. SATzilla is very successful, but is a rather complex machinery not easy to understand, reimplement or modify. In this paper we present an algorithm portfolio system, based on the k-nearest neighbors method, that is conceptually significantly simpler and more efficient than SATzilla. It derives from our earlier research on solver policy selection (Nikolić, Marić, & Janičić, 2009).

The rest of the paper is organized as follows. In Section 2, some of the existing algorithm portfolios are described. In Section 3, the proposed technique is described and in Section 4 the experimental results are presented. The conclusions are drawn in Section 5.

2 Algorithm Portfolios for SAT

Various approaches to algorithm portfolio for SAT and related problems have been devised (Gomes & Selman, 2001; Horvitz et al., 2001; Lagoudakis & Littman, 2001; Samulowitz & Memisevic, 2007), but the turning point in the field has been marked by the appearance of SATzilla portfolio (Nudelman, Brown, Hoos, Devkar, & Shoham, 2004; Xu et al., 2008). Here we describe several recent relevant approaches for algorithm selection for SAT, most of them using fragments of SATzilla methodology.

SATzilla. SATzilla, the algorithm portfolio that has been dominating recent SAT Competitions, is the most important and the most successful algorithm portfolio for SAT, with admirable performance (Xu et al., 2008; Xu, Hutter, Hoos, & Leyton-Brown, 2009). SATzilla represents instances by using different *features* and then predicts runtime of its constituent solvers based on these features and relying on *empirical hardness models* obtained during the training phase.

SATzilla is a very complex system. On a given input instance, SATzilla first runs two *presolvers* for a short amount of time, in a hope that easy instances will be quickly dispatched. If an instance is not solved by the presolvers, its features are computed. Since the feature computation can take too long, before computing features, the feature computation time is predicted using empirical hardness models. If the estimate is more than 2 minutes, a *backup solver* is run. Otherwise, using computed features, a runtime for each component solver is predicted. The solver predicted to be the best is invoked. If this solver fails (e.g., if it crashes or runs out of memory), the next best solver is invoked.

The training data are obtained by measuring the solving time for all instances from some chosen training set by all solvers from some chosen set of solvers (using some predetermined cutoff time). For each category of instances (used in SAT Competitions) — random, crafted, and industrial, a separate SATzilla system is built. For each system, a hierarchical empirical hardness model for each solver

---

[1] `http://www.satcompetitions.org`

is trained to predict its runtime. This prediction is obtained by combining runtime predictions of separate conditional models for satisfiable and for unsatisfiable instances. To enable this, SATzilla uses an estimator of probability whether the input instance is satisfiable that is trained using sparse multinomial logistic regression. Each conditional model is obtained in the following manner. First, starting from a set of base features, a feature selection step is performed in order to find features that maximally reduce the model training error. Then, the pairwise products of the remaining features are added as new features, and the second round of feature selection is performed. Finally, the ridge regression model for runtime prediction is trained using the selected features. From the set of solvers that have been evaluated on the training data, best solvers are chosen for the component solvers automatically, using a randomized iterative procedure. The presolvers and the backup solver are also selected automatically.

ArgoSmArT. ArgoSmArT is a system developed for instance-based selection of policies for a single SAT solver (Nikolić et al., 2009). As a suitable underlying SAT solver it uses a modular solver ArgoSAT (Marić, 2009). ArgoSmArT uses a training set of SAT instances divided manually in classes of instances of similar origin (e.g., as in the SAT Competition corpus). Each instance is represented using (a subset of) the SATzilla features. For the input instance to be solved, the feature values are computed and the nearest (with respect to some distance measure) neighbor instance from the training set, belonging to some class c is found. Then, the input instance is solved using the solver configuration that is known to perform best on the class c.

ArgoSmArT does not deal with solver tuning and assumes that good configurations for classes are provided in advance. This approach could be used for selection of policies for other solvers, too. Moreover, it can be also used as an algorithm portfolio.

ISAC. ISAC is a solver configurator that also has the potential to be applied to the general problem of algorithm portfolio (Kadioglu, Malitsky, Sellmann, & Tierney, 2010). It divides a training set in families automatically using a clustering technique. It is integrated with GGA (Ansótegui, Sellmann, & Tierney, 2009), a system capable of finding a good solver configuration for each family. The instances are represented using SATzilla features, but scaled in the interval [-1,1]. For an input instance, the features are computed and the nearest center of available clusters is found. If the distance from the center of the cluster is less than some threshold value, the best configuration for that cluster is used for solving. Otherwise, a configuration that performs the best on the whole training set is used.

Latent class models. Another recent approach promotes use of statistical models of solver behavior (*latent class models*) for algorithm selection (Silverthorn & Miikkulainen, 2010). The proposed models try to capture the dependencies between solvers, problems, and run durations. Each instance from the training set is solved many times by each solver and a model is fit to the outcomes observed in the training phase using the iterative expectation-maximization algorithm. During the testing phase, the model is updated based on new outcomes. The procedure for

algorithm selection chooses a solver and runtime duration trying to optimize discounted utility measure on the long run. The authors report that their system is roughly comparable to SATzilla.

Non-model-based portfolios. This, most recent, approach (Malitsky, Sabharwal, Samulowitz, & Sellmann, 2011) also relies on k-nearest neighbors, and was independently developed in parallel with our research. However, the two systems differ in some aspects (which will be shown to be important). This approach uses a standard Euclidean distance to measure the distance of neighbors, while each feature has to be scaled to the interval [0,1] to avoid dependence on order of magnitude of numbers involved. Also, the feature set is somewhat different from the one we use. The approach was evaluated on random instances from SAT 2009 competition and gave better results than SATzilla.

3 Nearest Neighbor-Based Algorithm Portfolio

The existing portfolio systems for SAT build their models (e.g., runtime prediction models, grouping of instances, etc.) in advance, regardless of characteristics of the input instance. We expected that a finer algorithm selection might be achievable if a local, input-specific model is built and used. A simple model of that sort can be obtained by the k-nearest neighbor method (Duda, Hart, & Stork, 2001), from just few instances similar to the instance being solved. In the rest of this section we describe our algorithm portfolio for SAT.

It is assumed that a *training set* of instances is solved by all solvers from the portfolio, and that the solving times (within a given cutoff time) are available. Based on these solving times, for each solver a *penalty* can be calculated for any instance (the greater the solving time, the greater the penalty). Each instance is represented by a vector of *features*.

Our algorithm selection technique is given in Figure 1. Basically, *for a new instance to be solved, its k-nearest neighbors from the training set (with respect to some* distance measure*) are found, and the solver with the minimal penalty for those instances is invoked.* In the case of ties among several solvers, one of them that performs the best on the whole training set can be chosen.[2]

To make the method concrete, the set of features, the penalty and the distance measure have to be defined.

Features. The authors of SATzilla introduced 96 features that are used to characterize SAT instance (Xu et al., 2008, 2009), used subsequently also by other systems (Nikolić et al., 2009; Kadioglu et al., 2010). The main problem with using a full set of these features is the time needed to compute them for large instances.[3] The features we chose, given in Figure 2, can be computed very quickly. They are some of the purely syntactical ones used by SATzilla. Though this subset may not be enough for good runtime prediction that SATzilla is based on, it may serve well for algorithm selection.

---

[2] In practice, it is highly unlikely that there are more than one such solver, but for completeness we allow for such possibility (step 5 of the procedure).

[3] As said, SATzilla even performs a feature computation time prediction and does the computation itself only if the predicted time does not exceed 2 minutes.

S: Set of available solvers
T: Set of feature vectors and solving times for each training instance
k: Number of neighbors to be considered ($k \leq |T|$)
i: Input instance

(Solver selection)
1. $f \leftarrow \mathtt{features}(i)$
2. $T' \leftarrow$ set of k instances from T nearest to f
3. $S' \leftarrow \{s \in S \mid \mathtt{penalty}(s, T') = min_{s' \in S}\mathtt{penalty}(s', T')\}$

(Resolution of ties among solvers from S')
4. $S^* \leftarrow \{s \in S' \mid \mathtt{penalty}(s, T) = min_{s' \in S'}\mathtt{penalty}(s', T)\}$

(Solving)
5. Solve i using any $s \in S^*$

Fig. 1 k-nearest neighbors algorithm portfolio for SAT.

Problem Size Features:

1-3. *Number of clauses c, Number of variables v, Ratio v/c*

Variable-Clause Graph Features:

4-8. *Variable nodes degree statistics*: mean, variation coefficient, min, max, and entropy.

9-13. *Clause nodes degree statistics*: mean, variation coefficient, min, max, and entropy.

Balance Features:

14-16. *Ratio of positive and negative literals in each clause*: mean, variation coefficient, and entropy.

17-21. *Ratio of positive and negative occurrences each variable*: mean, variation coefficient, min, max and entropy.

22-23. *Fraction of binary and ternary clauses.*

Proximity to Horn Formula:

24. *Fraction of Horn clauses*

25-29. *Number of occurrences in a Horn clause for each variable*: mean, variation coefficient, min, max, and entropy.

Fig. 2 SATzilla features used.

Penalty. If a solving time for a solver and for a given instance is less that a given cutoff time, the penalty for the solver on that instance is the solving time. If it is greater then the cutoff time, for the penalty time we take 10 times the cutoff time. This is the *PAR10 score* (Hutter, Hoos, Leyton-Brown, & Stützle, 2009). We define a PAR10 score of a solver on a set of instances to be the sum of its PAR10 scores on individual instances.

Distance measure. We prefer the distance measure that performed well for ArgoS-mArT:

$$d(x, y) = \sum_i \frac{|x_i - y_i|}{\sqrt{x_i y_i} + 1}$$

where x_i and y_i are coordinates of vectors x and y (containing feature values of the instance), respectively. However, any distance measure could be used.

Compared to the approaches described in Section 2, our procedure does not discriminate between satisfiable and unsatisfiable or between random, crafted, and industrial instances. The procedure does not use presolvers, does not predict feature computation time, nor it uses any feature selection or feature generation mechanisms. It is not assumed that the structure of instance families is given in advance, nor it is constructed in any way. Also, the algorithm does not use any advanced statistical techniques, nor does solve the same instances several times. Compared to the approach of Malitsky et al. (Malitsky et al., 2011), we use a smaller feature set, different distance measure and avoid feature scaling.

Note that the special case of 1-nearest neighbor technique, has some advantages compared to the general case. Apart for simpler implementation, it can have a wider range of application. In the case of algorithm configuration selection (that can be seen as a special case of algorithm selection — each configuration of an algorithm can be considered as a different algorithm), it would be expensive or practically impossible to have each instance solved for all algorithm configurations. Therefore, neither SATzilla nor k-nearest neighbor approach for $k > 1$ is applicable in this situation. However, there are special optimization based techniques for finding good solver configurations off-line (Hutter et al., 2009; Ansótegui et al., 2009). Hence, for each instance, one good configuration can be known. This is sufficient for the 1-nearest neighbor approach to be used.

4 Implementation and Experimental Evaluation

Our implementation of the presented algorithm portfolio for SAT, ArgoSmArT k-NN,[4] consists of less than 2000 lines of C++ code. The core part, concerned with the solver selection, has around 100 lines of code, while the rest is a feature computation code, solver initialization and invocation, time measurement, etc. All the auxiliary code (everything except the solver selection mechanism) is derived from the SATzilla source code by removing or simplifying parts of its code.

In the evaluation we compare ArgoSmArT k-NN with SATzilla 2009. We are not aware of a publicly available implementation related to the approach of Malitsky at al., but we compare different decisions in our two approaches within ArgoSmArT k-NN.

Instead of solving instances from a training set, the training data for SATzilla 2009[5], available from the SATzilla web site[6], was used. SATzilla was trained using 5883 instances from SAT Competitions (2002-2005 and 2007) and SAT Races 2006 and 2008 (Xu et al., 2009). The data available on the web site include the solving information for 4701 instances (the solving data for other instances SATzilla is trained on is not available on the web). The available solving times of these instances were used, while the feature values were recomputed (in order to avoid using the SatELite preprocessor that SATzilla 2009 uses as a first step of feature computation). The cutoff time of 1500s was used. When instances that where not solved by any solver within that time limit and when duplicate instances were excluded, there were 4276 remaining instances in the training set. The feature vectors

---

[4] The source code and the binary are available from http://argo.matf.bg.ac.rs/ download section.

[5] SATzilla 2009 is a winner of SAT Competition 2009 in random category.

[6] http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/

of training instances and their solving times for all solvers used are included in the implementation of ArgoSmArT (1.3Mb of data). As a test set, we used all the instances from the SAT Competition 2009.

There are 13 solvers for which the solving data are available, and that are used by SATzilla 2009 as components solvers in 3 versions of SATzilla (random, crafted, and industrial) (Xu et al., 2009). Each version of SATzilla uses a specialized subset of these 13 solvers that is detected to perform the best on each category of instances. No specialized versions of ArgoSmArT are made, but simply all 13 solvers are used as component solvers.

ArgoSmArT k-NN can use any distance measure, but we take into consideration two measures. The first is the one shown in Section 3 that performed best for ArgoSmArT, and for some other problems (Tomovic, Janicic, & Keselj, 2006). The second one is the Euclidean distance (with features being scaled to [0,1]) as used by Malitsky et al. The comparison of these distances for various k is shown in Figure 3. The number of solved instances for each distance and each k is obtained by *leave one out* procedure (the solver to be used for each instance is chosen by excluding the instance from the training set, and applying the solver selection procedure using the rest of the training set). It is obvious that ArgoSmArT distance is uniformly better than the Euclidean one. For both distances, the highest number of solved instances is obtained for $k = 9$. Hence, we use these choices for ArgoSmArT k-NN in further evaluation. Also, we justify our choice of features by measuring the feature computation times. For the full set of 48 features used by Malitsky et al., minimal, average and maximal computation times on the training set are 0.002, 19.2 and 6257 seconds. For our, reduced, set of features the minimal, average and maximal computation times are 0.0017, 0.45, and 51.2 seconds.

An experimental comparison between ArgoSmArT k-NN and any individual version of SATzilla would not be fair since each version is designed specifically for one kind of instances. So, in order to make a fair comparison, on random instances we used SATzilla random, on crafted instances we used SATzilla crafted, and on industrial instances we used SATzilla industrial. This virtual SATzilla system will be just referred to as SATzilla. In our experimental comparison, we included MXC08 (the best single solver on the training set), SATzilla, the ArgoSmArT system based on (Nikolić et al., 2009) with 13 SATzillasolvers instead of ArgoSAT configurations, ArgoSmArT 1-NN, and ArgoSmArT 9-NN. Also, we compare to the virtual best solver — a virtual solver that solves each instance by the fastest available solver for that instance (showing the upper achievable limit). Experiments were performed on a cluster with 32 dual core 2GHz Intel Xeon processors with 2GB RAM per processor. The results are given in Table 1 and they show that ArgoSmArT 1-NN/ArgoSmArT 9-NN outperformed all other solvers/portfolios in all categories.

It is a common practice on SAT Competitions and SAT Races to repeat instances known from previous events. This results in overlapping of training and test set. To be thorough, in Table 2, we provide experimental evaluation on the test set without the instances contained in the training set.

One can observe that on one subset of instances (industrial instances that did not appear on earlier competitions), MXC08 component solver performs better than all the portfolio approaches. This probably means that the test set is somewhat biased with respect to the training set. However, the presented results show that ArgoSmArT k-NN significantly outperformed other entrants on this test set
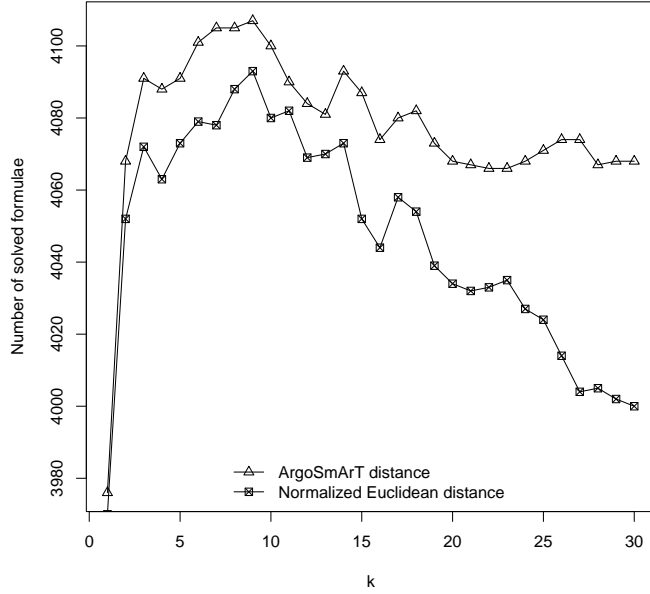
Fig. 3 The number of solved instances from the training set for ArgoSmArT using each of the compared distances for each k from 1 to 30.

	MXC08	SATzilla	ArgoSmArT	ArgoSmArT 1-NN	ArgoSmArT 9-NN	VBS
$Time_{ALL}$	>1500s	635s	874s	390s	353s	115s
N_{ALL}	**355**	**635**	**609**	**685**	**692**	**816**
N_{RND}	84	375	308	367	390	454
N_{CRF}	124	128	154	158	149	188
N_{IND}	147	132	147	160	153	174

Table 1 Experimental results on instances from SAT Competition 2009. For each solver/portfolio the number of solved instances and the median solving time are given for the whole corpus. Also, the number of solved instances is given for each of the categories of instances — random, crafted, and industrial. The total number of instances is 1143.

	MXC08	SATzilla	ArgoSmArT	ArgoSmArT 1-NN	ArgoSmArT 9-NN	VBS
$Time_{ALL}$	>1500s	497s	895s	343s	274s	76s
N_{ALL}	**243**	**513**	**475**	**533**	**553**	**659**
N_{RND}	84	375	308	367	390	454
N_{CRF}	77	68	88	86	83	115
N_{IND}	82	70	79	80	80	90

Table 2 Experimental results on instances from SAT Competition 2009 without the instances known from previous SAT Competitions and SAT Races. For each solver/portfolio the number of solved instances and the median solving time are given for the whole corpus. Also, the number of solved instances is given for each of the categories of instances — random, crafted, and industrial. The total number of instances is 894.

as well. Possible reasons for this involve two characteristics of ArgoSmArT k-NN. First, in contrast to the original ArgoSmArT and SATzilla, ArgoSmArT k-NN does not use predefined groups or precomputed prediction models, built regardless of the input instance. Instead, ArgoSmArT k-NN selects a solver by considering only a local set of instances similar to the input instance. Second, ArgoSmArT and SATzilla make their choices by considering specific groups of instances, but these groups are typically large. On the other hand, ArgoSmArT k-NN considers only a very small number of instances (e.g., $k = 9$) and this eliminates influence of less relevant instances. Indeed, SATzilla improves its predictive performance by building specific versions for smaller sets of related instances (i.e., random, crafted, industrial) (Xu et al., 2008), which also supports the above speculation.

5 Conclusions

We presented a strikingly simple algorithm portfolio for SAT stemming from our work on ArgoSmArT (Nikolić et al., 2009). The presented system, ArgoSmArT k-NN, benefits from the SATzilla system in several ways: it uses a subset of SATzilla features for representation of instances, a selection of SAT solvers, SATzilla solving data for the training corpus, and fragments of SATzilla implementation. However, in its core part — selection of a solver to be used — ArgoSmArT k-NN significantly differs from SATzilla. Instead of predicting runtimes, our system selects a solver based on the knowledge about instances from a local neighborhood of the input instance, using the k-nearest neighbors method. The proposed system is implemented and publicly available as open source. The experimental evaluation shows that the particular decisions made in the design of our system are even better than the decisions made in the similar recent system (Malitsky et al., 2011). Also, it (even the simplest version ArgoSmArT 1-NN) performed substantially better than SATzilla— the most successful, but rather complex, algorithm portfolio for SAT. We believe that the presented approach proves there is a room for improving of algorithm portfolio systems for SAT, not necessarily by overengineering.

References

Ansótegui, C., Sellmann, M., & Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the 15th international conference on Principles and practice of constraint programming*, pp. 142–157. Springer.

Duda, R., Hart, P., & Stork, D. (2001). *Pattern Classification (2nd Edition)*. Wiley-Interscience.

Gomes, C., & Selman, B. (2001). Algorithm portfolios. *Artif. Intell.*, *126*(1-2), 43–62.

Horvitz, E., Ruan, Y., Gomes, C., Kautz, H., Selman, B., & Chickering, D. M. (2001). A bayesian approach to tackling hard computational problems. In *UAI*, pp. 235–244.

Huberman, B., Lukose, R., & Hogg, T. (1997). An economic approach to hard computational problems. *Science*, *27*, 51–53.

Hutter, F., Hoos, H., Leyton-Brown, K., & Stützle (2009). Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). Isac – instance-specific algorithm configuration. In *Proceeding of the 19th European Conference on Artificial Intelligence*, pp. 751–756. IOS Press.

Lagoudakis, M., & Littman, M. (2001). Learning to select branching rules in the dpll procedure for satisfiability. *Electronic Notes in Discrete Mathematics*, *9*, 344–359.

Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2011). Non-model-based algorithm portfolios for SAT. In *Theory and Applications of Satisfiability Testing (SAT)*.

Marić, F. (2009). Formalization and implementation of modern sat solvers. *Journal of Automated Reasoning*, *43*, 81–119.

Nikolić, M., Marić, F., & Janičić, P. (2009). Instance-based selection of policies for sat solvers. In Kullmann, O. (Ed.), *Theory and Applications of Satisfiability Testing - SAT 2009*, Vol. 5584 of *Lecture Notes in Computer Science*, pp. 326–340. Springer.

Nudelman, E., Brown, K. L., Hoos, H. H., Devkar, A., & Shoham, Y. (2004). Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In Wallace, M. (Ed.), *Principles and Practice of Constraint Programming - CP 2004*, Vol. 3258 of *Lecture Notes in Computer Science*, pp. 438–452. Springer.

Samulowitz, H., & Memisevic, R. (2007). Learning to Solve QBF. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pp. 255–260, Vancouver, British Columbia, Canada. AAAI Press.

Silverthorn, B., & Miikkulainen, R. (2010). Latent Class Models for Algorithm Portfolio Methods. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Tomovic, A., Janicic, P., & Keselj, V. (2006). n-Gram-Based Classification and Unsupervised Hierarchical Clustering of Genome Sequences. *Computer Methods and Programs in Biomedicine*, *81*(2), 137–153.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2008). SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research*, *32*, 565–606.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2009). SATzilla2009: an Automatic Algorithm Portfolio for SAT. In *SAT Competition 2009*.