Meta-learning for Recommending Metaheuristics for the MaxSAT Problem

Enrico S Miranda[1], Fabio Fabris[2], Chrystian G M Nascimento[1], Alex A Freitas[2], and Alexandre C M Oliveira[1,2†]

[1]Department of Informatics - DEINF, UFMA Brazil
[2]School of Computing, University of Kent, CT2 7NZ, UK
[†]Corresponding author: alexandre.cesar@ufma.br

Abstract. Solving even moderately-sized Maximum Satisfiability (MaxSAT) problems exactly can be unfeasible due to their NP-Hardness. This leads to the use of metaheuristics that find a solution without exactness guarantees but run in a reasonable time. Yet, choosing the best metaheuristic to solve a MaxSAT problem is hard, justifying the use of meta-learning algorithms for metaheuristic recommendation. These meta-learning algorithms use past experience to choose the best metaheuristic to solve an unseen problem. As far as we know, this is the first time a meta-learning approach is proposed to select *metaheuristics* for solving a MaxSAT problem. Our approach includes the creation of new meta-features derived from graph representations of MaxSAT problems and an interpretation of part of a meta-model. Our approach successfully selected the best metaheuristic to solve the problems 87% of the time, the new meta-features have shown to be as good as the state-of-the-art meta-features, and the meta-model interpretation found interesting problem-specific knowledge.

1 Introduction

The SATisfiability (SAT) problem is the task of deciding if there is a set of assignments to the variables of a boolean expression in conjunctive normal form that meets all clauses. The Maximum SAT (MaxSAT) problem is a generalization of the SAT problem that seeks to find which variable configuration meets most clauses, and what is the maximum number of clauses that can be satisfied.

The MaxSAT problem is NP-Hard [1], meaning that beyond a certain size, the problem becomes intractable. For this reason heuristic algorithms are commonly applied in large problems (http://www.maxsat.udl.cat/16/results-incomplete/). Although heuristic algorithms do not guarantee a solution at a fixed bound from optimum, they return good solutions in a reasonable time budget.

Since no single metaheuristic is the best for every problem [2], it is of great interest to have a 'meta-learning' recommendation system capable of choosing the best metaheuristic to solve a particular MaxSAT problem. To achieve this, characteristics of several MaxSAT problems with known performance in several metaheuristics can be extracted and a data mining algorithm can be used to

learn associations between the characteristics and the best metaheuristics and thus, induce a model that can predict which metaheuristic should be used to solve an unseen problem given its characteristics.

Note that meta-learning does not occur at the problem level (where the metaheuristics operate), rather at a higher level (meta-level), where two design choices must be made: 1) how to represent a meta-instance (how to create the meta-features); and 2) how to select the best metaheuristic method given past experience. In this work we use classification algorithms to solve the second problem and use existing and novel meta-features to solve the first.

The main objective of this paper is to propose a meta-learning approach to predict which metaheuristics should be used to solve a particular MaxSAT problem. This includes the proposal of new meta-features to represent MaxSAT problems and the interpretation of parts of highly accurate meta-models, so that useful knowledge can be extracted from past experience.

This paper is organized as follows: Section 2 presents related work, Section 3 provides the background on the MaxSAT problem and meta-learning. Section 4 presents our proposed meta-learning approach. Section 5 presents the computational results, including an interpretation of some meta-models. Finally, Section 6 highlights the main findings and future research.

2 Related Work

Meta-learning approaches for metaheuristic recommendation have been successfully used on NP-Hard problems other than MaxSAT. Meta-features were extracted from the graph representation of the Traveling Salesman Problems (TSP). In [3], a meta-learning approach was proposed to select appropriate metaheuristics. Authors concluded that their approach was superior to simpler selection strategies. In [4], meta-features based on k-Nearest Neighbors graphs were especially helpful for improving the performance of recommendation systems.

Previous works have applied meta-learning to MaxSAT problems, however, their focus was on choosing the best *exact* method [5, 6], or on finding how long would it take for an algorithm to run [7]. In [8], a collaborative filtering approach for SAT algorithm selection is proposed, in which metaheuristic algorithms are selected considering previous users' choices.

In [9], features of various optimization problems (assignment, traveling salesperson, knapsack, bin-packing, graph coloring, and timetabling) are discussed in a context of problem-independent landscape metrics in order to gain insights about how to share features among problems exhibiting similar structures. Meta-learning has also been applied far beyond classical combinatorial problems, such as berth allocation problem [10].

In [11], traditional data mining datasets were represented as graphs and next, some graph-related descriptors were extracted. These descriptors were used as meta-features for a meta-learning approach for classifier recommendation. The results showed that the new meta-features are competitive in relation to traditional meta-features used in meta-learning for classifier recommendation.

3 Background

3.1 The MaxSAT problem

The MaxSAT optimization problem comprises a set of clauses C and a set of variables X. Each $c_i \in C$ is a clause expressed in the Conjunctive Normal Form (CNF), that is, $C_i \equiv (x_j \vee x_k ... x_l)$, where $x_j, x_k ... x_l \in X$ and each x may or may not be negated. The objective function to be maximised can be expressed as $F(w) = \sum d_i(w_i)$, where $d_i(w_i) = 0$ if c_i is unsatisfied and $d_i(w_i) = 1$ if c_i is satisfied under the variable assignment w_i.

There are several *exact* algorithms for solving MaxSAT problems, most are based on generic techniques for solving integer optimization problems [5]. Recent MaxSAT solvers are based on SAT solvers that employ complex analysis of the problem structure, splitting the original problem into sub-problems, each of them being either solved or aborted by conflict-driven mechanisms [12]. Unfortunately, these algorithms have exponential average runtime complexity on the size of the problem, which makes their application unfeasible in certain moderately-sized problems. To fill this niche, metaheuristic algorithms without exactness guarantees, but that run in reasonable times, are used.

Although metaheuristic algorithms can be used to find good solutions for MaxSAT problems, choosing the best metaheuristic-parameter pair for a particular problem is not trivial. It is common for the same metaheuristic to perform well in a problem and poorly in another. Thus, choosing the right metaheuristic to a problem, given the characteristics of the problem, is hard.

The traditional metaheuristic selection approach is to use expert knowledge to decide, given the characteristics of the problem, which metaheuristics to use. Unfortunately, this approach is error-prone and time-consuming. To avoid this, one can use the Machine Learning approach of "meta-learning for metaheuristic recommendation" to find relations between problem characteristics and their past performance in a systematic way, as discussed in the next section.

3.2 Meta-learning

Meta-learning for metaheuristic recommendation is the task of predicting the performance of a set of metaheuristics (the meta-classes) in a new problem given their past performance and meta-features that describe the problems (meta-instances). In this work, we are predicting the best metaheuristic (the meta-class) to solve a MaxSAT problem (a meta-instance), given characteristics extracted from the problem (the meta-features). As far as we know, this is the first time meta-learning is applied to recommending metaheuristics for MaxSAT problems. We tested several combinations of meta-features and meta-classification algorithms to solve the meta-learning problem, as shown in the next section.

4 Methods

4.1 Definition of the meta-features

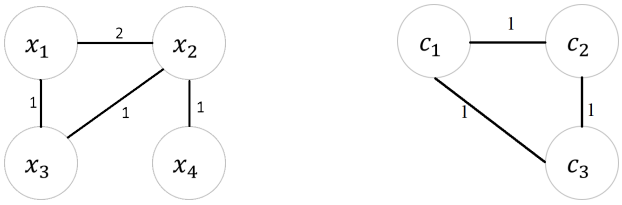
MaxSAT problems can be represented in several ways, e.g.: matrices, binary vectors and graphs. Graph representations are particularly interesting as we can make use of the vast machinery of Graph Theory to extract meta-features. Also, graph representations are normally used to solve the MaxSAT problem as a constraint satisfaction problem [13], which is an indication of their potential. Choosing a *powerful* graph representation is also important: different ways of representing the MaxSAT problems as graphs can generate meta-features with different predictive power. In the next sections, we explain how we transformed the MaxSAT problems into graphs and how the meta-features were extracted from them, creating some novel meta-features. Also, we present traditional (non-graph related) meta-features, already used in the literature.

Graph-based representations: There are three major graph representations for MaxSAT problems [13] (all with undirected edges): 1) Variable Graph (VG), where nodes represent variables and edges link variables that occur in the same clause at least once; 2) Clause Graph (CG) where nodes represent clauses and edges link clauses sharing at least one variable; 3) Variable-Clause Graph (VCG), a bipartite graph where, in the first node subset each node is a variable, and in the second subset each node is a clause. The edges link a variable (a node in the first set) to the clauses where they occur (a node in the second set).

The previously defined problem-to-graph mappings are not one-to-one mappings. That is, different MaxSAT problems are mapped to the same graph. Even worse, problems with very different structures are mapped to isomorphic graph representations. For instance, it does not matter if a pair of variables co-occur in one or all clauses, the VG representation would be the same. Similarly, clauses sharing one or all variables are indistinguishable in the CG representation.

To alleviate this issue, we propose new graph representations encoding more information about the structure of the problem by weighting the edges. The new representations are called WVG and WCG (W for weighted). In WVG (Fig. 1a) the weights mean the number of clauses that the variables share, being a way to signal the strength of the existing relationship between them. Similarly, in WCG (Fig. 1b) the weights mean the number of variables a pair of clauses shares.

Extracted meta-features: Several non-graph-related meta-features have been already proposed for MaxSAT problems. For this reason, the first meta-feature type (MaxSAT-specific meta-features) we evaluated was taken from [13], where a comprehensive list of such meta-features is compiled. The next two meta-features, proposed in this work, are: 1) Unweighted meta-features, extracted from the unweighted graph representations (VG, CG and VCG); and 2) Weighted meta-features, extracted from the weighted graph representations (WVG and



(a) WVG (b) WCG

Fig. 1: Weighted Variable Graphs (WVG) and Weighted Clause Graphs (WCG) representing the expression $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_2)$

WCG). Note that some meta-features are calculated using the following 'summary statistics': mean, standard deviation, variation coefficient, minimum value, maximum value, and Shannon entropy [13].

- **MaxSAT-specific meta-features (20 meta-features)** [13]:
  - Problem-size meta-features (2 meta-features) – number of clauses (c) and number of variables (v);
  - Variable-Clause-balance meta-features (3 meta-features) – ratio clauses/variable (c/v), reciprocal ratio (v/c), linearised ratio ($|4.26 - c/v|$).
  - Ratio of positive-to-negative variables across clauses (6 meta-features) – summary statistics of the rate of positive variables in all clauses.
  - Ratio of positive-to-negative variables overall (6 meta-features) – Summary statistics of the ratio of positive-to-negative occurrence of each variable.
  - Clause complexity (3 meta-features) – Boolean value representing the presence or absence of unary, binary and ternary clauses.
- **Unweighted meta-features (15 meta-features, proposed in this work)**:
  - Simple graph properties (3 meta-features) – number of nodes, number of edges, density.
  - Node-rank meta-features (6 meta-features) – the rank is calculated for each node and summary statistics are obtained for the graph.
  - Weighted clustering coefficient meta-features (6 meta-features)– calculated for each node by dividing the number of edges of all neighbour nodes by $k(k+1)/2$, where k is the number of neighbors of the node [13]. Next, summary statistics are obtained for the graph.
- **Weighted meta-features (6 meta-features, proposed in this work)**:
  - Weights meta-features (6 meta-features) – calculated for each node by summing the weights of its edges. Next, summary statistics are obtained for the graph.

With these 3 meta-feature types we created 6 datasets: a dataset comprising the 'MaxSAT-specific meta-features' (named 'Bespoke'); and 5 datasets containing novel meta-feature types, proposed in this work, namely: 3 datasets comprising 'Unweighted meta-features', one for each one of the VG, CG and

VCG graph representations (named, respectively, VGMF, CGMF and VCGMF – MF for 'Meta-Feature'); and 2 datasets comprising 'Unweighted+Weighted meta-features', one for the VG and another for the CG graph representation (named, respectively, VGWMF and CGWMF – W for 'Weighted').

4.2 Definition of the meta-classes

We defined our meta-classes by running several metaheuristics (varying their parameter settings) in the MaxSAT problems (the meta-instances) and annotating each meta-instance with a meta-class-label representing the best metaheuristic for that problem in terms of number of satisfied clauses.

We have chosen three metaheuristics for this work: Genetic Algorithm (GA)[14], Particle Swarm Optimization (PSO) [15] and Population-Based Incremental Learning (PBIL) [16]. For each one we tried 4 parameter settings, namely: a) two settings with good results obtained after empirical experimentation; b) a setting with a focus on exploitation (smaller populations, more generations); and c) a setting with a focus on exploration (larger populations, fewer generations). These parameter settings are based both on expert knowledge (parameters setting 'a') and on opposite strategies for solving a problem (parameters 'b' and 'c'). Next, we list the metaheuristics-parameter pairs (the meta-class labels) used in this work. For more information about the meaning of parameters see the corresponding references of each metaheuristic.

Genetic Algorithm (GA): For GA, we have considered large and small population sizes, different selection operators with different selective pressures (when possible), and operators of crossover and mutation with different rates, as can be observed in Table 1.

Table 1: GA parameters

Pop.	**Crossover**	**Rate**	**Mutation**	**Rate**	**Update**	**Selection**
50	1 point	60%	Det. Bitflip	5%	1%	2-Tourn.
70	2 point	80%	Std. Bitflip	90%	1%	Stoc. Tourn. 70%
500	1 point	60%	Det. Bitflip	5%	10%	Stoc. Tourn. 60%
70	1 point	80%	Std. Bitflip	75%	1%	5-Tourn.

Particle Swarm Optimization (PSO): For PSO, we used parameter settings that allowed the testing of various population sizes, neighbourhood topologies and sizes, as well as the flight parameters (inertia, attraction parameters L1 and L2). The tested parameters can be seen in Table 2a.

Population Based Incremental Learning (PBIL): For PBIL, we chose parameter settings to evaluate different population sizes, the number of individuals and learning and tolerances. The tested settings can be seen in Table 2b.

Table 2: PSO and PBIL parameter settings.

(a) PSO parameters

Pop	**Neigh.**	**Size**	**Inert.**	**L1**	**L2**
20	Linear	3	1.0	1.7	2.3
50	Linear	5	1.0	1.3	1.9
250	Ring	2	1.0	1.0	1.0
50	Star	-	0.8	2.0	2.5

(b) PBIL parameters

Pop.	n_b	η_b	n_w	η_w	**Threshold**
50	1	0.1	0	0.01	0.01
50	1	0.1	0	0.01	0.05
200	5	0.05	5	0.10	0.3
30	1	0.1	0	0.01	0.01

Metaheuristics results: The set of MaxSAT problems used to create our meta-instances was extracted from the 2014 MaxSAT Evaluation programming challenge (http://www.maxsat.udl.cat/). We collected 555 canonical MaxSAT problem on the *crafted* and *random* categories.

Each one of the 12 metaheuristic-parameter pairs was run on the 555 MaxSAT problems for 15 seconds each on a single thread in an Intel i3 machine with 2358MB of RAM. This test was repeated 20 times in order to increase robustness (due to the stochastic nature of the metaheuristics). Then, the performance of each metaheuristic was compared to others using a test of statistical significance. The metaheuristic with significantly superior results ($p \leq 0.05$) was used to annotate the meta-instance. A total of 11 instances were removed from the dataset due to a tie between algorithms (non statistically significant results). Table 3 shows the 4 metaheuristic-parameter pairs that were the best method in at least one MaxSAT problem (and the number of times they were the best), and therefore only these 4 pairs (out of the 12 tried pairs) were used as meta-classes.

It is possible to notice in Table 3 that two metaheuristic-parameter pairs were far superior to the others. One can interpret the parameter settings of the dominant metaheuristic, 50-pop GA, as inducing a faster convergence, adjusted to perform a smaller number of crossovers and mutations when compared to the second dominant approach, 70-pop GA.

4.3 Meta-learner choice

Now that we have defined the meta-classes and meta-features, we can apply standard classification algorithms as a meta-learner to predict which metaheuristic (meta-class) should be applied to a specific MaxSAT problem (meta-instance). Generally speaking, the classification task is defined as learning a mapping between the instances's features and class labels using a *training set* maximizing some measure of predictive performance. The training set contains instances

Table 3: Number of MaxSAT problems where each metaheuristic and its parameter settings was the best method.

Metaheurstic	**# problems**
GA 50 pop, x-over 1pt(60%), det. flip (5% rate, 1% bits), 2-tourn.	344
GA 70 pop, x-over 1pt(80%), std. bitflip (75% rate, 1% bits), 5-tourn	177
PBIL 30 pop, $n_b = 1$, $\eta_b = 0.1$, $n_w = 0$, $tolerance = 0.01$	19
PBIL 50 pop, $n_b = 1$, $\eta_b = 0.1$, $n_w = 0$, $threshold = 0.01$	4

whose class labels are available to the classifier. Usually, the predictive performance of the classification model is estimated by classifying instances in a *testing set*, a set of instances with known class labels that were not present in the training set. The predictions of the classification model in the testing set are then compared with the actual class labels and some measure of predictive performance is calculated.

In our meta-learning context, the classification task can be defined as building a meta-model using the meta-instances in the training set to predict which metaheuristic should be applied to an unseen MaxSAT problem. Next, we briefly describe the classification algorithms used as meta-learners.

MultiLayer Perceptron (MLP): The MLP classification algorithm is a type of Artificial Neural Network (ANN) that is commonly used in Machine Learning. The MLP algorithm is very powerful: it can induce models representing complex non-linear decision boundaries to classify the instances.

The MLP algorithm mimics natural neural networks by modelling artificial neurons. Each neuron contains several inputs and a single output (that can serve as the input of other neurons). The output of a neuron is defined by multiplying each input value by a different weight, summing up these values and passing them to an 'activation function' that maps its argument to the $[0, 1]$ domain. The weights are learned in the *training phase* of the algorithm.

Support Vector Machine (SVM): The SVM algorithm, like ANNs, learns complex decision boundaries to separate the instances. However, unlike ANNs, SVMs learn 'optimal' decision boundaries, that is, decision boundaries (also called hyperplanes) that maximize the distance between the instances closest to the boundary (the support vectors) and the decision boundary itself.

More precisely, the SVM algorithm maximises the decision margin, the sum of the minimum distances between the hyperplane and the closest support vectors for each class. This margin is maximized by solving a quadratic optimization problem. Note that the boundaries found by SVMs tend to have better generalization properties (in principle) than the boundaries found by ANNs.

***k*-Nearest Neighbours (*k*-NN):** The k-NN classifier is based on the intuitive idea that 'closely related' instances probably share the same class label. As the

name suggests, to classify an unseen instance, the k-NN classifier finds its k (a parameter) nearest training instances and outputs the majority class among the k Nearest Neighbours as the prediction.

Decision Trees (DT): Note that the MLP and SVM algorithms generate 'black-box' models (models that are very difficult to parse). The k-NN 'model' can arguably be interpreted by analyzing the k nearest neighbours of each instance. But this interpretation is instance-specific, lacking generalisation, and so, no interesting insights about the overall problem can be extracted. For this reason, we have also induced an interpretable Decision Tree (DT) meta-model.

The J48 algorithm (WEKA's version of the C4.5 tree-induction algorithm) creates a binary decision tree using the training instances, where each non-leaf node contains a condition that tests the value an instance's feature against a constant. Note that every path from the root to a leaf node (where a prediction is made) in the DT is a rule where the antecedent is a sequence of conditions (non-leaf nodes) that must be satisfied to 'activate' the rule, and the consequent is the classification of the DT. These rules are potentially interpretable and can give the user valuable insights about the underlying classification problem.

5 Meta-learning Computational Experiments

5.1 Predictive accuracy results

In this section, we evaluate the performance of the meta-learner classification algorithms (k-NN, SVM, MLP, DT), using the default parameters in the WEKA data mining tool [17]. The results in Table 4 were obtained by averaging performance estimation of 30 runs of the 10-fold cross-validation procedure (10CV). The 10CV procedure randomly divides the dataset into 10 folds (parts) of equal size and then, each one of these 10 folds is used a 'testing set', whereas the others 9 are used as a 'training set'. The 'percentage of correctly classified instances' is calculated for each testing fold and averaged across the 10 cross-validation iterations, and returned as the performance estimation of the algorithm. To find statistically significance differences among classifiers we used the paired t-test, with $\alpha = 0.05$, (adjusted for multiple comparisons) [18].

Table 4: Percentage of correctly classified meta-instances. Numbers in bold represent the best dataset – i.e., the best type of meta-feature – for each classifier (tied datasets in terms of statistical significance, $p > 0.05$, are also in bold).

Classifier	Bespoke	VGMF	CGMF	VCGMF	WVGMF	WCGMF
DT	**0.8769**	0.8732	0.8677	0.8668	0.8704	0.8650
k-NN	0.8082	0.8041	**0.8209**	**0.8212**	0.7875	**0.8218**
SVM	**0.8720**	0.7632	0.6339	0.8162	0.7891	0.6308
MLP	0.8701	**0.8718**	0.8650	**0.8733**	**0.8702**	0.8692

All classifier-dataset pairs achieve statistically significant superior performance than the naive approach of always predicting the most frequent meta-class ($p = 0.001$), which achieves a predictive performance of 0.6324. In fact, most classifier-dataset pairs are *far* superior to the naive approach with the exception of SVM+CGMF and SVM+WCGMF pairs. We attribute this to the fact that we have not tried to optimize the SVM kernel nor its parameters.

We can also see in Table 4 that there is no clearly superior dataset (meta-feature type) for all classifiers: for DT and SVM, the best dataset is the 'Bespoke' dataset. For k-NN, the best dataset is the WCGMF dataset (tied with CGMF and VCGMF). For the MLP, the best dataset is VCGMF (tied with VCMF and WVGMF). Analysing the impact of the Weighted meta-feature type, we can conclude that adding weights to the VGMF dataset improved the predictive performance significantly when using the SVM classifier ($p = 10^{-14}$) and adding weights to the CGMF representation improved the predictive performance significantly when using the MLP classifier ($p = 0.01$)

Over all classifier-dataset pairs, the best classifier-dataset pair was the DT using the 'Bespoke' dataset. This classifier pair had better predictive performance than all other pairs according to the Hochberg's step-up procedure [18] ($p = 0.0003$). In the next section, we shall interpret this meta-model.

5.2 Interpreting the best Meta-model (a decision tree)

In this section, we focus our interpretation efforts on rules predicting the meta-classes GA_{fast} and GA_{slow}, where GA_{fast} corresponds to the fast-converging, expert-tuned GA (first line of Table 1) and GA_{slow} to the slow-converging, expert-tuned GA (second line of Table 1). A set of interesting rules (paths from the root to a leaf node) were manually extracted from the DT induced using the whole 'Bespoke' dataset (the dataset with best predictive accuracy results) as 'training', to maximize the rules' support. Each rule is presented as a sequence of conditions that the meta-instance must satisfy, followed by the predicted meta-class label if that sequence of conditions is met. Following the meta-class label, in parenthesis, we show the total number of training meta-instances associated with that rule followed by the number of misclassified training meta-instances associated with that rule (if any). We focus on rules with high recall (the number of correctly classified meta-instances covered by the rule divided by the total number of meta-instances with that meta-class label) and high precision (the number of meta-instances correctly classified by that rule divided by the number of meta-instances classified by that rule).

The following powerful rule using the 'Bespoke' dataset only uses the meta-feature 'variables', i.e., the number of variables in the Max-SAT problem.

```
IF (variables > 100) THEN GA_fast (383.0/40.0)
```

This rule has a very good recall (0.997) and precision (0.90) for meta-class label GA_{fast} and encodes the knowledge: 'if the MaxSAT problem has more than 100 variables, metaheuristic GA_{fast} should be used'. This rule is compatible with

the intuitive notion that given that we had a limited time budget (15 seconds per problem), it is better to use faster-converging metaheuristics when the problem is big, so that the population has time to converge to a local minimum.

The next rule is used to classify meta-instances as GA_{slow}:

```
IF (variables ≤ 100) AND (variables > 43) THEN GA_slow (112.0/5.0)
```

This rule has a good recall (0.60) and a very good precision (0.96) for meta-class label GA_{slow}. The conditions of this rule create the interval $43 <$ variables ≤ 100, which seems very good to identify MaxSAT problems with high affinity with metaheuristic GA_{slow}. Again, the rule makes intuitive sense, since problem sizes in this interval are relatively small and slower-converging metaheuristics that explore more the search space should be able to perform better than metaheuristics that converge quickly.

The next rule is another good predictor of meta class GA_{slow} (precision = 0.70, recall = 0.16).

```
IF (variables ≤ 100) AND (variables ≤ 43) AND (ratio > 16.15) THEN GA_slow (40.0/12.0)
```

Note that the condition (variables ≤ 100) of this rule is superfluous, so the rule translates to the phrase: 'if the MaxSAT problem has less than or exactly 43 variables and the clause-to-variable ratio is greater than 16.15, use GA_{slow}'. This suggests that GA_{slow} should be used when the problem has few variables but a relatively complex search space (more than 16.25 clauses per variable).

6 Conclusions and Future Works

The usual approach to metaheuristic parameter optimization is to empirically fine-tune the parameters for a given problem. In this context, meta-learning arises as an alternative to recommend parameter-tuned metaheuristics.

In this work, meta-learning was proposed to select the best metaheuristic-parameter pairs for solving MaxSAT problems. In order to be able to learn at the meta-level, a set of meta-features describing each MaxSAT problem (meta-instance) was used. To create the meta-classes, different population-based metaheuristics and parameters settings were used to solve the meta-instances.

Every used meta-learner was able to outperform the naive approach of always choosing the overall best metaheuristic to solve the problem. This result validates the proposal that meta-learning can successfully choose the best metaheuristics for MaxSAT. We have also observed that there is no dominant MaxSAT problem representation, the best representation is heavily dependent on the meta-learner used. In particular, we have observed that the graph-based representations proposed in this work (i.e., meta-features extracted from graphs) achieved the best performance when using 2 out of 4 meta-learners. We have also interpreted the best meta-model and observed that some rules recommending the use of GA and its parameter settings seem to be sound and in agreement with our intuitions.

As future work, we plan to tune the parameters of the meta-learners using an automated approach, apply our method to more MaxSAT instances and automatically choose the best parameter settings for each metaheuristic before the execution of the meta-learning process.

References

1. Jaumard, B., Simeone, B.: On the complexity of the maximum satisfiability problem for horn formulas. Inf. Process. Lett. **26** (1987) 1–4
2. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. Evolutionary Computation, IEEE Transactions on **1**(1) (1997) 67–82
3. Kanda, J., de Carvalho, A., Hruschka, E., Soares, C., Brazdil, P.: Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features. Neurocomputing **205** (2016) 393–406
4. Pihera, J., Musliu, N.: Application of machine learning to algorithm selection for TSP. In: 2014 IEEE 26th Int. Conf. on Tools with AI. (Nov 2014) 47–54
5. Anstegui, C., Malitsky, Y., Sellmann, M.: MaxSAT by improved instance-specific algorithm configuration. In: 28th AAAI Conf. on Art. Intel. (2014)
6. Matos, P., Planes, J., Letombe, F., Marques-Silva, J.: A MaxSAT algorithm portfolio. Frontiers in Artificial Intelligence and Applications **178** (2008) 911 – 912
7. Zhang, H., Shen, H., Many, F.: Exact algorithms for MaxSAT. Electronic Notes in Theoretical Computer Science **86**(1) (2003) 190 – 203
8. Mısır, M., Sebag, M.: Alors: An algorithm recommender system. Artificial Intelligence **244** (2017) 291–314
9. Smith-Miles, K., Lopes, L.: Measuring instance difficulty for combinatorial optimization problems. Computers & Operations Research **39**(5) (2012) 875 – 889
10. de Len, A.D., Lalla-Ruiz, E., Melin-Batista, B., Moreno-Vega, J.M.: A machine learning-based system for berth scheduling at bulk terminals. Expert Systems with Applications **87** (2017) 170 – 182
11. Morais, G., Prati, R.C.: Complex network measures for data set characterization. In: Brazilian Conf. on Intel. Systems (BRACIS), IEEE (2013) 12–18
12. Chen, J.: Building a hybrid SAT solver via conflict-driven, look-ahead and xor reasoning techniques. In: Theory and Applications of Satisfiability Testing - SAT 2009. (2009) 298–311
13. Nudelman, E., Leyton-Brown, K., Hoos, H.H., Devkar, A., Shoham, Y.: Understanding random SAT: Beyond the clauses-to-variables ratio. In: Principles and Practice of Constraint Programming–CP 2004. Springer (2004) 438–452
14. Goldberg, D.E., Holland, J.H.: Genetic algorithms and machine learning. Machine learning **3**(2) (1988) 95–99
15. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc., IEEE Intern. Conf. on Neural Networks,. Volume 4. (1995) 1942–1948 vol.4
16. Baluja, S.: Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, DTIC Document (1994)
17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD explorations newsletter **11**(1) (2009) 10–18
18. Demsar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. J. Mach. Learn. Res. **7** (2006) 1–30