Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Reactive Dialectic Search Portfolios for MaxSAT*

Carlos Ansótegui, Josep Pon
DIEI
Universitat de Lleida, Spain
{carlos,jponfarreny}@diei.udl.cat

Meinolf Sellmann
IBM Research, USA
meinolf@us.ibm.com

Kevin Tierney
DS&OR Lab
University of Paderborn, Germany
tierney@dsor.de

Abstract

Metaheuristics have been developed to provide general purpose approaches for solving hard combinatorial problems. While these frameworks often serve as the starting point for the development of problem-specific search procedures, they very rarely work efficiently in their default state. We combine the ideas of reactive search, which adjusts key parameters during search, and algorithm configuration, which fine-tunes algorithm parameters for a given set of problem instances, for the automatic compilation of a portfolio of highly reactive dialectic search heuristics for MaxSAT.

Even though the dialectic search metaheuristic knows nothing more about MaxSAT than how to evaluate the cost of a truth assignment, our automatically generated solver defines a new state of the art for random weighted partial MaxSAT instances. Moreover, when combined with an industrial MaxSAT solver, the self-assembled reactive portfolio was able to win four out of nine gold medals at the recent 2016 MaxSAT Evaluation on random, crafted, and industrial partial and weighted-partial MaxSAT instances.

Introduction

The meta-algorithmics community has seen major advances in recent years. On the one hand, techniques for algorithm selection (Gomes and Selman 2001; Leyton-Brown et al. 2003) have led to major advances in our ability to solve great ranges of different types of instances in various domains (Xu et al. 2008; O'Mahony et al. 2008; Kadioglu et al. 2011; Xu et al. 2012; Malitsky et al. 2013; Bischl et al. 2016). On the other hand, algorithm configurators have advanced from limited tuning approaches (Adenso-Diaz and Laguna 2006) to scalable, high-powered general methods (Hutter et al. 2009; Ansotegui, Sellmann, and Tierney 2009; Hutter, Hoos, and Leyton-Brown 2011; Ansotegui et al. 2015; Birattari et al. 2010). Combining portfolios and automatic configuration has led to input-specific tuners (Xu, Hoos, and Leyton-Brown 2010; Kadioglu et al. 2010; Ansótegui et al. 2016) that not only choose superior parameterizations for a target algorithm, but also create new ones based on the input to be processed.

*Research partially supported by the Ministerio de Economía y Competividad research project TASSAT2:TIN2013-48031-C4-4-P.

The methods above all focus on choices made before the actual target algorithm is run. Approaches that aim at modifying algorithm behavior a posteriori during the actual run have also been developed. The reactive tabu search (RTS) algorithm from (Battiti and Tecchiolli 1994) is the prototypical example. RTS modifies the length of the tabu list dynamically during search depending on how the search progresses. Another example is the Stage approach from (Boyan and Moore 2000), a heuristic local search method that analyzes search trajectories to construct predictive evaluation functions that are used during search to escape local minima. New theoretical results (e.g. by (Doerr and Doerr 2015)) prove that dynamic updates during search can guarantee strictly better asymptotic performance. In addition, (Stützle and López-Ibáñez 2016) argues that offline configuration can be used to create online control strategies.

In this paper we combine automatic, input-specific algorithm configuration with reactive search. We do so with the objective of complementing an existing MaxSAT solver with a portfolio of automatically generated search algorithms and thus achieve a more robust, state-of-the-art heuristic MaxSAT solver.

We first give a review of the existing research, followed by an introduction to our new reactive metaheuristic. Finally, we apply it to MaxSAT and compare it empirically with state-of-the-art MaxSAT solvers as well as its non-reactive counterpart, both in combination with the existing solver and in isolation. In the end, we obtain a solver that outperforms the state of the art in various categories of heuristic MaxSAT solving, as assessed independently in the 2016 MaxSAT Evaluation (Argelich et al. 2016).

Background

To position our contribution, we provide background information in three areas. We first introduce the problem domain, MaxSAT. We then review instance-specific algorithm configuration, and finally present the dialectic search metaheuristic that we make reactive. As we have to be brief, for full details we refer the reader to the respective literature.

MaxSAT

The MaxSAT problem is the optimization version of the well-known, quintessentially NP-hard Satisfiability (SAT) problem. Formally:

A *truth variable* is a variable that either takes true or false as values. Such an assignment is called a *truth assignment*. A *literal* is a truth variable (positive literal) or its negation (negative literal). A positive literal is said to evaluate to true iff its variable is set to true. A negative literal is said to evaluate to true iff its variable is set to false.

A *clause* is a disjunction of literals. A clause is said to be *satisfied* under a truth assignment to its variables if at least one of the literals in the clause evaluates to true. Otherwise the clause is said to be *falsified*. A *weighted clause* is a pair (C, w), where C is a clause and w is a natural number or infinity, indicating the cost for falsifying clause C.

A *Weighted Partial MaxSAT formula* (WPMS) is a set of weighted clauses $\varphi = \{(C_1, w_1), ..., (C_m, w_m), (C_{m+1}, \infty), ..., (C_{m+m'}, \infty)\}$ where the first m clauses are *soft* and the last m' clauses are *hard*. The objective of the *MaxSAT problem* is to find a truth assignment that satisfies all hard clauses while minimizing the total cost of all soft clauses that are falsified.

A *Partial MaxSAT formula* (PMS) is a WPMS formula where the weights of soft clauses are all equal. A *(plain) MaxSAT formula* is a partial MaxSAT formula where all clauses are soft.

MaxSAT solvers have been used to effectively tackle problems in several domains, such as scheduling (Vasquez and Hao 2001), FPGA routing (Xu, Rutenbar, and Sakallah 2003), and circuit design and debugging (Safarpour et al. 2007), among many others. Driven by the annual *MaxSAT Evaluations* (Argelich et al. 2016), MaxSAT solvers have seen significant performance improvements in recent years.

Input-specific Algorithm Configuration

Automatic algorithm configurators accept a parameterized *target algorithm* and a set of *training inputs* and search for a set of parameters providing superior performance with the target algorithm on the input. A typical application of algorithm configurators is the search for good *default parameters* for a given algorithm. The idea of input-specific algorithm configuration is to choose a superior parameterization for a target algorithm after inspecting the input that needs to be processed *at runtime* (Xu, Hoos, and Leyton-Brown 2010; Kadioglu et al. 2010; Ansótegui et al. 2016).

Although tempting, experience has shown that searching for a new parameterization for a given input at runtime is extremely difficult. Consequently, state-of-the-art approaches generate a set of high-performance parameterizations offline during training by employing an (input-oblivious) algorithm configurator such as SMAC (Hutter, Hoos, and Leyton-Brown 2011) or GGA++ (Ansotegui et al. 2015). The input-specific configurator ISAC++ (Ansótegui et al. 2016), e.g., first clusters the training input and then uses GGA++ to find a good parameterization for each cluster.

GGA++ (Ansotegui et al. 2015) is a genetic algorithm that uses a *surrogate model* to create offspring for two given parents: Instead of using random recombination, a predictive model is trained to estimate where superior parameterizations may be found. For two parents, the recombination of the parents' genes is then chosen so as to maximize performance as predicted by the surrogate model. The use of surrogates to guide the search for good parameterizations was first introduced in (Hutter, Hoos, and Leyton-Brown 2011).

Once parameterizations for different clusters of input data are determined, the last step in input-specific algorithm configuration is to combine the parameterizations that were generated in an algorithm portfolio, typically by using the SATzilla-2012 (Xu et al. 2012) method or, as in ISAC++, by employing cost-sensitive hierarchical clustering (CSHC) (Malitsky et al. 2013).

ISAC++ was successfully applied to complete MaxSAT solvers, i.e. those solvers that also prove the optimality of the solutions found (Ansótegui et al. 2016). In 2016, e.g., ISAC-based portfolios defined the state-of-the-art in 7 out of 9 complete categories (whereby portfolios were only allowed to participate outside of the competition). At the same time, however, ISAC-based portfolios could only score one second and one third place in 2 out of 9 incomplete categories, where solvers provide best-effort truth assignments without proofs of optimality. This can either mean that progress in incomplete MaxSAT solving is much faster than for complete solvers (as the ISAC portfolio entries are based on solvers from the year prior to the current competition). Or it could mean that instance-specific tuning of local search solvers is inherently too limited as an approach, and that we need to open search heuristic parameters much more than we currently do to allow automatic configuration to have the desired impact.

Dialectic Search

The last ingredient we need for our work is the metaheuristic that we aim to improve. Dialectic search was introduced in (Kadioglu and Sellmann 2009), and its parameterized form is shown in Algorithm 1. The algorithm accepts the following parameters:

- f: The objective function to be optimized.
- g: The size of the greedy candidate set as percentage of all variables in the problem.
- a_l, a_u: A lower and upper bound on the percentage of variables to be changed to construct an antithesis. The exact size of the change is then chosen uniformly at random in the interval given whenever a new antithesis is generated.
- p_a: The probability of greedily improving the antithesis.
- p_r: The probability of restarting the search.
- r_l, r_u: A lower and upper bound on the percentage of variables to be changed to construct a new starting point. The exact size of the change is then chosen uniformly at random in the interval for each restart.

The algorithm starts with an initial assignment to all variables, and aims to improve this assignment in a greedy search, making the best individual variable change in each step until any such change would lead to a worsening of the objective function. Depending on the problem addressed, the greedy search itself may take significant time. In dialectic search, we therefore randomly limit the search for a best variable to only alter a set of candidate variables that are randomly selected in each greedy step. If none of these can improve the objective, the greedy algorithm halts, otherwise we make the best change of a candidate variable.

Algorithm 1 Parameterized Dialectic Search

```
1: function DIALECTIC-SEARCH (f, g, a_l, a_u, p_a, p_r, r_l, r_u)
2:     INIT(thes)
3:     best ← thes ← GREEDY(thes, g)
4:     while not timeout do
5:         while true do
6:             anti ← MODIFY(thes, a_l, a_u)
7:             if ANTIGREEDY(p_a) then
8:                 anti ← GREEDY(anti, g)
9:             syn ← GREEDY(MERGE(thes, anti), g)
10:            if f(syn) < f(best) then best ← syn
11:            if f(syn) < f(thes) then
12:                thes ← syn
13:            else if RESTART(p_r) then break
14:        thes ← GREEDY(MODIFY(thes, r_l, r_u), g)
15:    return best
```

The resulting assignment is called the "thesis." Dialectic search now alters parts of the thesis, thus generating a so-called "antithesis" (function *Modify*). The antithesis may be improved by a greedy search itself. Thus equipped with a thesis and an antithesis, dialectic search then aims to generate a "synthesis" by searching the space between thesis and antithesis (function *Merge*).

The synthesis is constructed by searching the area between the thesis and antithesis, for example by a nested local search (Lourenço, Martin, and Stützle 2003) or, our choice for this work, by path relinking (Glover, Laguna, and Marti 2000). Starting from the thesis, we greedily select the best (in the sense that it most favorably affects the objective) variable among those where the current assignment still differs from the antithesis. We then set this variable to the value it takes in the antithesis. We repeat this until we arrive at the antithesis. The synthesis is then the best assignment we encountered. Then, we greedily improve the synthesis. If the resulting synthesis improves the current thesis, it becomes the new thesis. If the synthesis is worse than the thesis, we restart with a given probability at a new starting point that obtained by applying some random modification to the current thesis. If we do not restart, we choose a new antithesis. One such iteration is called a "move." Each change in a variable, either within a greedy search or when traversing from thesis to antithesis, is called a "step."

There are a number of decisions that dialectic search must make: What is the best size for the candidate set in the greedy search (parameter g)? How many variables should be changed to generate the antithesis (parameters a_l, a_u)? With what probability should we greedily improve the antithesis (parameter p_a)? With what probability should we restart the search (parameter p_r)? When a restart is triggered, how many variables should be changed to new random values to generate a new starting point (parameters r_l, r_u)?

Note that the parameters could be set to realize a wide variety of search algorithms, from an iterated local search with an outer random walk over nested greedy searches ($p_r = 1$, $a_l = a_u = 0$, $g = 1$) to iterated path relinking between the currently best known solution and randomly generated local optima ($p_r = 0$, $g = 1$, $a_l = a_u = 0.5$, $p_a = 1$).

We could simply employ meta-algorithmics technology to automatically and, possibly input-specifically, tune our metaheuristic solver for any application. As such, our starting point is similar to the SATenstein solver that was developed for SAT (KhudaBukhsh et al. 2009). SATenstein is a highly parameterized local search SAT solver that can be instantiated to behave like virtually any successful existing local search approach. SATenstein was configured in an input-specific fashion. That is, the SATenstein solver is really a portfolio of different parameterizations of a very flexible local search framework specifically designed for SAT. In a similar fashion, multi-objective evolutionary algorithms are configured by (Bezerra, López-Ibáñez, and Stützle 2016).

In the following, we go one step further by making all dialectic search parameters *reactive*. That is, we do not simply build a portfolio of different dialectic searches for MaxSAT (and thus build some sort of MaxSATenstein). Instead, we allow the metaheuristic to *change its characteristics dynamically during search* based on the search progression.

Reactive Dialectic Search Portfolios

To set the parameters that guide the search, we will track the progress of the time-limited search as it is unfolding. In particular, we propose to track the following eleven values. We emphasize that other properties could be tracked in addition to these ones:

1. Time elapsed as percentage of total time before timeout.
2. Number of restarts conducted as a percentage of total restarts expected to be completed within the time limit.
3. Number of moves as a percentage of the total moves expected to be completed within the time limit.
4. Number of steps as a percentage of the total steps expected to be completed within the time limit.
5. Total number of improving syntheses found over the total number of dialectic moves expected to be completed within the time limit.
6. Number of moves in the current restart over the total number of dialectic moves expected to be completed within the time limit.
7. Number of moves since the current best known solution was found over the total number of dialectic moves expected to be completed within the time limit.
8. Number of moves since the last thesis update in the current restart over the total number of dialectic moves expected to be completed within the time limit.
9. Number of steps in the current restart over the total number of steps expected to be completed within the time limit.
10. Number of steps since the current best known solution was found over the total number of steps expected to be completed within the time limit.
11. Number of steps since the last thesis update in the current restart over the total number of steps expected to be completed within the time limit.

The objective is now to find a way to make the search use these values to set the seven parameters that guide the search dynamically. Note that all parameters are values between 0

and 1, either because they represent probabilities or percentages of the total number of variables in the given problem instance. Naming the values above $v_1, \ldots, v_{11}$, we set

$$p_k = \frac{1}{1 + e^{(w_0^k + \sum_i v_i w_i^k)}}$$

for each dialectic search parameter p_k, $k = 1 \ldots 7$.

We have thus transformed the configurable dialectic search with seven static parameters into a *hyper-configurable reactive dialectic search (HRDS)* with 84 (7 times 12) meta-parameters.

An adequate interpretation of this approach is that the dependence of the dialectic search parameters from the statistics of the unfolding search is determined by a *logistic regression* (in the original sense, not its common application to classification). The obvious challenge now is to *learn* the meta-parameters w_i^k in a way that will lead to good search performance.

Lacking any other supervision than the total search performance, we employ ISAC++ (Ansótegui et al. 2016) for this task. We first cluster training inputs, then run GGA++ (Ansotegui et al. 2015) on each cluster, and finally build a portfolio of the parameterizations found using cost-sensitive hierarchical clustering (CSHC) (Malitsky et al. 2013).

It is noteworthy that, until now, nothing in our approach has been MaxSAT specific. That is, in principle we can employ the reactive dialectic search approach outlined above on any combinatorial search problem. Additionally, the idea can be adapted to any reactive search metaheuristic. To use it for time-limited local search for MaxSAT, however, we need to make some decisions.

Evaluating Parameterization Performance: We first need to set a metric to ascertain when a parameterization is better than another. For each MaxSAT training instance, we record the best known-solution from prior experience. When HRDS finds a truth assignment with that quality, we will consider the instance "solved" for training purposes and stop the dialectic search run.

In the beginning of our tuning, all parameterizations may be so bad that, within our training time limit, none can find a solution with that best known cost. Therefore, all parameterizations time out and all we can compare is how well they did relative to the best known solution within the time limit. Later, when some parameterizations get some instances to solve to the best known solution quality before the time limit, we can count the number of instances the parameterization is able to "solve" in this way, and the one with the highest count wins. Finally, towards the end of the tuning, we will hopefully have high quality parameterizations that can "solve" all instances. In this case, we can consider the average time it took to solve the instances to determine the winning parameterization.

Algorithm 2 shows the comparison function that guides how we determine top-performing parameterizations within GGA++ accordingly. Rather than giving each parameterization a numeric score (such as a penalized runtime, e.g., as done in many prior applications of algorithm tuners such as (Hutter et al. 2015)) at the end of each tournament, we have defined a comparison algorithm that allows us to sort the parameterizations and to determine the winners.

Algorithm 2 Parameterization Comparison Function

```
1: function SELECTWINNER (p1, p2)
2:     if num-solved-bk(p1) ≠ num-solved-bk(p2) then
3:         return argmax_{p∈{p1,p2}}(num-solved-bk(p))
4:     if num-finished(p1) ≠ num-finished(p2) then
5:         return argmax_{p∈{p1,p2}}(num-finished(p))
6:     G1 ← sort-by-gap(results(p1))
7:     G2 ← sort-by-gap(results(p2))
8:     for i = 1 ... |G1| do
9:         if G1[i] < G2[i] then return p1
10:        else if G1[i] > G2[i] then return p2
11:    return argmin_{p∈{p1,p2}}(avg-cpu-time(p))
```

The first criterion is which parameterization solves more instances to the best known quality within the time limit. If these are the same (for example because neither p_1 nor p_2 can "solve" any instances), then the second criterion is to compare the number of runs that finished correctly, i.e., where there were no problems with memory etc. We next compare which parameterization is closer to getting one more instance solved to best-known quality by considering the quality gap to best-known solutions. Finally, if all these criteria do not lead to a winner (for example because both p_1 and p_2 "solve" all instances) we return the parameterization that needs lower average runtime, with ties broken randomly.

We devised this method so as to give the GGA++ tuner a responsive objective function that guides the tuning search effectively no matter how well the current pool of parameterizations currently performs. However, this leaves us with a problem for the surrogate model used to genetically engineer the offspring within GGA++. Namely, the surrogate model needs to predict in what regions of the parameter space we may expect superior parameterizations to be found. We solved this problem by using relative ranks rather than absolute performance when training the surrogate.

Evaluating Truth Assignments: HRDS solves the MaxSAT problem using an (incremental) evaluation of a truth assignment. We simply maintain make-profits and break-costs (the weighted variants of make-counts and break-counts (Gomes et al. 2008)) for each variable to quickly compute the effect on the objective when flipping a variable's truth assignment.

Characterizing MaxSAT Instances: For instance-specific configuration we need features to characterize the

Category	Solver	MS		PMS		WPMS	
		Time	#	Time	#	Time	#
Crafted	HRDS	**0.80**	79	18.73	**48**	41.68	**24**
	CDS	6.44	79	24.79	45	17.40	21
Random	HRDS	9.67	**82**	73.44	**37**	**2.29**	99
	CDS	3.73	76	25.41	24	2.64	99

Table 1: Average time to best bound and number of best upper bounds found for HRDS and CDS.

Solver	Time	#
WPM3	17.73	5
CDS	6.44	79
HRDS	0.80	79
CDS/WPM3	6.56	79
HRDS/WPM3	0.87	79
CCLS	**5.69**	**81**
CnC-LS	2.49	80
CCEHC	3.42	80

(a) MS Crafted (81 instances)

Solver	Time	#
WPM3	17.30	106
CDS	24.79	45
HRDS	18.73	48
CDS/WPM3	18.69	114
HRDS/WPM3	**18.52**	**116**
WPM3-2015-in	15.93	107
Optiriss6	37.85	99
Dist	6.97	81

(b) PMS Crafted (136 instances)

Solver	Time	#
WPM3	26.61	34
CDS	17.40	21
HRDS	41.68	24
CDS/WPM3	19.22	44
HRDS/WPM3	**23.20**	**46**
CCEHC	18.54	39
Ramp	12.18	29
SC2016	4.13	27

(c) WPMS Crafted (65 instances)

Solver	Time	#
WPM3	0.00	0
CDS	3.73	76
HRDS	9.67	82
CDS/WPM3	3.74	76
HRDS/WPM3	11.87	83
CnC-LS	**2.05**	**89**
borealis	2.28	89
SC2016	2.37	89

(d) MS Random (89 instances)

Solver	Time	#
WPM3	34.75	8
CDS	25.41	24
HRDS	73.44	37
CDS/WPM3	12.09	25
HRDS/WPM3	53.49	34
Dist-r	**2.07**	**42**
SC2016	2.55	42
CCLS	3.00	42

(e) PMS Random (42 instances)

Solver	Time	#
WPM3	91.72	1
CDS	2.64	99
HRDS	**2.29**	**99**
CDS/WPM3	2.67	99
HRDS/WPM3	2.56	99
SC2016	2.53	99
Ramp	4.22	99
CCLS	4.46	99

(f) WPMS Random (99 instances)

Table 2: Average time and number of best upper bounds found on MS, PMS and WPMS crafted (top) and random (bottom) for WPM3, the various solvers developed in this paper, and the top 3 solvers of each category from the MaxSAT Evaluation 2016.

inputs. We use the features proposed in (Ansótegui et al. 2016) for this purpose.

Numerical Results

Having developed our approach in the previous section, we now evaluate it empirically.

Benchmark Set: Recall that we eventually intend to complement a solver that was already designed for industrial instances, WPM3 (Ansótegui, Didier, and Gabàs 2015). As this solver already achieves state-of-the-art performance on industrial MaxSAT instances, we focus the training of our hyper-configurable dialectic search approach on randomly generated instances (Random category) as well as instances that are derived as encodings of other problems (Crafted category). Our base set of MaxSAT instances are all instances in the Random and Crafted categories in the MaxSAT Evaluation 2016 (MSE16) (Argelich et al. 2016).

The three variants of the MaxSAT problem divide the base set further: (plain) MaxSAT (MS), Partial MaxSAT (PMS), and Weighted Partial MaxSAT (WPMS). In each category, instances are grouped into families: 3 families for MS crafted, 2 for MS random, 11 for PMS crafted, 4 for PMS random, 11 for WPMS crafted and 3 for WPMS random. We cleanly split each group randomly 80 to 20, whereby the 80% are assigned to our training set while the remaining 20% are set aside for testing.

Infrastructure: We run all our experiments on a cluster featured with Intel Xeon CPU E5-26020 @ 2.6GHz processors, a memory limit of 3.5 GB, and each machine runs an instance of Rocks Cluster 6.5 (Linux 2.6.32), which is the exact same environment used in the MSE16.

ISAC++ Setup: We perform algorithm configuration exclusively on the instances marked for training for each group of instances within the MSE16 dataset that have more than 15 training instances left after the 80/20 split. We use a distributed version of GGA++ with 8 machines with 8 cores each, a population size of 100 individuals and 100 generations, using a 30 second target algorithm timeout. The time limit for the test instances is as in the MSE16, 300 seconds.

Competitors: We compare the following algorithms. HRDS is the new hyper-configurable reactive dialectic search. CDS is an ISAC++ generated portfolio of the statically parameterized dialectic search with seven parameters. HRDS/WPM3 is a portfolio built from HRDS parameterizations plus ISAC++-tuned parameterizations of WPM3. CDS/WPM3 is a portfolio built from CDS parameterizations plus ISAC++-tuned parameterizations of WPM3.

Reactive vs. Non-Reactive Dialectic Search: Our first inquiry is to find out whether making dialectic search hyper-configurable is at all beneficial. The hyper-configuration space includes all static parameterizations, which are obtained by setting all 11 weights corresponding to search statistics to zero and setting the constant weight for each of the seven parameters to the right value. This means that the best parameterization for HRDS will always be at least as good as that of CDS. However, there is no guarantee that the ISAC++ tuner is able to find that parameterization, nor that the best parameterization thus found for the training set also generalizes well to the test instances.

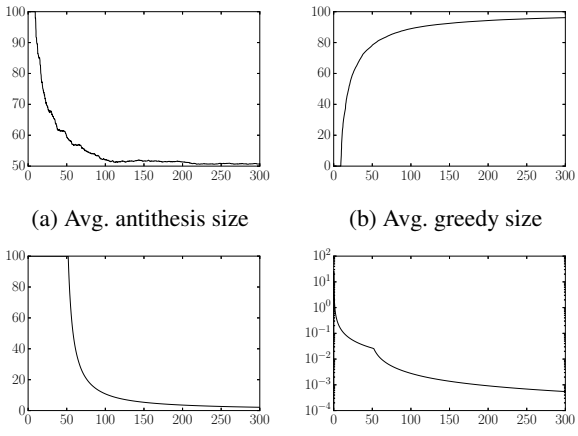
Table 1 shows that our doubts are unfounded. In all categories we tested, HRDS outperforms CDS. The difference in performance is particularly noticeable in two categories. First, on random PMS instances where the reactive dialectic search finds 37 best upper bounds compared to only 24 for the non-reactive counterpart of the otherwise identical approach. Second, on random MS instances HRDS solves 82 instances compared to 76 for CDS. This shows that the algorithm configurator can tune the hyper-configurable heuristic

effectively.

Figure 1 sheds light on the inner workings of HRDS. On top, we track the effective running averages of the antithesis size on the left and the greedy candidate set size on the right for one parameterization, run on a PMS random instance with a time limit of 300 sec. Not shown here is that this parameterization holds the restart probability at zero and anti-greedy probability steady at 1. In this parameterization, the dialectic search thus never restarts, and it always conducts an antithesis greedy improvement, while during the search decreasing the level of difference between thesis and antithesis and simultaneously increasing the greedy candidate set size.

Contrast this with another parameterization, shown at the bottom. This one holds the restart probability and greedy candidate size firmly at 100% (not shown here). In the beginning, the antithesis size is 100%, which means there is a full greedy search started on the exact opposite pole from the current thesis and a path-relink conducted between the thesis and the result of that greedy run. At the same time, the restart size is somewhere between 20 and 50%, leading to an almost entirely new random starting point with only a slight bias towards keeping parts of the current thesis intact in the antithesis. During the course of the optimization, the restart and antithesis sizes are then reduced further and further, making the search stay closer and closer to the current thesis. This behavior, moving from vast exploration in the beginning to more and more conservative moves in the end, is familiar from the simulated annealing metaheuristic. Yet this HRDS parameterization was not invented by a human, but found as an effective method for solving some MaxSAT instances by an algorithm configurator.

Random Weighted-Partial MaxSAT: The next question of interest is whether a solver that consists of reactive dialectic search parameterizations, a solver that is based on a local search metaheuristic that lacks any domain knowledge and only has access to an incremental evaluator of the target objective function, a solver that was programmed in only two working days and was then left to be tuned automatically, whether such a solver could outperform solvers that have been devised by teams of humans that have often developed and evolved their programs for years.



(a) Avg. antithesis size (b) Avg. greedy size

(c) Avg. antithesis size (d) Avg. restart size

Figure 1: Statistics recorded during search for two different HRDS parameterizations.

		1st		2nd		3rd	
Random	MaxSAT	borealis		SC2016		Swcca-ms	
		2.29	454	2.30	454	2.40	454
	Partial	Dist-r		CCEHC		SC2016	
		2.08	209	3.86	209	2.46	208
	W. Partial	**HRDS/WPM3-s**		SC2016		**HRDS/WPM3**	
		26.04	501	2.60	500	9.12	500
Crafted	MaxSAT	CCLS		CCEHC		SC2016	
		3.56	402	3.49	399	2.62	398
	Partial	**HRDS/WPM3**		WPM3-2015-in		**HRDS/WPM3-s**	
		20.26	575	13.33	539	17.99	522
	W. Partial	**HRDS/WPM3**		CCEHC		**HRDS/WPM3-s**	
		28.15	204	19.53	192	29.53	183
Industrial	MaxSAT	CnC-LS		**HRDS/WPM3**		WPM3-2015-in	
		49.00	47	40.84	40	22.57	38
	Partial	**HRDS/WPM3**		WPM3-2015-in		Optiriss6-in	
		26.70	513	22.29	505	37.57	433
	W. Partial	WPM3-2015-in		**HRDS/WPM3**		**HRDS/WPM3-s**	
		17.80	405	22.98	402	18.37	339

Table 3: MaxSAT evaluation 2016 winners, with the average solution time (sec.) and number of best upper bounds found.

Table 2-(f) gives the answer: on random WPMS instances, HRDS outperforms all competitors from the MSE16. While multiple solvers manage to solve 99 test instances within the time limit, it is remarkable to see that the automatically generated solver can even outperform SC2016, which itself works significantly better than all other competitors.

Augmenting an Industrial Solver: We finally arrive at our original objective, namely to make WPM3, a solver designed for industrial MaxSAT instances, a serious contender in other categories as well. Table 2 shows how WPM3 by itself compares with HRDS/WPM3 and CDS/WPM3 on all six random and crafted categories from the MSE16.

We observe that for some categories WPM3 is better than HRDS or CDS, while for others it is the other way around. However, joining WPM3 with HRDS or CDS parameterizations creates an algorithm portfolio that often exceeds the performance of the best choice for each category, and otherwise trails the best performance only slightly. In summary, augmenting a human-developed industrial MaxSAT solver with a machine-trained solver results in a much more robust and across-the-board applicable MaxSAT solver.

2016 MaxSAT Evaluation: Using the methodology above, we entered the MSE16 with an earlier version of HRDS/WPM3 (DSAT-WPM3 at MSE16) with less HDRS parameterizations. In Table 3 we present the official results. Our two submissions (with one version, *-s, using a static schedule in the portfolio) won gold medals in 4 out of 9 categories: WPMS crafted and random, and PMS crafted and industrial. Overall, the two entrants won 10 medals out of 18 possible.

Conclusion

We introduced hyper-configurable dialectic search portfolios and applied them to MaxSAT. We demonstrated that reactive search methods can be tuned effectively and outperform static instance-specific configuration in practice. By itself, the new method was able to define a new state-of-the-art for the random WPMS category, despite its general ignorance regarding the problem it is solving. Used to automatically complement an existing industrial MaxSAT solver, it defined a new approach that works robustly for random, crafted, and industrial instances. The approach was independently evaluated and compared with state-of-the-art MaxSAT solvers at the 2016 MaxSAT Evaluation where it won 4 out of 9 possible gold medals.

Acknowledgements

We thank the Paderborn Center for Parallel Computation (PC^2) and University of Lleida for the use of their respective high-throughput clusters.

References

Adenso-Diaz, B., and Laguna, M. 2006. Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research* 54(1):99–114.

Ansotegui, C.; Malitsky, Y.; Samulowitz, H.; Sellmann, M.; and Tierney, K. 2015. Model-based genetic algorithms for algorithm configuration. In *IJCAI*, 733–739.

Ansótegui, C.; Gabàs, J.; Malitsky, Y.; and Sellmann, M. 2016. MaxSAT by improved instance-specific algorithm configuration. *Artificial Intelligence* 235:26 – 39.

Ansótegui, C.; Didier, F.; and Gabàs, J. 2015. Exploiting the structure of unsatisfiable cores in maxsat. In *IJCAI*, 283–289.

Ansotegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In *CP*, 142–157.

Argelich, J.; Li, C.; Manyà, F.; and Planes, J. 2016. MaxSAT Evaluations. www.maxsat.udl.cat.

Battiti, R., and Tecchiolli, G. 1994. The reactive tabu search. *ORSA Journal on Computing* 6(2):126–140.

Bezerra, L. C. T.; López-Ibáñez, M.; and Stützle, T. 2016. Automatic component-wise design of multiobjective evolutionary algorithms. *IEEE Trans. Evolutionary Computation* 20(3):403–417.

Birattari, M.; Yuan, Z.; Balaprakash, P.; and Stützle, T. 2010. F-race and iterated F-race: An overview. In *Empirical Methods for the Analysis of Optimization Algorithms*, 311–336.

Bischl, B.; Kerschke, P.; Kotthoff, L.; Lindauer, M.; Malitsky, Y.; Fréchette, A.; Hoos, H.; Hutter, F.; Leyton-Brown, K.; Tierney, K.; and Vanschoren, J. 2016. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence* 237:41–58.

Boyan, J., and Moore, A. 2000. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research* 1(Nov):77–112.

Doerr, B., and Doerr, C. 2015. Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In *GECCO*, 1335–1342.

Glover, F.; Laguna, M.; and Marti, R. 2000. Fundamentals of scatter search and path relinking. *Control and Cybernetics* 39:653–684.

Gomes, C., and Selman, B. 2001. Algorithm portfolios. *Artificial Intelligence* 126:43–62.

Gomes, C.; Kautz, H.; Sabharwal, A.; and Selman, B. 2008. *Satisability Solvers*. Elsevier B.V. 89–134.

Hutter, F.; Hoos, H.; Leyton-Brown, K.; and Stuetzle, T. 2009. ParamILS: An automatic algorithm configuration framework. *JAIR* 36:267–306.

Hutter, F.; Lindauer, M.; Balint, A.; Bayless, S.; Hoos, H.; and Leyton-Brown, K. 2015. The configurable SAT solver challenge (CSSC). *arXiv preprint arXiv:1505.01221*.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization (LION 5)*, 507–523.

Kadioglu, S., and Sellmann, M. 2009. Dialectic search. In *CP*, 486–500.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC–Instance-Specific Algorithm Configuration. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *ECAI*, volume 215 of *FAIA*, 751–756.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm selection and scheduling. *CP* 454–469.

KhudaBukhsh, A.; Xu, L.; Hoos, H.; and Leyton-Brown, K. 2009. SATenstein: Automatically building local search sat solvers from components. *IJCAI* 517–524.

Leyton-Brown, K.; Nudelman, E.; Andrew, G.; McFadden, J.; and Shoham, Y. 2003. A portfolio approach to algorithm selection. *IJCAI* 1542–1543.

Lourenço, H.; Martin, O.; and Stützle, T. 2003. Iterated local search. In *Handbook of metaheuristics*. Springer. 320–353.

Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2013. Algorithm portfolios based on cost-sensitive hierarchical clustering. *IJCAI* 608–614.

O'Mahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. *Irish Conference on Artificial Intelligence and Cognitive Science*.

Safarpour, S.; Mangassarian, H.; Veneris, A.; Liffiton, M.; and Sakallah, K. 2007. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design*, 13–19. IEEE.

Stützle, T., and López-Ibáñez, M. 2016. Automatic (offline) configuration of algorithms. In *GECCO Companion*, 795–818.

Vasquez, M., and Hao, J. 2001. A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Computational Optimization and Applications* 20(2):137–157.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for sat. *JAIR* 32(1):565–606.

Xu, L.; Hutter, F.; Shen, J.; Hoos, H.; and Leyton-Brown, K. 2012. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. SAT Competition.

Xu, L.; Hoos, H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. *AAAI* 210–216.

Xu, H.; Rutenbar, R.; and Sakallah, K. 2003. sub-SAT: a formulation for relaxed boolean satisfiability with applications in routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22(6):814–820.