Stable Resolving - A Randomized Local Search Heuristic for MaxSAT

Julian Reisch[1,2(✉)], Peter Großmann[1], and Natalia Kliewer[2]

[1] Synoptics GmbH, Chemnitzer Str. 48b, 01187 Dresden, Germany
{julian.reisch,peter.grossmann}@synoptics.de
[2] Freie Universität Berlin, Garystraße 21, 14195 Berlin, Germany
natalia.kliewer@fu-berlin.de
http://www.synoptics.de/

Abstract. Many problems from industrial applications and AI can be encoded as Maximum Satisfiability (MaxSAT). Often, it is more desirable to produce practicable results in very short time compared to optimal solutions after an arbitrary long computation time. In this paper, we propose Stable Resolving (SR), a novel randomized local search heuristic for MaxSAT with that aim. SR works for both weighted and unweighted instances. Starting from a feasible initial solution, the algorithm repeatedly performs the three steps of perturbation, improvements and solution checking. In the perturbation, the search space is explored at the cost of possibly worsening the current solution. The local improvements work by repeatedly flipping signs of variables in over-satisfied clauses. Finally, the algorithm performs a solution checking in a simulated annealing fashion. We compare our approach to state-of-the-art MaxSAT solvers and show by numerical experiments on benchmark instances from the annual MaxSAT competition that SR performs comparable on average and is even the best solver for particular problem instances.

Keywords: Maximum Satisfiability · MaxSAT · Incomplete solving · Randomized algorithm · Local search algorithm · Simulated annealing

1 Introduction

We consider the Constraint Satisfaction Problem of Maximum Satisfiability (MaxSAT). Many NP-hard optimization problems from applications in industry and AI can be encoded as MaxSAT and existing solution algorithms have proved to yield results that are competitive to domain specific solvers. The applications vary from periodic scheduling [15], to causal discovery [18], Bayesian network structure learning [9], correlation clustering [8], reasoning over bionetworks [16], probabilistic inference [20] and many more. A MaxSAT encoding consists a Boolean formula that we assume to be in conjunctive normal form (CNF) which means that the literals are grouped in clauses where they are connected disjunctively (*or*) and the clauses are connected conjunctively (*and*).

U. Schmid et al. (Eds.): KI 2020, LNAI 12325, pp. 163–175, 2020.
https://doi.org/10.1007/978-3-030-58285-2_12

Example. A Boolean formula in CNF: $F = (\neg x \vee \neg y) \wedge (\neg y \vee \neg z)$.

A literal is a variable together with a positive or negative sign. A clause is satisfied if at least one of its literals has the same sign as the variable in the solution. We also say the literal is *true* and denote the number of true literals in a clause its *stability*. An unsatisfied clause has a stability of zero. When a clause has a stability greater than 1, we say that the clause is over-satisfied. A solution is an assignment of the variables to *true* or $false$ and is called *feasible* if all hard clauses are satisfied. We assume the formula to consist of both hard clauses and (possibly weighted) soft clauses. The sum of (the weights of) satisfied soft clauses is the objective function value. Then, the task is to find a feasible solution maximizing the objective function value.

Example. Let $F = H_1 \wedge H_2 \wedge S_1 \wedge S_2 \wedge S_3$ where

$$
\begin{array}{lll}
H_1: & \neg x \vee \neg y & \\
H_2: & \neg y \vee \neg z & \\
S_1: & x & weight(S_1) = 2 \\
S_2: & y & weight(S_2) = 3 \\
S_3: & z & weight(S_3) = 2
\end{array}
$$

are hard and soft clauses with according weights respectively. Then, the optimal solution of value 4 is $x = z = true$ and $y = false$.

Due to its generic form, almost any problem from combinatorial optimization and many optimization problems in AI can be encoded as MaxSAT and practice shows that this conversion often works well. In this paper, we propose a novel heuristic solution approach to the MaxSAT problem called Stable Resolving (SR). The aim is to solve even large problem instances with millions of clauses and variables within short time, that is, up to 60 s, to a practicable solution. To do so, SR repeatedly performs the three steps of perturbation, improvements and solution checking, starting from an initial feasible solution. In the perturbation, the search space is explored by satisfaction of randomly picked unsatisfied (soft) clauses at the cost of other clauses becoming unsatisfied. More precisely, we consider the randomly picked clauses as hard clauses and call a SAT solver on them, together with the original hard clauses. If other, formerly satisfied soft clauses become unsatisfied by this perturbation, we write them in a list of unsatisfied candidate clauses. Then, in the improvement part, a local search technique is employed that builds on the clauses' stabilities. Starting with the first member of the list of unsatisfied candidate clauses, clauses with stability zero are being satisfied by flipping the sign of a randomly chosen variable. Flipping the sign of one of its variables increases the clause's stability by 1 but might cause other clauses to become unsatisfied. These unsatisfied clauses are added to the (local) search space and will be tried to be satisfied later. On the other hand, if flipping a variable's sign increases other clauses' stabilities to a number larger 1, that is, they become over-satisfied, they can have at least one literal falsified without becoming unsatisfied. This falsification can hence satisfy yet other clauses that contain the same variable with opposite sign and improve the objective

function again. In this way, the local search space grows until all unsatisfied clauses have been tried to satisfy. Then, the improvement step ends and if the objective function value has decreased, the previous solution is restored. Else, newly unsatisfied clauses are added to the list of unsatisfied candidate clauses. As candidate clauses are only added when the objective function value increases, and one candidate is erased when it decreases, this list will eventually be empty. Then, the solution checking part begins. Here, a worsening of the objective value is allowed with a probability that decreases during the run of the algorithm.

The outline of the paper will be as follows. After a literature overview over existing approaches in Sect. 2, we explain the algorithm in detail in Sect. 3. In Sect. 4, we present and discuss our results on common benchmark instances and finally give a conclusion and outlook in Sect. 5.

2 Related Work

There are numerous solution approaches for the MaxSAT problem both exact and heuristic ones. Let us point out the differences between SR and other state-of-the-art MaxSAT solvers. In the 2019's MaxSAT competition [1], the solver Loandra performed best in the incomplete unweighted track. It combines a core-guided approach for finding a lower bound [7] and a linear algorithm for an upper bound. As the linear algorithm, the authors use LinSBPS [13] that performs a neighborhood search in a complete algorithmic setting by repeatedly calling the SAT Solver glucose [4]. In contrast to LinSPBS, we only call glucose once at the beginning for an initial solution and for the perturbation of a solution but not in order to achieve an improvement. Moreover, we do not calculate lower bounds at all. The local search algorithms MaxRoster (a description can be found in [5]) which is based on Ramp [14] and SATLike [19] which iteratively flips the sign of variables that bring the best improvement work differently than our solver in the respect that they adapt weights of clauses in order to leave local optima. We, however, perturb a current solution for that purpose and instead of changing weights. (Max-)WalkSAT and GSAT [23] are local search approaches similar to SR in the sense that unsatisfied clauses are picked at random and one of their variables' sign is flipped. The difference to our approach is that SR searches a larger neighborhood with a more complex improvement heuristics based on stabilities. In fact, one can consider SR a large neighborhood search, as pursued in the OR world (cf. e.g. [21]), with the difference that SR finds improvements in the neighborhood heuristically and without calling an exact solver whereas the repair procedure in large neighborhood searches often involve an exact solver. At the end of each iteration, SR checks the solution in a simulated annealing fashion. Simulated annealing with reset has been used also for MaxSAT [10,17]. Finally, let us point out that the splitting of our algorithm into perturbation, improvement and solution checking was introduced for a state-of-the-art Maximum Independent Set (MIS) heuristic [3] that in a previous work, we have been able to extend by a different improvement technique and simulated annealing solution checking in order to solve MaxSAT instances that have been

transformed to MIS [22]. In contrast, in this paper we propose an algorithm that works directly on the Boolean formula.

3 Algorithm

The overall procedure of SR is shown in Algorithm 1. We first apply a SAT-based preprocessing on the formula. That is, we label the soft clauses meaning that each soft clause gets an additional variable l and will be considered a hard clause. In addition, for each label, we introduce a unit soft clause $\neg l$ with the weight the original soft clause had [6]. For the obtained equivalent formula, we apply unit clause propagation and bounded variable elimination (cf. e.g. [12]) on the hard clauses, as long as it is possible. Note that the label variables are excluded from the propagations since these operations are only sound for hard clauses. Then, for an initial feasible solution the SAT solver glucose [4] is called.

The algorithm then repeatedly executes the three steps of perturbation, improvement and solution checking.

Algorithm 1. StableResolving()

```
Preprocess()
CalculateInitialSolution()
while timeout has not been reached do
    Perturb()
    StableImprove()
    CheckSolution()
end
```

Let us explain the single parts in greater detail. In the perturbation part shown in Algorithm 2, we explore the search space. More precisely, we first sample a random number k from the geometric distribution with parameter p and select k unsatisfied clauses uniformly at random. Then, we call the SAT solver glucose on all hard clauses and the selected clauses. Additionally, we give the previous solution as an initial solution to the solver in order to speed up the computation. If this formula is feasible, we have altered the solution, but maybe at the cost of a lower objective function value because formerly satisfied clauses are now unsatisfied. These unsatisfied clauses are added to the back of a list of *candidates* that potentially can be satisfied by improvements. We keep and update this list throughout the algorithm.

Algorithm 2. Perturb()

k = random number where $\mathbb{P}[k = i] = p(1-p)^{i-1}$
$\mathcal{C}$ = set of k unsatisfied clauses picked uniformly at random
Call SAT solver on $\mathcal{C}$ and all hard clauses and overwrite the solution
Add newly unsatisfied clauses to the back of *candidates*

Example. Consider the example formula F from above. An initial feasible solution is given by all variables set to $false$. The perturbation might set $k = 1$, choose the unsatisfied clause $C = S_2$ and the SAT solver returns the feasible solution of $y = true$ and $x = z = false$. No clause gets unsatisfied by this step.

Remark. In some large instances from industrial applications, sampling a random unsatisfied clause is computationally expensive when all clauses are iterated through in order to detect the unsatisfied ones and sample among them. This is why we keep a superset of the unsatisfied clauses where every time a clause gets unsatisfied, it is added to. Moreover, we apply a heuristic in this superset and sample 1000 clause indices at random and only return if the corresponding clause indeed is unsatisfied. Only if all 1000 sampled clauses are satisfied, we iterate through the superset to find the unsatisfied clauses and sample among them.

Algorithm 3. StableImprove()

```
while candidates ≠ ∅ do
    C = pop first clause from candidates
    Init A = ∅ and 𝒞 = {C}
    while ∃v = variable picked uniformly at random in vars(𝒞) \ A do
        Flip sign of v and add v to A
        Add newly unsatisfied clauses to 𝒞
        Stab_{1→2} = set of clauses whose stability has grown to 2
        foreach S ∈ Stab_{1→2} do
            w = variable of second true literal in S
            if w is in no clause of stability 1 nor in A then
                Flip sign of w and add w to A
            end
        end
    end
    if objective function value has decreased then
        Revert flips of variables in A
    end
    else
        Add 𝒞 at the back of candidates
    end
end
```

In the improvement part shown in Algorithm 3, we iteratively pick a variable uniformly at random of an unsatisfied clause (at first from the candidates and later from the clauses that have been unsatisfied during this improvement step) and flip its sign. A flip might lead to other clauses becoming unsatisfied now and we store them in the set $\mathcal{C}$. Note that also hard clauses can become temporarily unsatisfied. On the other hand, there might be a set $Stab_{1\to 2}$ of clauses whose stability grows from 1 to 2 which means that there exists now a second true literal whose variable's sign can now be flipped without unsatisfying this clause.

This optimization technique of considering variables in over-satisfied constraints is well-known in mathematical optimization (cf. e.g. Simplex Method [11]) and we apply it here as a local improvement heuristics. In our algorithm, the clauses in $Stab_{1\to 2}$ are iterated through and checked for such an improvement. When no more variables are found that can be flipped, either because $\mathcal{C}$ is empty or all variables from $\mathcal{C}$, denoted $vars(\mathcal{C})$, are flipped already, the improvement step ends. Either the objective function value has increased, then the now unsatisfied clauses are added to the candidates, or it has not and the flips, stored in A, are reverted. Note that the feasible solution remains feasible as the objective function value cannot increase when hard clauses have become unsatisfied. The improvement part ends when there are no more candidate clauses.

Remark. In some test instances, the set $\mathcal{C}$ monotonously grows and never shrinks because there are more new unsatisfied clauses than clauses that can either be satisfied or whose variables have all been considered for an improvement. In order to avoid that we spend too much time in a single local improvement step, we set an iterations limit of 25 for the inner while-loop.

Note that both while-loops terminate. For the outer one, candidate clauses are only added if the objective function value has increased which cannot be infinitely often as MaxSAT instances are always bounded. The inner one ends - besides the iterations limit - when A contains all variables.

Example. Consider the example formula F with solution

$$(x, y, z) = (false, true, false)$$

from above. The stabilities of the following steps are illustrated in Fig. 1. S_1 is unsatisfied and might be the first candidate clause (a). Flipping the sign of its

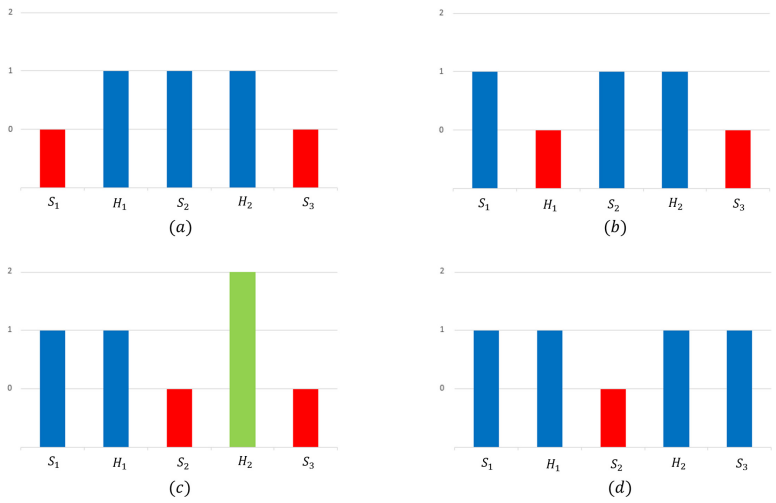

Fig. 1. Stabilities of clauses during an improvement step.

only variable x unsatisfies H_1 because y has been set to *true* in the perturbation already (b). Hence, H_1 gets added to $\mathcal{C}$ and x to A. Flipping the sign of one of the variables of H_1 (the only clause in $\mathcal{C}$) that has not already been flipped (i.e. that is not in A) means flipping y to $false$. Note that H_2 has now stability 2 and gets added to $Stab_{1\to 2}$ as both variables y and z are $false$ (c). The variable z is the second *true* literal that has been *true* before, so its sign gets flipped because no further clause is being unsatisfied by that flip. The improvement step ends with a objective function value that has increased from 3 to 4 (d).

Let us mention that during an improvement step (and after the perturbation), it is possible that formerly satisfied hard clauses become unsatisfied. Hard clauses have a weight greater than the sum of the weights of the soft clauses. Therefore, breaking hard clauses (without satisfying other formerly unsatisfied hard clauses) worsens the solution. In order to leave local optima, however, a worsening is possible in our algorithm - with decreasing probability according to the simulated annealing step, as will be explained in the remainder of this section.

Algorithm 4. CheckSolution()

```
if objective function value has increased to the best one ever seen then
    Save new best solution
end
else if objective function value has decreased then
    if number of iterations without improvement has exceeded m then
        Restore best solution
    end
    else
        Restore previous solution with probability exp(−prob)
    end
end
```

After the improvements we have arrived in a local optimum. The current solution might be of smaller objective function value than the previous solution from before this iteration of Algorithm 1 if the improvements could not compensate the perturbation. Still, we sometimes allow such a worsening in the simulated annealing approach shown in Algorithm 4 in order to be able to leave local optima. More precisely, we restore the previous solution if it had a better objective function value with a probability growing exponentially with a factor *prob* that decreases linearly during the course of the algorithm from 1 to 0 and represents the temperature of the simulated annealing. If, however, the number of iterations without an improvement exceeds a parameter m, we reset to the best solution ever seen. When SR terminates, this best solution is returned.

4 Experimental Results

We have applied SR to problem instances and compared it to results that are taken from the 2019's MaxSAT competition[1] [1]. The instances encode various industrial applications' and theoretical problems, such as scheduling, fault diagnosis, tree-width computation, max clique problems, causal discovery, Ramsey number approximation and many more. An overview of the competing solvers can be found in [5]. For all calculations, we set the parameters for the geometric distribution and maximum steps in SR to $p = 0.75$ and $m = 1000$, respectively, because they yield the best results on average. We performed all computations on an Intel Core i7-8700K and with a time limit of 60 s. Note that if a solver from [1] yields worse results on our machine than in the results of the 2019's MaxSAT competition where computations were performed on the StarExec Cluster [2], we include the better results for the analysis here. We mark such solvers with an asterisk*.

Table 1. Sum of scores by solver on unweighted instances

Loandra	LinSBPS 2018	SR	SATLike*	Open WBO g	sls mcs*	sls mcs lsu*	Open WBO ms
251.7327	238.3298	231.1436	227.4589	204.1828	202.7803	202.7158	190.9274

Table 1 and Table 2 show the sum of scores of the competing solvers on the unweighted and weighted benchmark instances, respectively, from the incomplete track of the MaxSAT competition against the scores of SR. The score of a solver on an instance is calculated in the following way. Maximizing the sum

Table 2. Sum of scores by solver on weighted instances

Loandra	236.2272
TT Open WBO Inc*	233.4784
LinSBPS2018	231.6581
Open WBO Inc (inc bmo satlike)*	220.3607
Open WBO Inc (inc bmo complete)*	218.6454
SR	213.3262
Open WBO g*	212.1081
SATLike*	210.6802
sls mcs2*	203.1498
Open WBO ms*	194.5451
sls mcs*	191.4503
uwrmaxsat inc*	190.7841

[1] We have submitted SR to the 2020's MaxSAT competition.

(of weights) of satisfied soft clauses is equivalent to minimizing the sum (of weights) of unsatisfied soft clauses, which is denoted by the *gap*. The score of a solver on an instance is the fraction of the best gap known divided by the gap of the particular solver. If a solver's solution violates a hard clause, its score is zero and its gap infinity.

Example. Consider the example above. In the optimal solution, only S_2 is unsatisfied which yields the optimal gap of 3. If solver A has achieved this optimum, solvers B and C satisfy only S_2 and not S_1 and S_3, then their scores are $3/4$ while solver A has the maximal score of 1 on this instance.

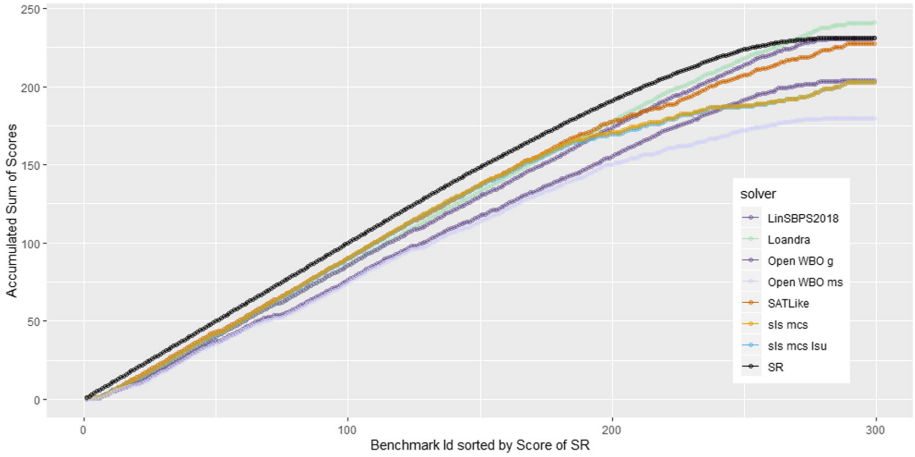

Fig. 2. Accumulated sum of scores of unweighted instances after 60 s computation time

The score hence reflects the ratio of the achieved result to the optimal (or best known) one. In Fig. 2 and Fig. 3, we see the accumulated sum of scores of the single instances, ordered by SR's scores and grouped by the competing solvers for the unweighted and weighted instances, respectively. The figures illustrate that SR (black) has the highest sum of scores on a large subset of instances. Counting all instances, including those where SR has low scores, we conclude that SR still has a competitive performance. More precisely, in 210 and 179 of the 299 unweighted and 297 weighted instances, SR has a score at least the mean of the other solvers. Furthermore, SR performs especially well on the unweighted instances, in comparison to the other solvers.

What is more, SR often has the best result among all solvers for particular instances. We observe that in the unweighted case, SR performs especially well on instances from *atcos*, *extension enforcement* and *set covering*. In the weighted case, SR is best on many instances encoding the *Minimum Weight Dominating Set Problem*. See Tables 3 and 4 for complete lists of such instances in the unweighted and weighted case, respectively. For a better comparison, we include a column showing the gaps of the winning solver Loandra, as well.

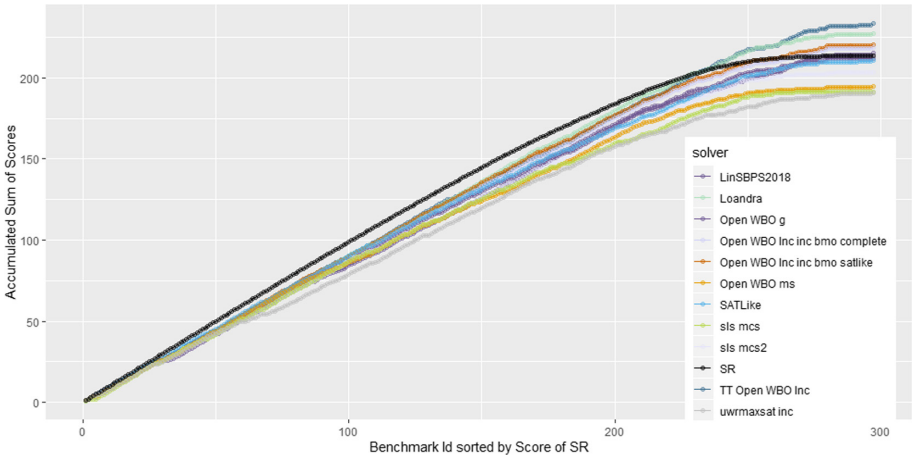

Fig. 3. Accumulated sum of scores of weighted instances after 60 s computation time

Table 3. Gaps of unweighted instances where SR performs best

	Benchmark	SR	Loandra
1	aes/sbox-8.wcnf.gz	443	690
2	atcoss/mesat/atcoss-mesat-04.wcnf.gz	97	Inf
3	atcoss/mesat/atcoss-mesat-05.wcnf.gz	74	Inf
4	atcoss/mesat/atcoss-mesat-10.wcnf.gz	32	40
5	atcoss/mesat/atcoss-mesat-18.wcnf.gz	80	Inf
6	atcoss/sugar/atcoss-sugar-15.wcnf.gz	133	Inf
7	extension-enforcement/extension-enforcement-non-strict-stb-200-0.05-1-10-0.wcnf.gz	7	18
8	extension-enforcement/extension-enforcement-non-strict-stb-200-0.05-2-10-2.wcnf.gz	12	20
9	extension-enforcement/extension-enforcement-non-strict-stb-200-0.05-3-10-1.wcnf.gz	8	9
10	extension-enforcement/extension-enforcement-non-strict-stb-200-0.05-3-10-4.wcnf.gz	10	16
11	extension-enforcement/extension-enforcement-non-strict-stb-200-0.05-4-10-1.wcnf.gz	7	17
12	extension-enforcement/extension-enforcement-non-strict-stb-200-0.05-4-10-2.wcnf.gz	6	13
13	extension-enforcement/extension-enforcement-non-strict-stb-200-0.05-4-10-4.wcnf.gz	8	18
14	min-fill/MinFill-R3-miles1000.wcnf.gz	3634	3755
15	optic/gen-cvc-add7to3-9999.wcnf.gz	197	204
16	pseudoBoolean/garden/normalized-g100x100.opb.msat.wcnf.gz	2163	2526
17	railway-transport/d4.wcnf.gz	8296	8524
18	SeanSafarpour/wb-4m8s1.dimacs.filtered.wcnf.gz	58	282
19	SeanSafarpour/wb-4m8s4.dimacs.filtered.wcnf.gz	220	230
20	set-covering/crafted/scpclr/scpclr13-maxsat.wcnf.gz	27	28
21	set-covering/crafted/scpcyc/scpcyc07-maxsat.wcnf.gz	145	149
22	set-covering/crafted/scpcyc/scpcyc08-maxsat.wcnf.gz	363	390
23	set-covering/crafted/scpcyc/scpcyc09-maxsat.wcnf.gz	835	972
24	set-covering/crafted/scpcyc/scpcyc10-maxsat.wcnf.gz	1967	2242
25	set-covering/crafted/scpcyc/scpcyc11-maxsat.wcnf.gz	4771	5623
26	uaq/uaq-ppr-nr200-nc66-n5-k2-rpp4-ppr12-plb100.wcnf.gz	75	78
27	xai-mindset2/liver-disorder.wcnf.gz	316	318

Table 4. Gaps of weighted instances where SR performs best

	Benchmark	SR	Loandra
1	causal-discovery/causal-Water-10-1000.wcnf.gz	11339025	16041455
2	causal-discovery/causal-Wdbc-8-569.wcnf.gz	1446339	2541316
3	correlation-clustering/Rounded-CorrelationClustering-Vowel-BINARY-N740-D0.200.wcnf.gz	120199215	130874895
4	correlation-clustering/Rounded-CorrelationClustering-Vowel-BINARY-N760-D0.200.wcnf.gz	120800405	132256968
5	drmx-cryptogen/geffe128-7.wcnf.gz	812	846
6	min-width/MinWidthCB-mitdbsample-100-43-1k-5s-2t-5.wcnf.gz	32010	32200
7	min-width/MinWidthCB-mitdbsample-200-64-1k-2s-1t-4.wcnf.gz	76975	78325
8	min-width/MinWidthCB-mitdbsample-300-43-1k-6s-1t-8.wcnf.gz	45780	45825
9	MinimumWeightDominatingSetProblem/delaunay-n24.wcnf.gz	304532225	350820532
10	MinimumWeightDominatingSetProblem/hugebubbles-00020.wcnf.gz	694937186	753286458
11	MinimumWeightDominatingSetProblem/inf-road-usa.wcnf.gz	840126999	903206743
12	MinimumWeightDominatingSetProblem/sc-rel9.wcnf.gz	15590036	16746750
13	MinimumWeightDominatingSetProblem/web-wikipedia2009.wcnf.gz	28120892	37674803
14	pseudoBoolean/miplib/normalized-mps-v2-20-10-p0548.opb.msat.wcnf.gz	12451	25494
15	spot5/log/1401.wcsp.log.wcnf.gz	463106	469110
16	spot5/log/1407.wcsp.log.wcnf.gz	459591	465638

5 Conclusion and Outlook

In this paper, we have proposed a novel local search algorithm for solving large MaxSAT problems in short time. We could prove by numeric experiments on benchmark instances encoding problems from combinatorial optimization and AI that our algorithm yields results that are comparable to and for some problem families even better than state-of-the-art solvers.

As a possible prospect, we aim at developing more sophisticated improvement methods that take into account not single over-satisfied clauses but sets of such. Also, we can think of caching unsuccessful local improvements so that they will never be performed a second time. Finally, we want to analyse the different components of our algorithm by replacing each of the perturbation, stable improvements and simulated annealing by a naive technique. This will give an insight into the contribution of each component to the solvers performance.

References

1. MaxSAT Evaluation 2019. https://maxsat-evaluations.github.io/2019/index.html
2. Starexec Cluster. https://www.starexec.org/starexec/public/about.jsp. Accessed 2019
3. Andrade, D.V., Resende, M.G.C., Werneck, R.F.F.: Fast local search for the maximum independent set problem. J. Heuristics **18**(4), 525–547 (2012). https://doi.org/10.1007/s10732-012-9196-4
4. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, San Francisco, CA, USA, pp. 399–404 (2009)

5. Bacchus, F., Järvisalo, M., Martins, R.: MaxSAT evaluation 2018: new developments and detailed results. J. Satisf. Boolean Model. Comput. **11**, 99–131 (2019). https://doi.org/10.3233/SAT190119
6. Belov, A., Morgado, A., Marques-Silva, J.: SAT-based preprocessing for MaxSAT. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 96–111. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45221-5_7
7. Berg, J., Demirović, E., Stuckey, P.J.: Core-boosted linear search for incomplete MaxSAT. In: Rousseau, L.-M., Stergiou, K. (eds.) CPAIOR 2019. LNCS, vol. 11494, pp. 39–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19212-9_3
8. Berg, J., Järvisalo, M.: Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. Artif. Intell. **244**, 110–142 (2017). https://doi.org/10.1016/j.artint.2015.07.001. Combining Constraint Solving with Mining and Learning
9. Berg, J., Järvisalo, M., Malone, B.: Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In: Kaski, S., Corander, J. (eds.) Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 33, pp. 86–95. PMLR, Reykjavik, 22–25 April 2014
10. Bouhmala, N.: Combining simulated annealing with local search heuristic for MaxSAT. J. Heuristics **25**(1), 47–69 (2019). https://doi.org/10.1007/s10732-018-9386-9
11. Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press, Princeton (1963)
12. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960). https://doi.org/10.1145/321033.321034
13. Demirović, E., Stuckey, P.J.: Techniques inspired by local search for incomplete MaxSAT and the linear algorithm: varying resolution and solution-guided search. In: Schiex, T., de Givry, S. (eds.) CP 2019. LNCS, vol. 11802, pp. 177–194. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30048-7_11
14. Fan, Y., Ma, Z., Su, K., Sattar, A., Li, C.: Ramp: a local search solver based on make-positive variables. In: MaxSAT Evaluation (2016)
15. Großmann, P., Hölldobler, S., Manthey, N., Nachtigall, K., Opitz, J., Steinke, P.: Solving periodic event scheduling problems with SAT. In: Jiang, H., Ding, W., Ali, M., Wu, X. (eds.) IEA/AIE 2012. LNCS (LNAI), vol. 7345, pp. 166–175. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31087-4_18
16. Guerra, J., Lynce, I.: Reasoning over biological networks using maximum satisfiability. In: Milano, M. (ed.) CP 2012. LNCS, pp. 941–956. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33558-7_67
17. Hoos, H.H.: Solving hard combinatorial problems with GSAT—A case study. In: Görz, G., Hölldobler, S. (eds.) KI 1996. LNCS, vol. 1137, pp. 107–119. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61708-6_53
18. Hyttinen, A., Eberhardt, F., Järvisalo, M.: Constraint-based causal discovery: conflict resolution with answer set programming. In: Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence, pp. 340–349 (2014)
19. Lei, Z., Cai, S.: Solving (weighted) partial MaxSAT by dynamic local search for sat. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, pp. 1346–1352. International Joint Conferences on Artificial Intelligence Organization, July 2018. https://doi.org/10.24963/ijcai.2018/187

20. Park, J.D.: Using weighted Max-SAT engines to solve MPE. In: Proceedings of the 18th National Conference on Artificial Intelligence, pp. 682–687 (2002)
21. Pisinger, D., Ropke, S.: Large neighborhood search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146, pp. 399–419. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-1665-5_13
22. Reisch, J., Großmann, P., Kliewer, N.: Conflict resolving - a maximum independent set heuristics for solving MaxSAT. In: Proceedings of the 22nd International Multiconference Information Society, vol. 1, pp. 67–71 (2019)
23. Selman, B., Kautz, H., Cohen, B.: Local search strategies for satisfiability testing. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 521–532 (1995)