Reduced Cost Fixing in MaxSAT*

Fahiem Bacchus[1], Antti Hyttinen[2], Matti Järvisalo[2], and Paul Saikko[2]

[1] Department of Computer Science, University of Toronto, Canada
[2] HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract. We investigate utilizing the integer programming (IP) technique of reduced cost fixing to improve maximum satisfiability (MaxSAT) solving. In particular, we show how reduced cost fixing can be used within the implicit hitting set approach (IHS) for solving MaxSAT. Solvers based on IHS have proved to be quite effective for MaxSAT, especially on problems with a variety of clause weights. The unique feature of IHS solvers is that they utilize both SAT and IP techniques. We show how reduced cost fixing can be used in this framework to conclude that some soft clauses can be left falsified or forced to be satisfied without influencing the optimal cost. Applying these forcings simplifies the remaining problem. We provide an extensive empirical study showing that reduced cost fixing employed in this manner can be useful in improving the state-of-the-art in MaxSAT solving especially on hard instances arising from real-world application domains.

1 Introduction

Maximum satisfiability (MaxSAT) [17] is a thriving constraint optimization paradigm, successfully applied in a growing number of NP-hard real-world problem domains. The currently most successful MaxSAT solvers are SAT-based, i.e., rely on Boolean satisfiability solver technology [4]. In particular, they use SAT solvers to iteratively extract unsatisfiable cores (unsatisfiable sets of soft clauses) and block these cores from the search in the later iterations, until a solution is found. One of the currently most successful algorithmic approaches—as witnessed by the most recent MaxSAT Evaluations [2]—are solvers implementing the so-called implicit hitting set (IHS) approach for MaxSAT. IHS MaxSAT solvers [8, 9, 11, 23, 24] employ a hybrid approach that exploits both a SAT solver for core extraction and an integer programming (IP) solver for obtaining minimum cost hitting sets of the accumulated cores.

Despite the success and recent algorithmic advances in MaxSAT solvers, the SAT-based MaxSAT solvers do not—as witnessed by the empirical results presented in this paper—currently harness the full potential of bounds-based problem simplification during search. Focusing on IHS as the approach which solved the most instances in the general weighted partial category of the 2016 MaxSAT Evaluation, we propose to take advantage of classical ideas from the realm of integer programming to further improve state-of-the-art MaxSAT solvers. In more detail, we show how to integrate *reduced cost*

* Work supported in part by Academy of Finland (grants 251170 COIN, 276412, 284591 and 295673), the Research Funds and DoCS Doctoral School in Computer Science of the University of Helsinki, and the Natural Sciences and Engineering Research Council of Canada.

fixing [7, 6, 22], a standard technique in IP solving that uses bounds on the optimal cost derived during search for inferring variables whose values can be fixed while preserving at least one optimal solution. As we will explain in detail, in terms of MaxSAT search, reduced cost fixing amounts to using upper bounds obtained during search to harden or falsify specific soft clauses, i.e., to force them to be satisfied or falsified. The IHS approach to MaxSAT is a prime candidate for integrating reduced cost fixing since the reduced costs of soft clauses can be readily obtained by solving a linear (LP) relaxation of the hitting set problem maintained during IHS search. Putting this idea into practice, we extend the IHS solver MaxHS with reduced cost fixing, and provide an extensive empirical evaluation showing that reduced cost fixing considerably speeds up MaxHS.

In terms of related work, different techniques of using lower and lower bounds for speeding up MaxSAT solver have studied in varying contexts, including branch-and-bound for MaxSAT [16, 18, 19, 15], use of bounds for MaxSAT solvers in general [13], and hardening based on SAT inferred costs of residual formulas in pure SAT-based core-guided MaxSAT solving [1, 21]. However, to the best of our knowledge, linear programming relaxation based reduced cost fixing has not been previously proposed in the context of MaxSAT. There has, however, been a number of related works exploiting the technique of reduced cost fixing in constraint programming, IP/constraint logic programming, and IP/constraint programming, e.g., [26, 12, 28].

After background on MaxSAT (Sect. 2), we give a bounds-based view of the IHS approach to MaxSAT (Sect. 3), explain how to integrate reduced cost fixing into it (Sect. 4), and present empirical results on the effectiveness of reduced cost fixing in speeding up the IHS solver MaxHS (Sect. 5).

2 Maximum Satisfiability

We work with propositional formulas expressed in conjunctive normal form (CNF). Satisfaction of CNF formulas is defined as usual. Whenever convenient we treat a clause as a set of literals and a CNF formula as a set of clauses. An instance of (weighted partial) maximum satisfiability (MaxSAT) $F = (F_h, F_s, wt)$ consists of two CNF formulas: the hard clauses F_h, the soft clauses F_s and a weight function $wt\colon F_s \to \mathbb{Q}$ associating a positive rational weight to each soft clause. Given such an instance, any truth assignment τ that satisfies the hard clauses is a **solution** to F. The **cost** of a solution τ, $cost(F, \tau)$, is the sum of the weights of the soft clauses it falsifies: $cost(F, \tau) = \sum_{\{\tau \not\models C | C \in F_s\}} wt(C)$. A solution τ is optimal if $cost(F, \tau) \leq cost(F, \tau')$ for all solutions τ'. Given an instance F the MaxSAT problem is to find an optimal solution to F. We denote the cost of optimal solutions to F by $opt\_cost(F)$. We also use $cost(S)$, for any set of soft clauses S, to denote the sum of weights of the soft clauses in S: $cost(S) = \sum_{c \in S} wt(c)$. For a MaxSAT instance $F = (F_h, F_s, wt)$, an unsatisfiable **core** of F is any subset $S \subseteq F_s$ of soft clauses such that $F_h \cup S$ is unsatisfiable.

3 The SAT-IP Implicit Hitting Set Approach to MaxSAT

IHS MaxSAT solvers [8, 9, 11, 10, 23, 24] utilize the so-called implicit hitting set approach [14, 20, 25] to solve weighted partial MaxSAT. These solvers use a SAT solver

Algorithm 1: The IHS approach to MaxSAT (generalized from [8])

```
1  IHS-MaxSAT (F = (F_h, F_s, wt))
2  (sat, κ, τ) ← SolveSAT(F_h)                    /* If unsat return a core κ, else a solution τ */
3  if not sat then return "No solutions since F_h is UNSAT"
4  UB ← cost(F, τ); best_τ ← τ; LB = 0                                   /* Initial bounds */
5  Optimizer.initialze(wt); new_cores ← ∅; hs_is_sat ← hs_is_opt ← false
6  while UB > LB do
7      (hs_is_opt, HS) ← Optimizer(new_cores, UB)
8      if hs_is_opt then LB = cost(HS)
9      (hs_is_sat, κ, τ) ← SolveSAT(F_h ∪ (F_s \ HS))
10     if not hs_is_sat then
11         repeat
12             new_cores ← new_cores ∪ κ
13             (sat, κ, τ) ← SolveSAT(F_n ∪ (F_s \ (HS ∪ ⋃_{κ∈new_cores} κ)))
14         until sat
15     if cost(τ) < UB then UB ← cost(τ); best_τ ← τ
16 return best_τ
```

to accumulate cores and an IP solver to compute a minimum-cost hitting set of the accumulated cores. Since each core is unsatisfiable, any solution must falsify at least one soft clause in every core, i.e., the set of soft clauses falsified by any solution must form a hitting set of the set of cores. Therefore, the cost of any solution is lower-bounded by the cost of the minimum-cost hitting. Further, if these costs are equal, then the solution must be optimal. As first described in [8] an iteration can be set up that ensures that the IHS solver finds an optimal solution after producing a finite number of cores.

This original algorithm does not, however, provide the upper bounds needed for reduced cost fixing. Upper bounds can be obtained by using non-optimal hitting sets as described in [11]. We give, in Algorithm 1, a new more general formalization of the algorithm described in [11] and a more general correctness condition.

The algorithm first computes an initial model by solving F_h. The returned τ is also a solution to F, and provides an initial upper bound once we check which clauses of F_s are satisfied by τ.

The **Optimizer** maintains the set of cores passed to it, adding the cores in new_cores to this set (line 7). It always returns a hitting set HS of its current set of cores, and a flag (hs_is_opt) indicating whether it has verified HS to be of minimum cost. (HS might be of minimum cost even if the **Optimizer** has not verified this). If HS is of minimum cost, its cost is a valid lower bound on $opt\_cost(F)$ and we can update LB. Note that **Optimizer**'s set of cores can only grow so the cost of a minimum-cost hitting set cannot decrease and the updates never decrease LB.

The SAT solver tests if removing HS from F_s results in satisfiability; if not we obtain a new core, κ, and add it to the set of new_cores. We then enter a loop where we accumulate more cores, repeatedly removing all of the soft clauses in HS and all newly discovered cores (cf. the "disjoint,g" strategy in [24]). At each step a new core is found and the set of soft clauses passed to the SAT solver is further reduced. Since F_h

is satisfiable, the loop must terminate as we will eventually remove enough soft clauses to obtain satisfiability. We then update the upper bound if the found solution has lower cost.

The algorithm terminates when it finds a solution whose cost achieves the lower bound. Such a solution must be optimal; hence Algorithm 1 always returns an optimal solution. We also have that Algorithm 1 must terminate as long as **Optimizer** satisfies the following general condition. During Algorithm 1 a sequence of calls are made to **Optimizer** (once every iteration of the while loop). In each call **Optimizer** computes a hitting set of the accumulated set of cores passed to it in the current and all previous calls, and during that call UB is the best known upper bound.

Definition 1 (Correctness Condition) ***Optimizer*** *always returns a hitting set of its accumulated set of cores. And, for every i there exists an $k > i$ such that the k'th call to* ***Optimizer*** *returns a hitting set, HS, such that either (a) $cost(HS) < UB$ or (b) HS is a minimum-cost hitting set.*

Theorem 1. *If* ***Optimizer*** *satisfies the correctness condition, then Algorithm 1 must eventually terminate returning an optimal solution.*

Proof. We show that the sequence of calls to **Optimizer** is finite, and thus the while loop must terminate. In fact, we need only consider the sub-sequence calls consisting of those calls where **Optimizer** returns a minimum-cost hitting set or a hitting set with cost less than the current upper bound. By the correctness condition this sub-sequence is infinite iff Algorithm 1 fails to terminate. We say that a hitting set HS returned by **Optimizer** is infeasible if $F_h \cup (F_s \setminus HS)$ is unsatisfiable, otherwise it is feasible. Note that when HS is feasible, we will have $cost(\tau) \leq cost(HS)$ for the returned model τ, and $UB \leq cost(HS)$ after line 15.

Optimizer cannot return an infeasible hitting set HS more than once: HS will cause a core to be added to **Optimizer** that HS does not hit, so HS will not be a hitting set for any subsequent calls. In the sub-sequence **Optimizer** can never return a feasible hitting set HS more than twice. After HS is returned we have that $UB \leq cost(HS)$. If **Optimizer** also returned $hs\_is\_opt = true$, then LB will become equal to UB and the algorithm will terminate. Otherwise, if $hs\_is\_opt = false$, the lowered UB implies that if HS is returned once more in the subsequence it must be with $hs\_is\_opt = true$, which will cause termination. There are only a finite number of hitting sets, so the sub-sequence must be finite, and Algorithm 1 must terminate. □

In the version of MaxHS reported on in our empirical evaluation **Optimizer** utilizes both a heuristic greedy solver and an exact IP solver (IBM CPLEX). It always uses the greedy solver unless it is passed an empty set new_cores (which happens when the previous call to **Optimizer** returned a feasible hitting set). For an empty new_cores it uses the IP solver to compute a hitting set. However, it does not ask the IP solver to compute a minimum-cost hitting set. Rather it stops the IP solver as soon as a hitting set with cost less than UB has been found. When UB is already equals the optimal cost, the IP solver will run to completion as a lower cost hitting set will never be found. In this case the IP solver will find a hitting set that it can verify to be of minimum cost, and this hitting set and $hs\_is\_opt = true$ is returned. This scheme is used to reduce the number of times the hitting set problem needs to be solved to optimality [11].

4 Reduced Cost Fixing

Reduced cost fixing is a standard technique in OR [27, 7, 6, 22]. It uses an upper bound and reduced costs obtained from an LP relaxation to fix variables in an IP. Given a minimization IP P containing Boolean (0/1) variables, we can solve P as an LP by allowing the Boolean variables to take on intermediate values between 0 and 1. The cost of the LP solution will be a lower bound on the optimal cost of P. The LP solver also provides a reduced cost for the non-basic[3] variables set at 0 or 1 in the LP solution. These reduced costs specify the influence of changing a non-basic variable at 0 (1) to 1 (0) on the cost of the LP. Suppose we know a feasible IP solution to P with cost z. If changing a non-basic variable causes the LP solution to increase in cost beyond z, then we can fix that variable to the value it has in the LP solution. Since the LP solution is a lower bound, putting such variables at their opposite values would cause the cost of the IP to increase beyond the cost of an already known feasible solution.

Here we explain how this technique can be used within IHS MaxSAT solvers. In contrast with standard uses of reduced cost fixing we do not want to fix variables of the IP (our IP is the IP of the hitting set problem). Rather we want to fix variables of the MaxSAT problem from which the IP has been derived. This can be done as follows.

Theorem 2. *For a MaxSAT problem $F = (F_h, F_s, wt)$, suppose we have (a) $B = \{b_1, \ldots, b_n\}$ a set of Boolean variables where each $b_i = 0$ ($b_i = 1$) represents the satisfaction (falsification) of soft clause $c_i \in F_s$, (b) IP_{HS} an IP over the b_i representing the minimum-cost hitting set problem over the current set of cores, (c) LP_{HS} the LP relaxation of IP_{HS}, (d) $best\_\tau$ a feasible solution to F, (e) an optimal solution to LP_{HS} with cost $z_{opt}^{LP_{HS}}$, and (f) LP reduced costs $rc(b_i)$ at the optimal basis.*

Then the following simplifications can be performed without changing $opt\_cost(F)$. (1) For every non-basic variable b_i set to 0 in the optimal LP_{HS} solution we can make soft clause c_i hard in F if $z_{opt}^{LP_{HS}} + rc(b_i) > cost(best\_\tau)$ or if $z_{opt}^{LP_{HS}} + rc(b_i) = cost(best\_\tau)$ and c_i is satisfied in $best\_\tau$. (2) For every non-basic variable b_i set to 1 in the optimal LP_{HS} we can make soft clause c_i false in F if $z_{opt}^{LP_{HS}} - rc(b_i) > cost(best\_\tau)$ or if $z_{opt}^{LP_{HS}} - rc(b_i) = cost(best\_\tau)$ and c_i is falsified in $best\_\tau$.

Proof. Let b_i be a non-basic variable at its lower bound in the optimal solution to LP_{HS}. Then either $b_i = 1$ is feasible in LP_{HS} or it is not.[4] If it is not, then, since LP_{HS} is a relaxation of IP_{HS}, $b_i = 1$ is also infeasible in IP_{HS}. Furthermore, since every core is a logical consequence of F, IP_{HS} is a relaxation of F and thus $c_i = false$ must be infeasible in F, and we can harden c_i. On the other hand, if $b_i = 1$ is feasible in LP_{HS}, then by the properties of reduced costs, forcing $b_i = 1$ will increase the optimal cost of LP_{HS} by at least $rc(b_i)$ [3]. Stated a different way, if LP^+ is LP_{HS} with the added constraint $b_i = 1$, then its optimal cost will be at least $z^{LP_{HS}} + rc(b_i)$. LP^+ is the linear relaxation of IP^+, which is IP_{HS} with the added constraint $b_i = 1$; and IP^+

[3] The variables in the LP solution are either basic or non-basic. All of the non-basic variables will be at their upper or lower bounds in the LP solution [5].

[4] In a hitting set problem $b_i = 1$ is always feasible. However, MaxHS can also add other constraints to the hitting set problem via a process of constraint seeding [9]. It is not difficult to show that all of our results continue to hold with seeding.

is a relaxation of F^+ which is $F \cup \neg c_i$. Hence, $cost(F^+) >= z^{LP_{HS}} + rc(b_i)$ and if $z^{LP_{HS}} + rc(b_i) > cost(best\_\tau)$, or if $z^{LP_{HS}} + rc(b_i) = cost(best\_\tau)$ and c_i is satisfied in $best\_\tau$, then we can force c_i to be satisfied in F while still preserving at least one of the optimal solutions of F. The argument for b_i at its upper bound is analogous. □

In Algorithm 1 reduced cost fixing can be utilized whenever $UB-LB$ decreases and is small enough to allow the forcing of some unforced soft clause. In particular, $rc(b_i)$ is upper-bounded by $wt(c_i)$ and hence c_i cannot be forced if $(UB - LB) > wt(c_i)$. We use CPLEX to solve the LP relaxation of the hitting set problem to obtain the reduced costs; we do this just before invoking CPLEX in **Optimizer**.

5 Experiments

We implemented reduced cost fixing in MaxHS v2.9.8 which entered the 2016 MaxSAT Evaluation. This version of MaxHS included a number of other features shown to improve the solver, described in [9, 11, 23]. We compare the performance of MaxHS with and without reduced cost fixing, with all other features unchanged. We utilized IBM CPLEX v12.7 as the IP/LP solver, and ran our experiments on computing nodes with Xeon 2.8-GHz cores and 256-GB RAM. We limited MaxHS to 1800 seconds and 3.5 GB on each instance. We also report on longer 5-hour (18,000 s), 5-GB runs on Xeon 2.0-GHz cores and 256-GB RAM.

We experimented with all non-random instances that have been collected by and made available by the MaxSAT Evaluation during the years 2008 to 2016. These include extra submitted benchmarks never used in the evaluation. After pruning duplicate instances this yielded 6290 MaxSAT instances (4361 unweighted, 1929 weighted). For the 5-hour runs, however, we omitted 507 unweighted instances with no hard clauses (MS instances) most of which encode MaxCut on random graphs. Core-based solvers, including IHS solvers, perform poorly on such instances, and we did not expect any of these instances to complete in 5 hours with or without reduced cost fixing. This left 5783 instances to run in these longer experiments (4361 unweighted, 1422 unweighted).

First we examine how frequently reduced cost fixing occurs in our benchmark suite. Figure 1 left shows a histogram of the instances grouped by the number of soft clauses that become fixed during solving. In 5024 of the 6290 instances no reduced cost fixing ever occurs (3953 unweighted, 1071 weighted), but in the remaining 1266 instances fixing can be quite common—in 791 of these instances 100 or more fixings occurred. In extreme cases over a million soft clauses were fixed by the technique (this makes average number of soft clauses fixed misleadingly large). There was little difference in the histograms between weighted and unweighted instances once the zero fixing instances were removed; in fact, the instance with the most fixings was unweighted.

The second question is how much overhead does reduced cost fixing incur, particularly since the LP is solved even when no fixing occurs. There were 26 instances where fixing took more than 100 seconds. However, 25 of these were not solvable with or without fixing (22 were MaxCut on random graphs). On one solved instance fixing required 214 seconds out of a total solve time of 835 seconds (this instance was solved in 416 seconds without fixing). Of the remaining 6264 instances, on 1782 instances fixing took zero seconds (LP solving was never invoked since the gap between UB and

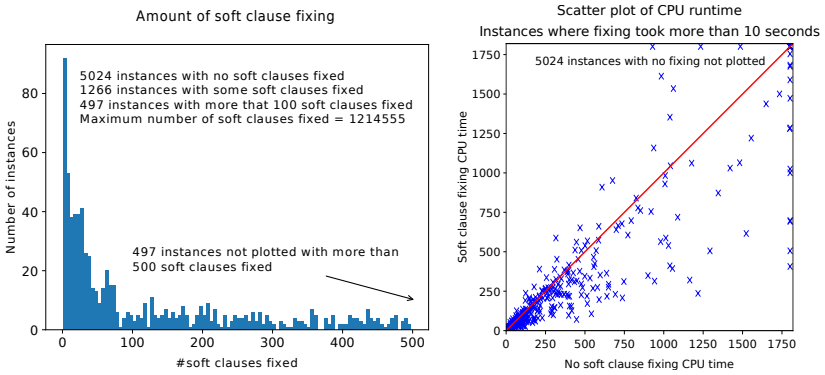

Fig. 1. Left: distribution of the frequency reduced cost fixing forces a soft clause to be relaxed or hardened. Right: Scatter plot showing that fixing on instances where fixing takes significant time also pays off.

LB was never small enough), on 3746 instances fixing took less than 1 second, on 298 instances fixing took between 1 and 10 seconds, and on 438 instances fixing took more than between 10 and 100 seconds. Figure 1 right shows, however, that on these 438 instances fixing is well worth the time it takes. The scatter plot shows that fixing provides a significant speedup for most of these instances, especially on the harder instances.

In the rest of our plots we omit data from the 5024 instances on which no reduced cost fixing occurred. We omitted these instances because their run times will only vary by the overhead of fixing (and experimental variances induced by varying cluster loads), and we have already provided data in the previous paragraph showing that this overhead is not significant.

Figure 2 shows scatter plots for all instances, all unweighted instances and all weighted instances. The plots show that fixing generally provides a speedup, and that speedups occur on both weighted and unweighted instances.

In Figure 3 we show in more detail the performance improvement obtained from reduced cost fixing. Here we computed the speedup ratio for each instance, i.e., the CPU time taken without reduced cost fixing divided by the CPU time taken when reduced cost fixing is used. As this ratio will be between 0 and 1, for instances that are slowed down by fixing we took $\log_2$ of this ratio which produces a symmetry between speedups and slowdowns. The plots are in the form of histograms showing for how many instances experience various ranges of the $\log$ speedup. Figure 3 left shows the $\log$ speedup ratio for all instances, while on the right we examine the 4361 instances that were run under a per-instance time limit of 5 hours.

These histograms verify the value of our technique for exploiting reduced cost fixing in IHS based MaxSAT solvers. When we look at the data from the 5-hour runs we see an even more pronounced effect with fewer instances being slowed down, and a smoother distribution for the instances being speeded up.

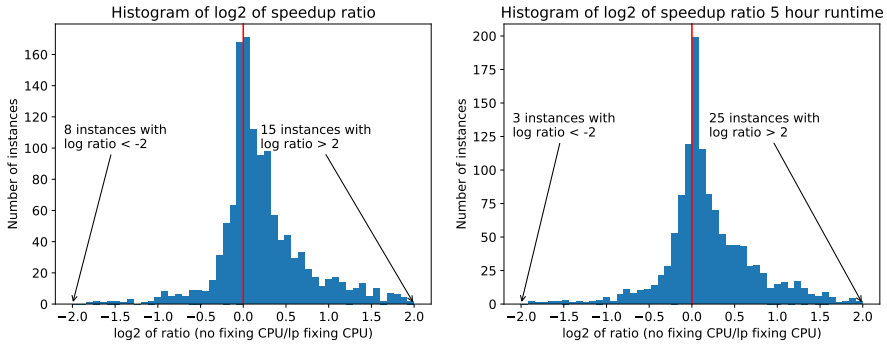

Fig. 2. Speedup histograms over instances on which reduced cost fixing would force some variables in terms of $\log_2$ of CPU time with fixing and without fixing. Left: under 30-minute per-instance time limit, right: under 5-hour per-instance time limit.

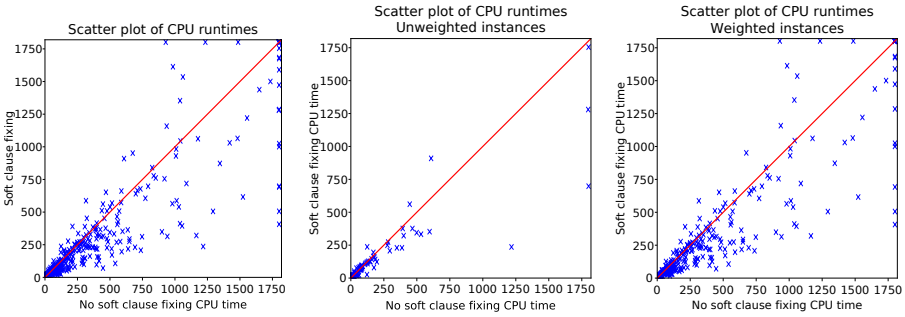

Fig. 3. Scatter plots of CPU times with and without reduced cost fixing, omitting instances 5024 where no fixing occurred. Left: all instances; middle: unweighted instances; right: weighted. instances.

6 Conclusions

We proposed the use of reduced cost fixing—a standard approach in IP—in MaxSAT solving as a means of utilizing bounds information during search to infer knowledge of soft clauses which are satisfied or left falsified by some optimal solutions. We explained how reduced cost fixing can be integrated into the implicit hitting set approach to MaxSAT by performing reduced cost analysis directly on the LP relaxation of the hitting-set IP already utilized in the IHS search routine. We showed through an extensive empirical evaluation that reduced cost fixing can provide considerable speedups improving on the overall performance of MaxHS.

References

1. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving WPM2 for (weighted) partial MaxSAT. In: Proc. CP. Lecture Notes in Computer Science, vol. 8124, pp. 117–132. Springer

(2013)
2. Argelich, J., Li, C.M., Manyà, F., Planes, J.: MaxSAT Evaluation 2016 (Accessed April 27, 2017), http://maxsat.ia.udl.cat/introduction/
3. Bajgiran, O.S., Ciré, A.A., Rousseau, L.: A first look at picking dual variables for maximizing reduced cost fixing. In: Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings. pp. 221–228 (2017)
4. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. IOS Press (2009)
5. Chvátal, V.: Linear Programming. Freeman (1983)
6. Crowder, H., Johnson, E.L., Padberg, M.: Solving large-scale zero-one linear programming problems. Operations Research 31(5), 803–834 (1983)
7. Danzig, G., Fulkerson, D., Johnson, S.: Solution of a large-scale traveling-salesman problem. Operations Research 2, 393–410 (1954)
8. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Proc. CP. Lecture Notes in Computer Science, vol. 6876, pp. 225–239. Springer (2011)
9. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MaxSAT. In: Proc. SAT. Lecture Notes in Computer Science, vol. 7962, pp. 166–181. Springer (2013)
10. Davies, J.: Solving MAXSAT by Decoupling Optimization and Satisfaction. Ph.D. thesis, University of Toronto (2013), http://www.cs.toronto.edu/~jdavies/Davies_Jessica_E_201311_PhD_thesis.pdf
11. Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: Proc. CP. Lecture Notes in Computer Science, vol. 8124, pp. 247–262. Springer (2013)
12. Focacci, F., Lodi, A., Milano, M.: Cost-based domain filtering. In: Proc. CP. Lecture Notes in Computer Science, vol. 1713, pp. 189–203. Springer (1999)
13. Heras, F., Morgado, A., Marques-Silva, J.: Lower bounds and upper bounds for MaxSAT. In: Proc. LION 6. Lecture Notes in Computer Science, vol. 7219, pp. 402–407. Springer (2012)
14. Karp, R.M.: Implicit hitting set problems and multi-genome alignment. In: Proc. CPM. Lecture Notes in Computer Science, vol. 6129, p. 151. Springer (2010)
15. Li, C.M., Manyà, F., Mohamedou, N.O., Planes, J.: Transforming inconsistent subformulas in MaxSAT lower bound computation. In: Proc. CP. Lecture Notes in Computer Science, vol. 5202, pp. 582–587. Springer (2008)
16. Li, C.M., Manyà, F., Planes, J.: Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In: Proc. AAAI. pp. 86–91. AAAI Press (2006)
17. Li, C., Manyà, F.: MaxSAT, hard and soft constraints. In: Handbook of Satisfiability, pp. 613–631. IOS Press (2009)
18. Lin, H., Su, K.: Exploiting inference rules to compute lower bounds for MAX-SAT solving. In: Proc. IJCAI. pp. 2334–2339 (2007)
19. Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in Max-SAT solving. In: Proc. AAAI. pp. 351–356. AAAI Press (2008)
20. Moreno-Centeno, E., Karp, R.M.: The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. Operations Research 61(2), 453–468 (2013)
21. Morgado, A., Heras, F., Marques-Silva, J.: Improvements to core-guided binary search for MaxSAT. In: Proc. SAT. Lecture Notes in Computer Science, vol. 7317, pp. 284–297. Springer (2012)
22. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley-Interscience (1999)
23. Saikko, P., Berg, J., Järvisalo, M.: LMHS: A SAT-IP hybrid MaxSAT solver. In: Proc. SAT. Lecture Notes in Computer Science, vol. 9710, pp. 539–546. Springer (2016)

24. Saikko, P.: Re-implementing and Extending a Hybrid SAT-IP Approach to Maximum Satisfiability. Master's thesis, University of Helsinki (2015), http://hdl.handle.net/10138/159186
25. Saikko, P., Wallner, J.P., Järvisalo, M.: Implicit hitting set algorithms for reasoning beyond NP. In: Proc. KR. pp. 104–113. AAAI Press (2016)
26. Thorsteinsson, E.S., Ottosson, G.: Linear relaxations and reduced-cost based propagation of continuous variable subscripts. Annals of Operations Research 115(1-4), 15–29 (2002)
27. Wolsey, L.A.: Integer Programming. John Wiley (1998)
28. Yunes, T.H., Aron, I.D., Hooker, J.N.: An integrated solver for optimization problems. Operations Research 58(2), 342–356 (2010)