CCAnr: A Configuration Checking Based Local Search Solver for Non-random Satisfiability

Shaowei Cai[1(✉)], Chuan Luo[2], and Kaile Su[3,4]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
shaoweicai.cs@gmail.com
[2] School of EECS, Peking University, Beijing, China
chuanluosaber@gmail.com
[3] Department of Computer Science, Jinan University, Guangzhou, China
[4] IIIS, Griffith University, Nathan, Australia
k.su@griffith.edu.au

Abstract. This paper presents a stochastic local search (SLS) solver for SAT named CCAnr, which is based on the configuration checking strategy and has good performance on non-random SAT instances. CCAnr switches between two modes: it flips a variable according to the CCA (configuration checking with aspiration) heuristic if any; otherwise, it flips a variable in a random unsatisfied clause (which we refer to as the focused local search mode). The main novelty of CCAnr lies on the greedy heuristic in the focused local search mode, which contributes significantly to its good performance on structured instances. Previous two-mode SLS algorithms usually utilize diversifying heuristics such as *age* or randomized strategies to pick a variable from the unsatisfied clause. Our experiments on combinatorial and application benchmarks from SAT Competition 2014 show that CCAnr has better performance than other state-of-the-art SLS solvers on structured instances, and its performance can be further improved by using a preprocessor CP3. Our results suggest that a greedy heuristic in the focused local search mode might be helpful to improve SLS solvers for solving structured SAT instances.

1 Introduction

The Satisfiability problem (SAT) is a prototypical NP-complete problem of importance in both theory and applications. Two popular approaches for solving SAT are conflict driven clause learning (CDCL) and stochastic local search (SLS). SLS algorithms for SAT perform a local search in the space of truth assignments by starting with a complete assignment, and then repeatedly flipping the truth value of a variable until a satisfying assignment has been found or some limits (usually the time limit) have been reached. The function for choosing the variable to be flipped is usually denoted as *pickVar*.

SLS algorithms for SAT mainly fall into two types: focused local search (FLS) and two-mode SLS. Focused local search (as called in [15,17]) always picks the flip variable from an unsatisfied clause [13,19]; two-mode SLS [1,5,10,16,18]

M. Heule and S. Weaver (Eds.): SAT 2015, LNCS 9340, pp. 1–8, 2015.
DOI: 10.1007/978-3-319-24318-4_1

switches between "global local search" (where the flip variable is chosen from a candidate set filtered from the set of all variables) and focused local search, usually depending on whether a local optimum is reached. Also, there is a significant line of research concerns about weighting techniques [8,14,20,21], which are usually utilized in two-mode SLS algorithms.

SLS is well known as the most effective approach for solving random satisfiable instances, and SLS solvers are often evaluated on random k-SAT benchmarks. For structured instances, SLS solvers have been considered not effective as complete solvers (particularly CDCL ones) for a long time. Nevertheless, recent progress shows promising results of SLS solvers, particularly two-mode SLS solvers, on crafted satisfiable instances. Modern two-mode SLS solvers are competitive and complementary with complete solvers in solving crafted instances. For example, in the crafted SAT track of SAT Competition 2009, the SLS solver Sattime solved 109 instances while the best CDCL solved 93 instances [11]; in SAT Competition 2013, CCAnr solved 21 crafted SAT instances that the best complete solver glucose (v2.3) failed to solve, while glucose solved 53 instances that CCAnr failed to solve in the same track (Hard-combinatorial SAT track).[1] Indeed, the top three solvers in the Hard-combinatorial SAT track of SAT Competition 2014, namely Sparrow2riss [2], CCAnr+glucose [3] and SGseq [9] are all hybrid solvers combining an SLS solver and a complete solver.

In this paper, we present the CCAnr solver, which is a two-mode SLS solver designed for solving structured instances. Existing two-mode SLS solvers for SAT, including state-of-the-art ones such as Sparrow [1], Sattime [11] and CCASat [5], employ greedy heuristics in the global local search mode with the aim of decreasing the number of unsatisfied clauses, and employ diversifying heuristics in the focused local search mode with the aim of better exploring the search space and avoiding local optima. However, CCAnr employs a greedy heuristic (i.e., picking the one with the greatest *score*) in the focused local search mode, which significantly contributes to its good performance on structured instances. Our experiments comparing different versions of CCAnr with various heuristics in the focused local search mode show that, for solving structured instances, it could be helpful to incorporate greedy heuristics in the focused local search mode.

The good performance of CCAnr as compared to other SLS solvers is also confirmed by the results in the SAT Competitions 2013 and 2014. In the Hard-combinatorial SAT track of SAT Competition 2013, CCAnr solves more instances than other SLS solvers except Sparrow+CP3, which solves only 2 more instances than CCAnr. In SAT Competition 2014, CCAnr+glucose is ranked second in the same track, solving only 1 less instance than Sparrow2riss. However, we note that both Sparrow+CP3 and Sparrow2riss utilize a powerful preprocessor CP3 [12] while CCAnr only performs unit propagation before local search. In this sense, CCAnr can be considered as the best pure SLS solver in this track. In this paper, we also combine CCAnr with the preprocessor CP3, and the resulting solver solves more instances in those benchmarks, yielding further improvement over other SLS solvers.

[1] http://satcompetition.org/edacc/SATCompetition2013/experiment/20/

2 Solver Description

CCAnr is built on top of the Swcca solver [4], and keeps the main technique there, namely the configuration checking (CC) strategy with an aspiration mechanism. In this section, we first introduce the CCA heuristic and the Swcca algorithm, and then describe the CCAnr algorithm by specifying the difference from Swcca. CCAnr is open-source and the code is available for download at http://lcs.ios.ac.cn/~caisw/SAT.html. Readers interested in the implementation may like to refer to the codes for more details.

2.1 The CCA Heuristic and Swcca

Originally proposed in [6], the configuration checking (CC) strategy aims at avoiding cycling (i.e., revisiting the already visited assignments too early) in local search. CC has proved effective in local search for SAT and has been widely used in recent SLS solvers. In the context of SAT, the idea of CC is to forbid flipping a variable if its *configuration* has not been changed after its last flip, where the *configuration* of a variable typically refers to truth values of all its neighbouring variables [4,5]. The CCA heuristic combines the CC strategy with an aspiration mechanism, which allows to flip a variable forbidden by CC if it has a significant *score*, recalling that a variable's *score* is the change on the number (or the total weight) of satisfied clause produced by its flip.

It is easy to see from the above discussions that, there are two kinds of variables of importance in the CCA heuristic, which are defined as follows.

- A *configuration changed decreasing* (CCD) variable is a decreasing variable (with positive *score*) whose configuration has been changed (i.e., at least one of its neighbouring variables has been flipped) since its last flip.
- A *significant decreasing* (SD) variable is a variable with $score(x) > g$, where g is a positive sufficient large integer, and in this work g is set to the averaged clause weight (over all clauses) $\overline{w}$.

A two-mode SLS algorithms based on the CCA heuristic works as follows. In the global local search mode, it prefers to pick a CCD variable with the greatest *score* to flip. If there are no CCD variables, an SD variable with the greatest *score* is selected there is one. If there are neither CCD variables nor SD variables, the algorithm updates the clause weights and switches to the the focused local search mode, where a variable in a random unsatisfied clause is picked to flip.

Based on the CCA heuristic, we have developed the Swcca algorithm [4], which has good performance on random 3-SAT instances and crafted instances. The pickVar function of Swcca is shown in Algorithm 1.

2.2 The CCAnr Algorithm

CCAnr is an improved version of Swcca for structured instances. Starting with a randomly generated complete assignment, CCAnr iteratively flips a variable until a satisfying assignment has been found or the given time limit has been reached. There are two differences between CCAnr and Swcca algorithms.

Algorithm 1. *pickVar* function in Swcca

1 **if** *CCD variables exist* **then return** a CCD variable with the greatest *score*, breaking ties in favor of the oldest one;
2 **if** *SD variables exist* **then return** an SD variable with the greatest *score*, breaking ties in favor of the oldest one;
3 increases clause weights of all unsatisfied clauses by one;
4 **if** $\overline{w} > \gamma$ **then** for each clause c_i, $w(c_i) := \lfloor \rho \cdot w(c_i) \rfloor + \lfloor (1-\rho)\overline{w} \rfloor$;
5 pick a random unsatisfied clause c;
6 **return** the oldest variable in c;

1. the smoothing formula (line 4 in Algorithm 1) is generalized as $w(c_i) := \lfloor \rho \cdot w(c_i) \rfloor + \lfloor q \cdot \overline{w} \rfloor$.
2. in the focused local search mode, CCAnr picks the variable with the greatest *score* from an unsatisfied clause, breaking ties by favoring the oldest one (replace line 6 in Algorithm 1).

As will be shown in the experiment parts, the first modification, which indeed is just about parameter setting, has little impact on the performance of CCAnr. The second modification (i.e., the greedy heuristic in focused local search mode) makes the essential contributions to the good performance of CCAnr, and renders CCAnr much more effective than the original solver Swcca on structured instances, and also outperforms (although sometimes slightly) other state-of-the-art SLS solvers for structured instances.

2.3 Implementation Details

CCAnr is implemented in C++, and compiled by g++ with the 'O3' optimization option. There are three parameters in CCAnr: the threshold parameter γ, and two factor parameters ρ and q, all of which belong to the clause weighting scheme. The parameters are set as follows: $\gamma = 300$; $\rho = 0.3$; q is set to 0 if $r \leq 15$, and 0.7 otherwise (r is the clause-to-variable ratio of the instance). This setting is adopted in CCAnr in SAT Competition 2013 and CCAnr+glucose in SAT Competition 2014. After the competition, we found that actually setting q to 0.7 (which equals $1 - \rho$, as the same setting in Swcca) for all instances has very close performance.

3 Experimental Results

According to results of SAT Competitions 2013 and 2014, CPSparrow, CCAnr and Sattime are the currently the best three SLS solvers for structured SAT instances. However, CCAnr itself did not participate in SAT Competition 2014; instead, the hybrid solver CCAnr+glucose did. We carried out experiments to compare CCAnr with CPSparrow (version sc14) and Sattime2014r as well as Sattime2013 (the version used in SGseq [9] in Hard-combinatorial track of SAT Competition 2014).

We also develop a solver called CCAnr+CP3 which employs the CP3 preprocessor before calling CCAnr (as CPSparrow does), and include it in the experiments. Sattime already has a sophisticated preprocessing procedure in it.

Our experiments are conducted on Hard-combinatorial SAT benchmark (150 instances) which is also known as the crafted benchmark, and application SAT benchmark (150 instances) in SAT Competition 2014. The experiments are carried out on a machine under GNU/Linux, using 2 cores of Intel Core i7 2.4 GHz and 7.8 GByte RAM. Each solver was executed one time on each instance, as in competitions.

The results are summarized in Table 1, where the first row presents the results on the Hard-combinatorial SAT benchmark, and the second row presents the results on the application SAT benchmark. CCAnr+CP3 gives better performance than other solvers on both benchmarks. We also observe that even without CP3, CCAnr solves more instances than the two competitors CPSparrow and Sattime2014r in this experiment, although indeed the performance of CCAnr and CPSparrow is indistinguishable. In particular, CCAnr solves 13 instances in the Hard-combinatorial SAT benchmark which were not solved by CDCL solvers in SAT Competition 2014.

Table 1. Comparative results on the hard-combinatorial SAT and the application SAT benchmarks from SAT Competition 2014.

Benchmark	CCAnr+CP3		CCAnr		CPSparrow		Sattime2014r		Sattime2013	
	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time
SC14_HC_SAT	**60**	**30115**	56	31427	55	31727	46	34784	43	35864
SC14_APP_SAT	**35**	**38440**	29	40449	28	40773	25	41822	23	42539

To demonstrate the importance of the greedy heuristic in the focused random mode in CCAnr, we also compare it with three alternatives. The alternatives are modified from from CCAnr as follows:

- CCAnr_fq (short for CCAnr with fixed q) is the CCAnr solver with a fixed setting of parameter q in the smoothing formula: $q = 0.7$ (i.e., $q = 1 - \rho$) for all instances (in this case, the clause weighting is the same with Swcca).
- CCAnr_rand picks a random variable in the selected unsatisfied clause in the focused local search mode.
- CCAnr_age picks the oldest variable in the selected unsatisfied clause in the focused local search mode. CCAnr_age corresponds to the Swcca algorithm, but use the parameter setting of SWT scheme in CCAnr, so that we can see the performance improvement due to the greedy heuristic.

The comparative results of CCAnr and its alternatives are reported in Table 2. The performance of CCAnr_fq is quite close to that of CCAnr, which indicates the conditional setting rule for parameter q in the smoothing formula (used in

Table 2. Comparative results of CCAnr and its alternatives on the hard-combinatorial SAT and the application SAT benchmarks from SAT Competition 2014.

Benchmark	CCAnr		CCAnr_fq		CCAnr_rand		CCAnr_age	
	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time	#solv.	par10 time
SC14_HC_SAT	**56**	**31427**	54	32113	47	34514	41	36484
SC14_APP_SAT	**29**	**40449**	28	40755	25	41753	22	42709

CCAnr version in competitions) is not a main factor of the good performance of CCAnr. The performance of CCAnr is considerably better than that of CCAnr_age and CCAnr_rand. Noting that the only difference between CCAnr and these two alternative solvers is that, CCAnr uses a greedy heuristic in the focused local search mode while CCAnr_age and CCAnr_rand employ diversifying heuristics. This indicates that using greedy heuristics rather than diversifying ones in the focused local search mode might be helpful to improve SLS-based SAT solvers on structured benchmarks.

CCAnr has also been discovered to be very competitive with other SLS-based SAT solvers on solving Satisfiability Modulo Theories (SMT) instances, although it can not compete with the SLS-based SMT solver called BV-SLS which works on the theory level representation [7]. The experiments in [7] were conducted with two benchmarks of bit-vector formulas namely `QF_BV` and `SAGE2`. The `QF_BV` benchmark can be found in the SMT-LIB and is also part of the SMT Competition. The `SAGE2` benchmark consists of problems generated as part of the SAGE project at Microsoft, describing some testcases for automated whitebox fuzz testing. The number of solved instances are reported in Table 3, which is taken from [7]. Z3 is a state-of-the-art SMT solver. As well as Z3 and BV-SLS, the experiments include state-of-the-art SLS-based SAT solvers or those have good performance on certain types of structured SAT instances.

Table 3. Number of solved instances in bit-vector SMT benchmarks

	QF_BV (7498 instances)	SAGE2 (8017 instances)
CCAnr	**5409**	64
CCASat	4461	8
probSAT	3816	10
Sparrow	3806	12
VW2	2954	4
PAWS	3331	**143**
YalSAT	3756	142
Z3 (Default)	7173	5821
BV-SLS	6172	3719

Acknowledgments. This work is supported by China National 973 Program 2014CB340301, National Natural Science Foundation of China 61370156 and 61472369. We would like to thank the anonymous referees for their helpful comments.

References

1. Balint, A., Fröhlich, A.: Improving stochastic local search for sat with a new probability distribution. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 10–15. Springer, Heidelberg (2010)
2. Balint, A., Manthey, N.: SparrowToRiss. In: Proc. of SAT Competition 2014: Solver and Benchmark Descriptions, p. 77 (2014)
3. Cai, S., Luo, C., Su, K.: CCAnr+glucose in SAT competition 2014. In: Proc. of SAT Competition 2014: Solver and Benchmark Descriptions, p. 17 (2014)
4. Cai, S., Su, K.: Configuration checking with aspiration in local search for SAT. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2012, pp. 334–340 (2012)
5. Cai, S., Su, K.: Local search for Boolean Satisfiability with configuration checking and subscore. Artif. Intell. **204**, 75–98 (2013)
6. Cai, S., Su, K., Sattar, A.: Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif. Intell. **175**(9–10), 1672–1696 (2011)
7. Fröhlich, A., Biere, A., Wintersteiger, C.M., Hamadi, Y.: Stochastic local search for satisfiability modulo theories. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015, pp. 1136–1143 (2015)
8. Hutter, F., Tompkins, D.A.D., H. Hoos, H.: Scaling and probabilistic smoothing: efficient dynamic local search for SAT. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 233–248. Springer, Heidelberg (2002)
9. Li, C.M., Habet, D.: Description of RSeq2014. In: Proc. of SAT Competition 2014: Solver and Benchmark Descriptions, p. 72 (2014)
10. Li, C.-M., Huang, W.Q.: Diversification and determinism in local search for satisfiability. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 158–172. Springer, Heidelberg (2005)
11. Li, C.M., Li, Y.: Satisfying versus falsifying in local search for satisfiability. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 477–478. Springer, Heidelberg (2012)
12. Manthey, N.: Coprocessor 2.0 – a flexible CNF simplifier. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 436–441. Springer, Heidelberg (2012)
13. McAllester, D.A., Selman, B., Kautz, H.A.: Evidence for invariants in local search. In: Proceedings of the 14th National Conference on Artificial Intelligence, AAAI 1997, pp. 321–326 (1997)
14. Morris, P.: The breakout method for escaping from local minima. In: Proceedings of the 11th National Conference on Artificial Intelligence, AAAI 1993, pp. 40–45 (1993)
15. Papadimitriou, C.H.: On selecting a satisfying truth assignment. In: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, FOCS 1991, pp. 163–169 (1991)
16. Pham, D.N., Gretton, C.: gNovelty+. In: Solver Description of SAT Competition 2007 (2007)

17. Seitz, S., Alava, M., Orponen, P.: Focused local search for random 3-satisfiability. J. Stat. Mech. (2005). P06006
18. Selman, B., Kautz, H.A.: Domain-independent extensions to gsat: solving large structured satisfiability problems. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence, IJCAI 1993, pp. 290–295 (1993)
19. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the 12th National Conference on Artificial Intelligence, AAAI 1994, pp. 337–343 (1994)
20. Thornton, J., Pham, D.N., Bain, S., Jr., V.F.: Additive versus multiplicative clause weighting for SAT. In: Proceedings of the 19th National Conference on Artificial Intelligence, AAAI 2004, pp. 191–196 (2004)
21. Wu, Z., Wah, B.W.: An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In: Proceedings of the 17th National Conference on Artificial Intelligence, AAAI 2000, pp. 310–315 (2000)