Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)
Sister Conferences Best Papers Track

Deep Cooperation of CDCL and Local Search for SAT (Extended Abstract)*

Shaowei Cai , Xindi Zhang
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
School of Computer Science and Technology, University of Chinese Academy of Sciences, China
{caisw,zhangxd}@ios.ac.cn

Abstract

Modern SAT solvers are based on a paradigm named conflict driven clause learning (CDCL), while local search is an important alternative. Although there have been attempts combining these two methods, this work proposes deeper cooperation techniques. First, we relax the CDCL framework by extending *promising* branches to complete assignments and calling a local search solver to search for a model nearby. More importantly, the local search assignments and the *conflict frequency* of variables in local search are exploited in the phase selection and branching heuristics of CDCL. We use our techniques to improve three typical CDCL solvers (glucose, MapleLCMDistChronoBT and Kissat). Experiments on benchmarks from the Main tracks of SAT Competitions 2017-2020 and a real world benchmark of spectrum allocation show that the techniques bring significant improvements, particularly on satisfiable instances. A resulting solver won the Main Track SAT category in SAT Competition 2020 and also performs very well on the spectrum allocation benchmark. As far as we know, this is the first work that meets the standard of the Selman's 7th challenge "Demonstrate the successful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches." on standard application benchmarks.

1 Introduction

The Satisfiability problem (SAT) asks to determine whether a given propositional formula is satisfiable or not. In the SAT problem, propositional formulas are usually presented in Conjunctive Normal Form (CNF), i.e., $F = \bigwedge_i \vee_j \ell_{ij}$. A growing number of problem domains are successfully tackled by SAT solvers, including the electronic design automation (EDA) industry [Silva and Sakallah, 2000], mathematical theorem proving [Heule *et al.*, 2016], AI planning [Kautz and Selman, 1992], spectrum allocation [Newman *et al.*, 2018], among others. Also, SAT solvers are often used as a core component of more complex tools such as solvers for Satisfiability Modulo Theory (SMT), which are indispensable for program analysis and software verification.

Conflict driven clause learning (CDCL) is the most popular approach to solving SAT. Since their inception in the mid-90s, CDCL-based SAT solvers have been applied, in many cases with remarkable success, to a number of practical applications. An alternative is local search, which is a main incomplete method biased towards the satisfiable side, and can be more successful on random and hard combinatorial instances [Li and Li, 2012; Cai *et al.*, 2015; Biere *et al.*, 2020]. There have been attempts combining CDCL and local search solvers. However, in previous hybrid solvers, CDCL and local search solvers usually see each other as a black box and the hybrid solver invokes the respective solver according to different situations [Mazure *et al.*, 1998; Habet *et al.*, 2002; Letombe and Marques-Silva, 2008; Balint *et al.*, 2009; Audemard *et al.*, 2010]. This work is devoted to deeper cooperation of CDCL and local search for SAT, where CDCL is the main solver and local search is used as an aiding tool.

We propose three ideas to use local search to help CDCL in different ways. The first idea is a method for plugging a local search solver into a CDCL solver, by extending promising branches to complete assignments (even meeting conflicts) and then calling local search to search for a model. If the local search cannot find a model within a given time limit, the CDCL search process continues as normal from the node where the algorithm enters the non-backtracking phase. Secondly, We propose a rephasing technique as a heuristic to select the value of the branching variable, which mainly relies on the assignments produced by the integrated local search solver. Lastly, We use the variables' *conflict frequency*, i.e., the frequency appearing in unsatisfied clauses during local search, to enhance the branching heuristic in CDCL.

We apply our techniques to three state-of-the-art CDCL solvers, including glucose [Audemard and Simon, 2009], and the winner of the Main track of SAT Competition 2019 and 2020 namely MapleLCMDistChronoBT-DL [Kochemazov *et al.*, 2019] and Kissat_sat [Biere *et al.*, 2020]. The results show that our techniques allow them to solve a remarkable number of additional instances in the benchmarks of SAT Competi-

*This is an extended abstract of a paper that appeared/won best award at conference SAT 2021.

tion 2017-2020. The integration of our techniques allow the three CDCL solvers to solve 62, 67 and 10 more instances in the benchmark of SAT Competition 2020. Besides, the improved version of the three CDCL solvers shows better results on a benchmark arising from a spectrum repacking problem.

The relaxed CDCL and local search rephasing techniques are very helpful for satisfiable instances, with slight degradation on unsatisfiable instances (usually solving 2 or 3 fewer unsatisfiable instances). The conflict frequency technique can be positive to satisfiable and also saves back a few unsatisfiable instances. Overall, these techniques significantly improve the performance of the CDCL solvers, leading to a remarkable increase on the number of solved instances.

2 Preliminaries

2.1 Preliminary Definitions and Notations

A mapping $\alpha : V \rightarrow \{0, 1\}$ is called an $assignment$. If α maps all variables to a Boolean value, it is a $complete\ assignment$; otherwise, it is a $partial\ assignment$. The size of an assignment α, denoted as $|\alpha|$, is the number of assigned variables in it. The value of a variable x under an assignment α is denoted as $\alpha[x]$. An assignment α satisfies a clause iff at least one literal evaluates to true under α, and satisfies a CNF formula iff it satisfies all its clauses.

A clause that contains exactly one single (unassigned) literal is called a *unit clause*. A key procedure in CDCL solvers is *unit propagation*. For a unit clause, the variable is assigned to satisfy this unit clause, and then the formula is simplified according to this setting. The iterative execution of such steps until no more unit clause remains is called *unit propagation*.

2.2 CDCL Solvers

A CDCL solver performs a backtracking search (can be non-chronological) in the space of partial assignments. Each node of the tree corresponds to a partial assignment, and the out edges represent the two *branching value* (aka. *phase*) . Each leaf corresponds to a complete assignment. CDCL solvers can prune a large part of the tree thanks to reasoning techniques. A *branch* is a path from the root to an inner node. In this work, we use $\alpha\_longeste$ to denote the largest conflict-free assignment that has been encountered by the solver so far.

Algorithm 1shows the standard procedure of a CDCL solver, where α is the current assignment, dl is the current decision level and bl denotes the backtrack level. Arguments to the functions are assumed to be passed by reference, and thus F and α are supposed to be modified during the search. The functions are explained here. *PickBranchVar* consists of selecting a variable to assign and the respective phase. *Unit-Propagation* performs unit propagation on the formula, and if a conflict is identified, then a conflict indication is returned. Once a conflict is derived, the reasons are analyzed and a clause is learnt (known as *learnt conflict clause*) and then added to the clause database. Finally, *BackTrack* backtracks to the decision level computed by ConflictAnalysis.

Two popular branching heuristics that are used to pick the branching variable include Variable State Independent Decaying Sum (VSIDS) [Moskewicz *et al.*, 2001] and Learning Rate Branching (LRB) [Liang *et al.*, 2016]. In VSIDS [Moskewicz *et al.*, 2001], each variable has an *activity*, and VSIDS picks the variable with the maximum activity. Every time a variable occurs in a conflict clause, its activity is increased (aka bumped). After the conflict, the activity of all the variables in the system are multiplied by a constant less than 1, thus decaying the activity of variables over time. LRB frames branching as an optimization problem that picks a variable to maximize a metric called *learning rate*. The learning rate of a variable x at interval I is defined as $\frac{P(x,I)}{L(I)}$, where I is the interval of time between the assignment of x until x transitions back to being unassigned, $P(x, I)$ is the number of learnt clauses x participates in interval I, and $L(I)$ is the number of learnt clauses generated in interval I.

Algorithm 1: Typical CDCL algorithm: CDCL(F, α)

```
dl ← 0;                    //decision level
if UnitPropagate(F,α)==CONFLICT then return UNSAT;
while ∃ unassigned variables do
    (x, v) ← PickBranchVar(F, α);
    dl ← dl + 1;
    α ← α ∪ {(x, v)};
    if UnitPropagate(F,α)==CONFLICT then
        bl ← ConflictAnalysis(F, α);
        if bl < 0 then
            return UNSAT;
        else
            BackTrack(F, α, bl);
            dl ← bl;
return SAT;
```

2.3 Local Search Solvers

For local search algorithms, we need to define the search space and a neighborhood relation. In the context of SAT, the search space is the set of complete assignments which can be characterized as the set of strings $\{0, 1\}^n$, where n is the number of variables in the formula. For SAT, the seemingly most natural neighborhood N maps candidate solutions to their set of Hamming neighbors, i.e., candidate solutions that differ in exactly one variable. A local search algorithm starts from a position of search space and then moves to one neighbor of the current position in each step, trying to find a position which represents a satisfying assignment.

2.4 Experiment Preliminaries

Base Solvers. We choose three state of the art CDCL solvers as the base solvers for our studies, including glucose (v4.2.1) [Audemard and Simon, 2009], MapleLCMDistChronoBT-DL (v2.1) [Kochemazov *et al.*, 2019], and Kissat_sat (2414b6d) [Biere *et al.*, 2020]. Glucose is a milestone of modern CDCL solvers and has won several gold medals in SAT Competitions. MapleLCMDistChronoBT-DL won the SAT Race 2019 and Kissat_sat won the Main Track of SAT Competition 2020.

We choose CCAnr [Cai *et al.*, 2015] as the local search solver to integrate into the CDCL solvers glucose and MapleLCMDistChronoBT-DL, while Kissat_sat itself already includes a local search solver YalSAT [Biere, 2014]. CCAnr has shown competitive results on various structured instances from SAT competitions and applications.

Benchmarks. Besides the benchmarks of the latest four SAT Competitions/Race (2017-2020), we include a benchmark suite consisted of 10000 instances from the spectrum repacking [Newman *et al.*, 2018].

Experiment Setup. Experiments were conducted on a cluster of computers with Intel Xeon Platinum 8153 @2.00GHz CPUs and 1024G RAM under the operating system CentOS 7.7.1908. For each instance, each solver was performed one run, with a cutoff time of 5000 CPU seconds. For each solver for each benchmark, we report the number of solved SAT/UNSAT instances and total solved instances, denoted as '#SAT', '#UNSAT' and '#Solved', and penalized run time 'PAR2' (as in SAT Competitions), where the run time of a failed run is penalized as twice the cutoff time.

3 Exploring Promising Branches

In CDCL, some promising branches that are close to a satisfying assignment are also pruned without any exploitation. This would make CDCL solvers miss some opportunities of finding a solution. The exploration on promising branches may improve CDCL solvers on satisfiable formulas, and a natural way to do so is to employ local search at such branches.

We present a method to explore promising branches in CDCL solvers, which can improve the ability to find solutions while keeping the completeness of the solvers. For this method, we need to identify which branches (i.e., partial assignments) deserve exploration. We propose two conditions below, and any assignment α satisfying at least one of them is considered as promising and will be explored:

- $\frac{|\alpha|}{|V|} > p$ and there is no conflict under α, where p is a parameter and is set to 0.4 according to preliminary experiments on a random sample of instances from recent SCs.
- $\frac{|\alpha|}{|\alpha\_longest|} > q$ and there is no conflict under α, where q is set to 0.9 similarly.

In order to ensure that the search space of adjacent local search calls is sufficiently different, we disallow local search for a certain number of k restarts, where k is set to 500 for glucose, and 400 for Maple.

During the search of CDCL, whenever reaching a promising branch, the algorithm enters a non-backtracking mode, which uses unit propagation and heuristics in CDCL to assign the remaining variables without backtracking, even an empty clause is detected. At the end, this leads to a complete assignment β, which is fed to a local search solver to search for a model nearby. If the local search fails to find a model within a certain time budget (reaching 5×10^7 memory accesses), then the algorithm goes back to the normal CDCL search from the node where it was interrupted.

https://www.cs.ubc.ca/labs/beta/www-projects/SATFC/cacm_cnfs.tar.gz

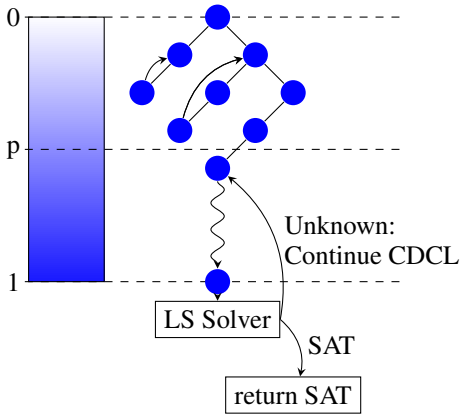

Figure 1: Overall procedure of relaxed CDCL

Phase	$\alpha\_longest\_LS$	α_latest_LS	α_best_LS	keep it
Prob.	20%	65%	5%	10%

Table 1: Rephasing probability in our rephasing mechanism

4 Rephasing with Local Search

Now, we propose an rephasing heuristic to import back an improved assignments obtained by the local search process, which is referred to as local-search rephasing (LS rephasing for short).

After each time the CDCL solver is restarted, the technique overwrites the saved phases of all variables with assignments produced by local search. When we say the assignment of a local search procedure, we refer to the best assignment (with fewest unsatisfied clauses) in this procedure.

For our phase resetting technique, we consider the following assignments, all of which come from the assignments of the local search procedures.

- $\alpha\_longest\_LS$: the assignment of the local search procedure in which the initial solution is extended based on $\alpha\_longest$.
- α_latest_LS: the assignment of the latest local search procedure.
- α_best_LS: the best assignment (with the fewest unsatisfied clauses) among all local search procedures so far.

It is easy to see that $\alpha\_longest\_LS$ and α_best_LS serve for the aim to maximize the depth of the branch, while α_latest_LS adds diversification, as different local search procedures start with initial assignments built upon different branches. Whenever the CDCL is restarted, we reset all variables with one complete assignment which is selected according to the rephasing probabilities given in Table 1. Such changes are always allowed, because they do not impact the underlying CDCL calculus, its correctness, nor termination.

5 Branching with Conflict Frequency

CDCL is a powerful framework owing largely to the utilization of the conflict information, and branching strategies aim to promote conflicts. Intuitively, both VSIDS and LRB prefer to pick variables with higher frequencies occurring in conflicts, with an emphasis in a recent period. We propose to enhance the branching strategy by utilizing the conflict frequency of variables in the *latest* local search procedure.

In a local search process, for a variable x, its conflict frequency, $ls\_confl\_freq(x)$, is defined as the number of steps in which it appears in at least one unsatisfied clause divided by the total number of steps of the local search process.

For each variable x, we multiply $ls\_confl\_freq(x)$ with a constant integer (100 in this work), and the resulting number is denoted as $ls\_conflict\_num(x)$. We use $ls\_conflict\_num(x)$ to enhance the branching strategies as follows. Note that $ls\_conflict\_num(x)$ is calculated according to the **latest** local search procedure. After each restart of the CDCL solver, $ls\_conflict\_num(x)$ is used to modify the activity score for VSIDS and learning rate for LRB.

- VSIDS: for each variable x, its activity score is increased by $ls\_conflict\_num(x)$.
- LRB: for each variable x, the number of learnt clause during its period I is increased by $ls\_conflict\_num(x)$. That is, both $P(x, I)$ and $L(I)$ are increased by $ls\_conflict\_num(x)$.

6 Experiment Results

For glucose and MapleLCMDistChronoBT-DL-v2.1, we implement all the three techniques, including relaxed CDCL (denoted as `rx`), LS rephasing (denoted as `rp`) and conflict frequency (denoted as `cf`). For Kissat_sat, we only implement the `cf` technique. All the source codes and experiment statistics can be downloaded online.

Evaluations on Benchmarks of SAT Competitions (Table 2). By implementing all the three techniques, very large improvements are obtained for glucose and MapleLCMDistChronoBT-DL-v2.1 for all the benchmarks. Particularly, glucose+rx+rp+cf solves 62 additional instances than the original solver, and Maple-DL+rx+rp+cf solves 67 additional instances than its original solver for the SC2020 benchmark (which has 400 instances). Maple-DL+rx+rp+cf is a simplified and optimized version of our solver Relaxed_LCMDCBDL_newTech which won the gold medal of Main Track SAT category and the silver medal of the Main Track ALL category in SC 2020.

Evaluations on Spectrum Repacking (Table3). For each CDCL solver, the improved version with our techniques has better performance than the base solver. Maple+rx+rp+cf solves the most instances (8759+218=8977), significantly better than all the other solvers.

Further Analyses on the Cooperation. We can see that the local search solver returns a solution for some instances. A natural question is: *Whether the improvements come mainly from the complementation of CDCL and local search solvers that they solve different instances?* To answer this question, we compare the instances solved by the hybrid solvers with those by the base CDCL solver and the local search solver. We observe that, there is a large number of instances that both CDCL and local search solvers fail to solve but can be solved by the hybrid solvers. For glucose, this number reaches 29, 36, 26 and 37 for the four benchmarks respectively, while for MapleLCMDIstChronoBT-DL, this number reaches 16, 18, 15 and 36 respectively. This indicates the cooperation techniques have essential contributions to the good performance of the hybrid solvers.

The number of local search calls is usually from 10 to 25 for these benchmarks. The portion of the time spent on local search is between 6% and 20% for satisfiable instances, and usually less than 7% for UNSAT instances.

https://github.com/caiswgroup/relaxed-sat

solver	#SAT	#UNSAT	#Solved	PAR2	#SAT	#UNSAT	#Solved	PAR2
	SC2017(351)				SC2018(400)			
glucose_4.2.1	83	101	184	5220	95	95	190	5745
glucose+rx	88	95	183	5237	113	95	208	5283
glucose+rx+rp	112	94	206	4618	141	87	228	4698
glucose+rx+rp+cf	110	94	204	4668	150	91	241	4438
Maple-v2.1	101	113	214	4531	133	102	235	4533
MapleL+rx	101	112	213	4520	149	101	250	4148
Maple+rx+rp	111	103	214	4447	158	93	251	4147
Maple+rx+rp+cf	116	107	223	4139	162	97	259	3927
Kissat_sat	115	114	229	3943	167	98	265	3786
Kissat_sat+cf	113	113	226	4001	178	104	282	3409
CCAnr	13	N/A	13	9629	56	N/A	56	8622
	SC2019(400)				SC2020(400)			
glucose_4.2.1	118	86	204	5437	68	91	159	6494
glucose+rx	120	84	204	5443	93	88	181	6018.1
glucose+rx+rp	134	85	219	5096	130	85	215	5123
glucose+rx+rp+cf	140	85	225	4923.6	134	87	221	4977.9
Maple-v2.1	143	97	240	4601	86	104	190	5835
Maple+rx	146	93	239	4602	121	105	226	4977
Maple+rx+rp	155	94	249	4416	142	99	241	4589
Maple+rx+rp+cf	154	95	249	4377	151	106	257	4171
Kissat_sat	159	88	247	4293	146	114	260	4048
Kissat_sat+cf	162	90	252	4211	157	113	270	3896
CCAnr	13	N/A	13	9678	45	N/A	45	8978

Table 2: Experiment results on benchmarks of SAT Competitions 2017-2020

	glucose-4.2.1		Maple		kissat-sat		CCAnr
	.	+	.	+	.	+cf	.
#SAT	7330	8075	8084	8759	8192	8214	7853
#UNSAT	187	197	215	218	207	211	0
#Solved	7517	8272	8299	8977	8399	8425	7853
PAR2	2555.8	1850.5	1867.1	1243.6	1760.5	1734.6	2215.3

Table 3: Experiment results on FCC. The default version is marked as "`.`", whereas "+" stands for `+rx+rp+cf`.

7 Conclusions

This work took a large step towards deep cooperation of CDCL and local search. We proposed three techniques for using local search to improve CDCL solvers.

This is the first time that the combination of stochastic search and systematic search techniques leads to essential improvements over the state of the art of both approaches on application benchmarks, thus answering Challenge 7 of the Ten Challenges in Propositional Reasoning and Search [Selman *et al.*, 1997].

Acknowledgements

This work is supported by NSFC Grant 62122078, Beijing Academy of Artificial Intelligence (BAAI).

References

[Audemard and Simon, 2009] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of IJCAI 2009*, pages 399–404, 2009.

[Audemard *et al.*, 2010] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais. Boosting local search thanks to cdcl. In *Proceedings of LPAR 2010*, pages 474–488, 2010.

[Balint *et al.*, 2009] Adrian Balint, Michael Henn, and Oliver Gableske. A novel approach to combine a SLS- and a dpll-solver for the satisfiability problem. In *Proceedings of SAT 2009*, pages 284–297, 2009.

[Biere *et al.*, 2020] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, pages 51–53, 2020.

[Biere, 2014] Armin Biere. Yet another local search solver and lingeling and friends entering the sat competition 2014. *Sat competition*, 2014(2):65, 2014.

[Cai *et al.*, 2015] Shaowei Cai, Chuan Luo, and Kaile Su. CCAnr: A configuration checking based local search solver for non-random satisfiability. In *Proceedings of SAT 2015*, pages 1–8, 2015.

[Habet *et al.*, 2002] Djamal Habet, Chu Min Li, Laure Devendeville, and Michel Vasquez. A hybrid approach for SAT. In *Proceedings of CP 2002*, pages 172–184, 2002.

[Heule *et al.*, 2016] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Proceedings of SAT 2016*, pages 228–245, 2016.

[Kautz and Selman, 1992] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of ECAI 1992*, pages 359–363, 1992.

[Kochemazov *et al.*, 2019] Stepan Kochemazov, Oleg Zaikin, Victor Kondratiev, and Alexander Semenov. Maplelcmdistchronobt-dl, duplicate learnts heuristic-aided solvers at the sat race 2019. *Proceedings of SAT Race*, pages 24–24, 2019.

[Letombe and Marques-Silva, 2008] Florian Letombe and João Marques-Silva. Improvements to hybrid incremental SAT algorithms. In *Proceedings of SAT 2008*, pages 168–181, 2008.

[Li and Li, 2012] Chu Min Li and Yu Li. Satisfying versus falsifying in local search for satisfiability. In *Proceedings of SAT 2012*, pages 477–478, 2012.

[Liang *et al.*, 2016] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proceedings of SAT 2016*, pages 123–140, 2016.

[Mazure *et al.*, 1998] Bertrand Mazure, Lakhdar Sais, and Éric Grégoire. Boosting complete techniques thanks to local search methods. *Ann. Math. Artif. Intell.*, 22(3-4):319–331, 1998.

[Moskewicz *et al.*, 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001*, pages 530–535, 2001.

[Newman *et al.*, 2018] Neil Newman, Alexandre Fréchette, and Kevin Leyton-Brown. Deep optimization for spectrum repacking. *Commun. ACM*, 61(1):97–104, 2018.

[Selman *et al.*, 1997] Bart Selman, Henry A. Kautz, and David A. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of IJCAI 97*, pages 50–54, 1997.

[Silva and Sakallah, 2000] João P. Marques Silva and Karem A. Sakallah. Boolean satisfiability in electronic design automation. In *Proceedings of the DAC 2000*, pages 675–680, 2000.