Deep Cooperation of CDCL and Local Search for SAT

Shaowei Cai[1,2(✉)] and Xindi Zhang[1,2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
{caisw,zhangxd}@ios.ac.cn
[2] School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China

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. For example, 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. 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 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." [35] on standard application benchmarks.

Keywords: CDCL · Local search · 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

S. Cai and X. Zhang—The authors are considered to have equal contributions. Cai contributes mostly on the ideas and partly on the implementations and writes the paper, while Zhang contributes mostly on the implementations and partly on the ideas.

C.-M. Li and F. Manyà (Eds.): SAT 2021, LNCS 12831, pp. 64–81, 2021.
https://doi.org/10.1007/978-3-030-80223-3_6

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 [37], mathematical theorem proving [20], AI planning [21], spectrum allocation [32], among others. Also, SAT solvers are often used as a core component of more complex tools such as solvers for Satisfiability Module Theory (SMT), which are indispensable for program analysis and software verification.

Many approaches have been proposed to solve SAT, among which conflict driven clause learning (CDCL) is the most popular one. 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. Indeed, one of the main reasons of the widespread use of SAT is that CDCL solvers are so effective in practice. CDCL is evolved from the DPLL backtracking procedure [14], and usually involves a number of key techniques, mainly including 1) clause learning from conflicts [36], 2) exploiting the structure of conflicts during clause learning [36], 3) learnt clause management scheme [4], 4) lazy data structures for the representation of formulas [31], 5) effective branching heuristics, e.g., VSIDS [31], and 6) periodically restarting [18]. Additional techniques used in recent CDCL solvers include phase saving [34], switching between "stabilizing" mode (seldom-restart) and frequent-restart mode [33], clause veriification [29], among others.

On the other hand, there is another paradigm named local search, which is a main incomplete method biased towards the satisfiable side. Local search SAT solvers begin with a complete assignment and iteratively modify the assignment until a model is found or a resource limit (usually the time limit) is reached. Although local search solvers usually have poor performance on application instances, they may be competitive on certain types of instances [10,12,26].

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 [3,5,19,24,30]. 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, while the other two ideas concern with using information produced by the local search solver to enhance CDCL. We summarize the three techniques below.

– Explore promising branches by local search (Sect. 3)
  The first idea is to a novel method to plug a local search solver into a CDCL solver. We relax the backtrack process by allowing some promising branches to be extended to a complete assignment without backtracking, even if conflicts are met during extending the assignment. Then, a local search solver is called to find a model nearby. 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.

- Phase selection with local search assignments (Sect. 4)
  Phase selection refers to pick a truth value (usually called phase) to assign the branching variable. Most modern CDCL solvers implement a phase selection heuristic named phase saving [34], which keeps the branching phase and uses the saved phase when a variable is picked to branch. Recent progress shows that using some other forms of *target phase*, e.g., the value under the largest conflict-free assignment in the solver, random value and the opposite of the saved phase, to reset the saved phase periodically could be beneficial [10]. We propose a phase resetting technique, which mainly relies on the assignments produced by the integrated local search solver.
- Branching with local search conflict information (Sect. 5)
  We use the variables' *conflict frequency*, i.e., the frequency appearing in unsatisfied clauses during local search, to enhance the branching heuristic in CDCL. Specifically, such information is used to modify the variables' *activity* in VSIDS heuristic and the variables' *learning rate* in LRB heuristic.

We apply our techniques to three state-of-the-art CDCL solvers, including the latest version of glucose [4], and the winner of the Main track of SAT Competition 2019 and 2020 namely MapleLCMDistChronoBT-DL [22] and Kissat_sat [10]. The experimental results show that our techniques allow them to solve a remarkable number of additional instances in the main track benchmark of SAT Competition 2017–2020. For example, 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 also shows better results on a real world benchmark arising from a spectrum repacking problem in the context of bandwidth auction.

Seen from experiments, the promising branches exploration technique and the local search based phase resetting techniques are very helpful to solve satisfiable instances, with a price of slight degradation on unsatisfiable instances (usually solving 2 or 3 fewer unsatisfiable instances). The local search conflict frequency enhanced branching strategy can be positive to satisfiable and also saves back a few unsatisfiable instances. Overall, these techniques significantly improves the performance of the CDCL solvers, leading to a remarkable increase on the number of total solved instances.

2 Preliminaries

2.1 Preliminary Definitions and Notations

Let $V = \{x_1, x_2, ..., x_n\}$ be a set of Boolean variables, a *literal* is either a variable x or its negation $\neg x$. A *clause* is a disjunction of literals. A clause that contains only one single literal is called a *unit clause*. A Conjunctive Normal Form (CNF) formula $F = C_1 \wedge C_2 \wedge ... \wedge C_m$ is a conjunction of clauses.

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 CNF formula F is satisfiable iff there is at least one satisfying assignment. The empty clause $\square$ is always unsatisfiable, and represents a conflict. SAT is the problem of deciding whether a given CNF formula is satisfiable.

The process of conditioning a CNF formula F on a literal ℓ amounts to removing the clauses containing an occurrence of ℓ and all occurrences of $\neg\ell$. 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 conditioned on 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, which is organized as a tree. Each node of the tree corresponds to a partial assignment, and the out edges represent the two *branching value* (also known as *branching phase*) for a variable. The root represents the empty assignment, while 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 α_max to denote the largest conflict-free assignment that has been encountered by the solver so far.

Algorithm 1 shows 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. *UnitPropagation* 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. This is done by the *ConflictAnalysis* function. Finally, *BackTrack* backtracks to the decision level computed by ConflictAnalysis. Note that Algorithm 1 shows the skeleton of a typical CDCL algorithm, and does not describe a few often used techniques, including restarts, clause deletion polices, learnt clause simplification, among others.

We introduce two branching heuristics that are used to pick the variable to assign in CDCL, which are used in the studied solvers of this paper.

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

```
1  dl ← 0;            //decision level
2  if UnitPropagation(F,α)==CONFLICT then return UNSAT
3  while ∃ unassigned variables do
       /* PickBranchVar picks a variable to assign and picks the
          respective value                                        */
4      (x, v) ← PickBranchVar(F, α);
5      dl ← dl + 1;
6      α ← α ∪ {(x, v)};
7      if UnitPropagation(F,α)==CONFLICT then
8          bl ← ConflictAnalysis(F, α);
9          if bl < 0 then
10             return UNSAT;
11         else
12             BackTrack(F, α, bl);
13             dl ← bl;
14 return SAT;
```

Variable State Independent Decaying Sum (VSIDS) [31]: Here we describe the version used in MiniSAT [15] and most modern CDCL solvers. Each variable has an *activity* attached to it. Every time a variable occurs in a recorded conflict clause, its activity is increased. This is referred to as *bumping*. 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. When selecting a branching variable, VSIDS picks the variable with the maximum activity score.

Learning Rate Branching (LRB) [27]: It 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. The authors of LRB proposed to solve the optimization problem via a Multi-Armed Bandit algorithm.

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

In this work, we use our methods to improve CDCL solvers and carry out extensive experiments to evaluate the effectiveness of the methods. In this subsection, we introduce the experiment setup including base solvers, benchmarks, running environment and reporting methodology.

Base Solvers: We choose three state of the art CDCL solvers as the base solvers for our studies, including glucose (v4.2.1)[1] [4], MapleLCMDistChronoBT-DL (v2.1)[2] [22], and Kissat_sat (2414b6d)[3] [10]. 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 [12] 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 [9]. CCAnr is a local search solver with the aim for solving structured SAT instances and has shown competitive results on various structured instances from SAT competitions and applications.

Benchmarks: The experiments are carried out with the main track benchmarks of the latest four SAT Competitions/Race (2017–2020). Additionally, we evaluate the solvers on an important application benchmark suite consisted of 10000 instances[4] from the spectrum repacking in the context of bandwidth auction which resulted in about 7 billion dollar revenue [32].

Experiment Setup: All 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 the penalized run time 'PAR2' (as used in SAT Competitions), where the run time of a failed run is penalized as twice the cutoff time.

3 Exploring Promising Branches by Local Search

In this section, we present our method for plugging a local search solver into a CDCL solver. The method helps finding a model faster, by exploring promising branches via local search.

[1] http://sat-race-2019.ciirc.cvut.cz/solvers/glucose-4.2.1.zip.
[2] http://sat-race-2019.ciirc.cvut.cz/solvers/MapleLCMDistChronoBT-DL-v2.1.zip.
[3] https://github.com/arminbiere/kissat.git.
[4] https://www.cs.ubc.ca/labs/beta/www-projects/SATFC/cacm_cnfs.tar.gz.

First, we provide the motivation of our method. By using reasoning techniques, CDCL solvers are able to prune most of the branches of the search tree. This is useful for solving unsatisfiable instances—to prove a formula is unsatisfiable, a CDCL solver needs to examine the whole search space, and therefore the more of the search tree is pruned, the more efficient the solver is. However, when solving satisfiable formulas, 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. In our opinion, 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.

Now, we present a method to explore promising branches during the search procedure of 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.
- $\frac{|\alpha|}{|\alpha\_max|} > q$ and there is no conflict under α, where q is set to 0.9 similarly.

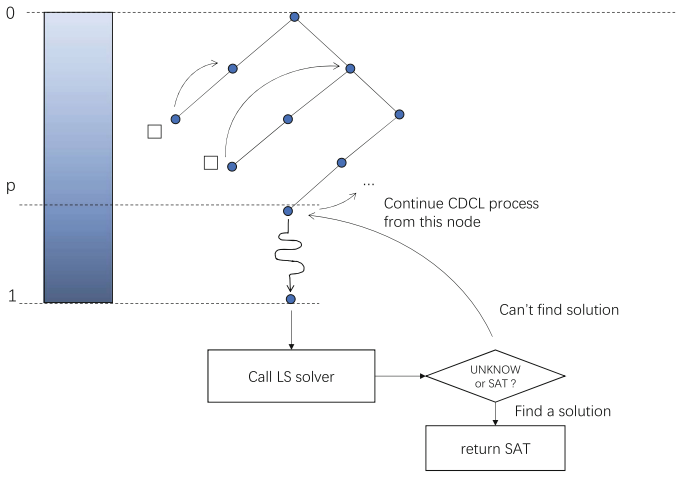

Fig. 1. Overall Procedure of Relaxed CDCL

With the conditions of promising assignments, the method is described as follows (depicted in Fig. 1). During the search of CDCL, whenever reaching a node corresponding to a promising assignment, 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, then the algorithm goes back to the normal CDCL search from the node where it was interrupted (we call this a breakpoint). The non-backtracking phase does not change the data structures used for CDCL search process. In this work, each call of the local search solver is cutoff when reaching a certain amount of memory accesses (5×10^7).

4 Phase Resetting with Local Search Assignments

In Sect. 3, we propose a method to plug a local search solver into boost CDCL solvers. Now, we propose a phase resetting heuristic based on the assignments obtained by the local search processes.

Phase selection is an important component of a CDCL solver. Most modern CDCL solvers utilize the phase saving heuristic [34], which returns the phase of a variable x corresponding to the last time x was assigned. This caching scheme reduces the repetition caused by non-chronological backtracking. Recently, Biere et al. proposed a phase resetting technique which overwrites all saved phases with some other information, based on the interval of number of conflicts encountered, which gives another boost to the performance [10].

Algorithm 2 describes a CDCL solver that implements the idea of exploring promising branches and phase resetting technique. After each time the CDCL solver is restarted, the technique overwrites the saved phases of all variables with assignments produced by local search. To this end, we record the best assignment (with the fewest unsatisfied clauses) in each run of the local search solver, and when we say the assignment of a local search procedure (run), we refer to the best assignment 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.

- α_max_LS. This refers to the assignment of the local search procedure in which the initial solution is extended based on α_max. Thus, whenever α_max is updated, the algorithm calls the local search solver and updates α_max_LS.
- α_latest_LS. This is the assignment of the latest local search procedure.
- α_best_LS. Among all local search assignments so far, we denote the best one (with the fewest unsatisfied clauses) as α_best_LS.

It is easy to see that α_max_LS and α_best_LS serve for the aim to maximize the depth of the branch, while α_latest_LS adds diversification in some sense, as different local search procedures start with initial assignments built upon different branches. Overall, it is expected this phase resetting technique with local search assignments would work well particularly for satisfiable instances, and our experiment results confirm this.

Phase Resetting Based on Local Search Assignment: Whenever the CDCL is restarted, we overwrites the saved phases. For each variable x, its phase is set according to the following probability distribution (Table 1).

Algorithm 2: Relaxed CDCL Algorithm with Phase Reset

```
1  dl ← 0, α ← ∅, α_max ← ∅ ;
2  if UnitPropagation(F,α)==CONFLICT then
3      return UNSAT
4  while ∃ unassigned variables do
5      (x, v) ← PickVariable(F, α);
6      dl ← dl + 1;
7      α ← α ∪ {(x, v)};
8      if UnitPropagation(F,α)==CONFLICT then
9          bl ← ConflictAnalysis(F, α);
10         if bl < 0 then
11             return UNSAT
12         else
13             α_max ← max(α_max, α);
14             BackTrack(F, α, bl), dl ← bl;
       /* lines 15-22 corresponds to the technique in Section 3       */
15     else if (|α|/|V| > p OR |α|/|α_max| > q) then
16         β ← α;
17         while β is not complete do
18             (x, v) ← PickVariable(F, β);
19             β ← β ∪ {(x, v)};
20             UnitPropagation(F,β);
21         if LocalSearch(β, terminate_condition) then
22             return SAT
23     if Meet Restart Conditions then
24         BackTrack(F, α, 0) ;
25         dl ← 0;
26         PhaseReset();          //corresponds to Section 4
27 return SAT;
```

Table 1. Probability of different phases in our phase resetting mechanism

Phase Name	$\alpha\_max\_LS[x]$	$\alpha\_latest\_LS[x]$	$\alpha\_best\_LS[x]$	no change
Probability	20%	65%	5%	10%

5 Branching with Conflict Frequency in Local Search

CDCL is a powerful framework owing largely to the utilization of the conflict information, and branching strategies aim to promote conflicts. In this section, we use a variable property which we refer to as *conflict frequency* in local search to improve the branching strategy of CDCL.

The best known branching strategy is VSIDS (Variable State Independent Decaying Sum) [31], which is surprisingly effective and also works well with

restarts. Although variants [7,8,15–17] have been proposed over the years, they are similar in spirit to VSIDS in the sense that they prefer to pick variables participating in recent conflicts. Briefly speaking, the VSIDS heuristic maintains an activity score for each variable, and prefers to pick the variable with the maximum activity score. The activity score of a variable reflects the frequency that it occurs in conflicts, with emphasis on those in the recent period (please refer to [31] and [15] for more details).

Recently, a new branching strategy LRB (the learning rate based branching heuristic) [27] shows its effectiveness in the Maple series, which regularly won gold medals in main track of SAT Competitions since 2016. LRB is based on the concept called *learning rate*, which measures the portion of learnt clauses involving the variable among all learnt clauses in the period between the assignment of x until it transitions back to being unassigned (please refer to [27] for more details).

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.

Definition 1. *In a local search process for SAT, for a variable x, its conflict frequency, denoted as $ls\_confl\_freq(x)$, is 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.*

Now we describe how to use the *local search conflict frequency* in the branching strategies. As $ls\_confl\_freq(x)$ is a real number between 0 and 1, we first transfers it to an integer number so that it can be combined well with VSIDS and LRB. 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 Experiments

We carry out extensive experiments to evaluate the effectiveness of our methods. The experiment setup is described in Sect. 2.4. For glucose and MapleLCMDistChronoBT-DL-v2.1, we implement all the three techniques in this work, including relaxed CDCL with local search (denoted as `rx`), phase

resetting with local search (denoted as `rp`) and local search conflict frequency enhanced branching (denoted as `cf`). For Kissat_sat, we only implement the `cf` technique, as it is challenging to implement the relaxed CDCL framework in it, due to the difficulty of identifying all current clauses (which should be provided to local search) in the Kissat_sat solver. Nevertheless, it is easy to apply the `cf` technique to Kissat, which is what we do in this work. All the source codes, origin experiment statistics and the detailed data for Table 2 can be downloaded online.[5]

Evaluations on Benchmarks of SAT Competitions. The results of evaluations of all the base solvers and the different versions with our techniques are reported in Table 2. According to the results, we have some observations.

Table 2. Experiment results on benchmarks from SAT Competitions 2017–2020, where Maple-DL-v2.1 is short for MapleLCMDistChronoBT-DL-v2.1

solver	#SAT	#UNSAT	#Solved	PAR2	#SAT	#UNSAT	#Solved	PAR2
	SC2017(351)				SC2018(400)			
glucose_4.2.1	83	101	184	5220.0	95	95	190	5745.9
glucose+rx	88	95	183	5237.0	113	95	208	5283.4
glucose+rx+rp	112	94	206	4618.2	141	87	228	4698.3
glucose+rx+rp+cf	110	94	204	4668.5	150	91	241	4438.2
Maple-DL-v2.1	101	113	214	4531.0	133	102	235	4533.9
Maple-DL+rx	101	112	213	4520.3	149	101	250	4148.6
Maple-DL+rx+rp	111	103	214	4447.1	158	93	251	4147.2
Maple-DL+rx+rp+cf	116	107	223	4139.4	162	97	259	3927.6
Kissat_sat	115	114	229	3943.5	167	98	265	3786.4
Kissat_sat+cf	113	113	226	4001.0	178	104	282	3409.4
CCAnr	13	N/A	13	9629.9	56	N/A	56	8622.0
	SC2019(400)				SC2020(400)			
glucose_4.2.1	118	86	204	5437.6	68	91	159	6494.6
glucose+rx	120	84	204	5443.9	93	88	181	6018.1
glucose+rx+rp	134	85	219	5096.3	130	85	215	5123.7
glucose+rx+rp+cf	140	85	225	4923.6	134	87	221	4977.9
Maple-DL-v2.1	143	97	240	4601.8	86	104	190	5835.7
Maple-DL+rx	146	93	239	4602.1	121	105	226	4977.8
Maple-DL+rx+rp	155	94	249	4416.3	142	99	241	4589.2
Maple-DL+rx+rp+cf	154	95	249	4377.4	151	106	257	4171.1
Kissat_sat	159	88	247	4293.5	146	114	260	4048.8
Kissat_sat+cf	162	90	252	4211.7	157	113	270	3896.8
CCAnr	13	N/A	13	9678.3	45	N/A	45	8978.7

– The `rx` technique improves glucose and MapleLCMDistChronoBT-DL-v2.1 on solving satisfiable instances, particularly for the benchmarks of 2018 (increased by 18 and 16 for #SAT) and 2020 (increased by 25 and 35 for

[5] https://github.com/caiswgroup/relaxed-sat.

#SAT). On the other hand, the glucose+rx and Maple-DL+rx have slightly worse performance than the original versions on UNSAT instances, and the decrease on #UNSAT is only 2 on average, considering both solvers on all benchmarks.

- By adding the `rp` technique, glucose+rx+rp and Maple-DL+rx+rp gain further improvement on #SAT, which is significant for all benchmarks. Specifically, the #SAT number of glucose+rx+rp is greater than glucose+rx by 24, 28, 14 and 37 for benchmarks of 2017, 2018, 2019 and 2020 respectively, and the increment is 10, 9, 9 and 21 for Maple-DL+rx+rp over Maple-DL+rx. Similar to the `rx` technique, we observe slight degradation on solving UNSAT instances, and the decrease on #UNSAT is 3 on average for both solvers.
- The impact of the `cf` technique can be seen from the comparisons of glucose+rx+rp vs. Glucose+rx+rp+cf, Maple-DL+rx+rp vs. Maple-DL+rx+rp+cf, and Kissat_sat vs. Kissat_sat+cf. Overall, the `cf` technique is positive for solving both satisfiable and unsatisfiable instances on all benchmarks, with the exceptions of glucose+rx+rp+cf and Kissat_sat+cf on the 2017 benchmark (dropping 2 and 3 instances). For the benchmarks of 2018, 2019 and 2020, the `cf` technique leads to a remarkable increment on the #Solved number, which is (13, 6, 6) for glucose+rx+rp+cf, (8, 0, 7) for Maple-DL+rx+rp+cf, and (17, 5, 10) for Kissat_sat+cf. Particularly, noting that Kissat_sat is the winner of Main Track in SC 2020 and represents the latest state of the art, such improvements are remarkable by a single technique.
- 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). We note that, 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 Benchmarks of Spectrum Repacking. We also carry out experiments on a suite of instances arising from an important real world project—the spectrum repacking project in US Federal Communication Commission (FCC). The instances from this project was available on line[6] [32]. This benchmark contains 10 000 instances, including both satisfiable and unsatisfiable instances. We compare each base CDCL solver with its final version using our techniques, as well as the underlying local search solver CCAnr.

The results on this benchmark suite are reported in Table 3. According to the results, for each of the base CDCL solvers, the improved version with our techniques has better performance than the base solver. Particularly, the Maple-

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

DL+rx+rp+cf solver solves the most instances (8759+218=8977), significantly better than all the other solvers.

Table 3. Comparing with state-of-the-art solvers on FCC. glucose+ is short for glucose+rx+rp+cf, and malple+ is short for Maple-DL+rx+rp+cf.

Benchmark	glucose	glucose+	Maple	Maple+	kissat_sat	kissat_sat+cf	CCAnr
	#SAT	#SAT	#SAT	#SAT	#SAT	#SAT	#SAT
	#UNSAT	#UNSAT	#UNSAT	#UNSAT	#UNSAT	#UNSAT	#UNSAT
	#Solved	#Solved	#Solved	#Solved	#Solved	#Solved	#Solved
	PAR2	PAR2	PAR2	PAR2	PAR2	PAR2	PAR2
FCC (10000)	7330	8075	8084	8759	8192	8214	7853
	187	197	215	218	207	211	0
	7517	8272	8299	8977	8399	8425	7853
	2555.85	1850.58	1867.13	1243.66	1760.55	1734.61	2215.35

Further Analyses on the Cooperation. We perform more analyses to study the role of local search in the hybrid solvers based on glucose and MapleLCMDIstChronoBT-DL. This experiment does not include Kissat_sat as we do not apply the relaxed CDCL framework to it and the statistics in this experiment are not applicable to Kissat_sat+cf. Some important information is provided in Table 4.

We can see that the local search solver returns a solution for some instances, and this number varies considerably with the benchmarks. A natural question is: *Whether the improvements come mainly from the complementation of CDCL and local search solvers that they solve different instances?* If this were true, then a simple portfolio that runs both CDCL and local search solvers would work similarly to the hybrid solvers in this work. 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 (both the CDCL and local search solver are given 5000 s for each instance). We observe that, there is a large number of instances (denoted by #SAT_bonus) that both CDCL and local search solvers fail to solve but can be solved by the hybrid solvers. For these instances, even a virtual best solver that picks the solver with the best result for each instance would fail. 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.

We also calculate the number of calls of the local search solver in each run. This figure is usually from 10 to 25 for these benchmarks. As for the run time of local search, which can be seen as the price paid for the benefit of using local search, we calculate the portion of the time spent on local search. This figure is between 6% and 20% for the satisfiable instances, and it drops significantly on unsatisfiable instances, which is usually less than 7%. This is not inconsistent with the observations that the number of local search calls is not necessarily

Table 4. Analyses on the impact of Local Search on the CDCL solvers. Maple is short for Maple-DL to save space, #byLS is the number of instance for which the solution is given by the local search solver, #SAT_bonus is the number of instances for which both base CDCL solver and Local Search solver fail to solve but the hybrid solver finds a satisfiable solution. #LS_call is the average number of calls on Local Search, while LS_time is the average value of the proportion of time (in percentage %) spent on local search in the whole run, and these two figures are calculated for satisfiable and unsatisfiable instances respectively.

Solver	Analysis for SAT				Analysis for UNSAT	
	#byLS	#SAT_bonus	#LS_call	LS_time(%)	#LS_call	LS_time(%)
	SC2017(351)					
glucose+rx	20	11	24.28	21.66	16.36	5.52
glucose+rx+rp	10	33	17.77	18.46	14.33	4.86
glucose+rx+rp+cf	17	29	22.7	22.19	15.3	5.81
Maple+rx	16	9	13.86	7.52	11.18	2.03
Maple+rx+rp	11	15	9.63	10.43	6.54	2.36
Maple+rx+rp+cf	6	16	12.59	7.49	8.59	2.12
	SC2018(400)					
glucose+rx	50	4	11.27	20.66	29.62	4.94
glucose+rx+rp	47	31	9.46	18.4	21.66	5.64
glucose+rx+rp+cf	53	36	11.43	20.28	20.62	6.64
Maple+rx	52	7	4.8	13.02	11.69	2.81
Maple+rx+rp	56	13	4.84	15.21	8.7	3.04
Maple+rx+rp+cf	51	18	6.52	12.53	15.62	2.94
	SC2019(400)					
glucose+rx	14	8	26.46	10.79	17.42	6.39
glucose+rx+rp	10	26	22.68	8.67	14.59	5.14
glucose+rx+rp+cf	11	26	20.39	11.82	15.51	5.95
Maple+rx	14	7	12.66	2.67	12.94	1.98
Maple+rx+rp	9	14	8.6	3.17	16.59	2.53
Maple+rx+rp+cf	12	15	11.21	3.05	17.23	2.22
	SC2020(400)					
glucose+rx	30	9	14.94	11.75	14.67	10.27
glucose+rx+rp	23	37	13.17	10.79	9.4	9.71
glucose+rx+rp+cf	23	37	12.78	11.67	10.52	10.28
Maple+rx	19	13	14.21	6.69	10.24	5.25
Maple+rx+rp	30	29	8.53	6.62	11.7	6.18
Maple+rx+rp+cf	23	36	10.95	6.05	14.17	5.42

fewer on unsatisfiable instances, because the portion of the time on local search also depends on the total time of the hybrid solver. Our statistics show that the averaged time on solving UNSAT instances is about 1.5× to 2× that on SAT instances for both glucose+rx+rp+cf and Maple-DL+rx+rp+cf. In a nutshell, the price is acceptable and usually small for the UNSAT instances, which also

partly explains that our techniques do not have obvious negative impact on solving UNSAT instances although they incline to the satisfiable side.

7 Related Works

There has been interest in combining systemic search and local search for solving SAT. Indeed, it was pointed as a challenge by Selman et al. [35]. Previous attempts can be categorized into two families according to the type (DPLL/CDCL or local search) of the main body solver.

A family of hybrid solvers use a local search solver as the main body solver. An incomplete hybrid solver hybridGM [5] calls CDCL search around local minima with only one unsatisfied clause. Audemard et al. proposed a hybrid solver named SATHYS [2,3]. Each time the local search solver reaches a local minimum, a CDCL solver is launched. Some reasoning techniques or information from CDCL solvers have been used to improve local search solvers. Resolution techniques were integrated to local search solvers [1,13]. Recently, Lorenz and Wörz developed a hybrid solver GapSAT [28], which used a CDCL solver as a preprocessor before running the local search solver ProbSAT. The experiments showed that, the learnt clauses produced by CDCL solver were useful to improve the local search solver on random instances.

The other family of hybrid solvers focus on boosting CDCL solvers by local search, and this work belongs to this line. A simple way for hybridizing local search and CDCL is to call local search before CDCL begins, trying to solve the instance or derive information such as variable ordering to be used in CDCL. The hyrid solvers Sparrow2Riss [6], CCAnr+glucose [11] and SGSeq [25] belong to this family.

Some works use local search to find a subformula for CDCL to solve. In [30], a local search solver is used to find a part of the formula which is satisfiable, which helps to divide the formula into two parts for the DPLL solver. In HINOTOS [24], a local search is used to identify a subset of clauses to be passed to a CDCL solver in an incremental way.

The most related works belong to those that call a local search solver during the CDCL procedure. WalkSatz [19] calls a local search solver WalkSAT at each node of a DPLL solver Satz. However, this is time consuming. This can be done in parallel with shared memory [23]. In CaDiCaL and Kissat [10], a local search solver is called when the solver resets the saved phases, and the phases produced by local search are used only once immediately after the local search process. However, the way CaDiCaL and Kissat use the local search assignments is different from our phase resetting method based on local search. CaDiCaL and Kissat only record the current local search assignment, which is used just for once right after the local search exists. They do not use information of previous local search processes. In fact, we also carry out experiments to see the impact of local search on the performance of Kissat_sat, which turns out to be limited. When Kissat_sat works without local search, #SAT drops by 5 on average over the SC benchmarks.

Although previous attempts have been made trying to combine the strength of CDCL and local search, they did not lead to hybrid solvers essentially better than CDCL solvers on application instances. This work, for the first, meets the standard of the challenge "create a new algorithm that outperforms the best previous examples of both approaches" [35] on standard application benchmarks from SAT Competitions.

8 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. The first idea is to protect promising branches from being pruned, and exploit them using a local search solver. The second idea is to utilize the assignments of the local search processes to reset the saved phases in the phase selection heuristic. Finally, we proposed to enhance the branching strategy of CDCL solvers by considering the conflict frequency of variables in the local search process. These techniques significantly improve the performance of state of the art CDCL solvers on application benchmarks. The proposed methods are generic and can be applied to improve other 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 [35].

Acknowledgement. This work is supported by Beijing Academy of Artificial Intelligence (BAAI), and Youth Innovation Promotion Association, Chinese Academy of Sciences [No. 2017150].

References

1. Anbulagan, Pham, D.N., Slaney, J.K., Sattar, A.: Old resolution meets modern SLS. In: Proceedings of AAAI 2005, pp. 354–359 (2005)
2. Audemard, G., Lagniez, J., Mazure, B., Sais, L.: Integrating conflict driven clause learning to local search. In: Proceedings of LSCS 2009, pp. 55–68 (2009)
3. Audemard, G., Lagniez, J.-M., Mazure, B., Saïs, L.: Boosting local search thanks to CDCL. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 2010. LNCS, vol. 6397, pp. 474–488. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16242-8_34
4. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of IJCAI 2009, pp. 399–404 (2009)
5. Balint, A., Henn, M., Gableske, O.: A novel approach to combine a SLS- and a DPLL-solver for the satisfiability problem. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 284–297. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_28
6. Balint, A., Manthey, N.: SparrowToRiss 2018. In: Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions, pp. 38–39 (2018)

7. Biere, A.: Adaptive restart strategies for conflict driven SAT solvers. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 28–33. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79719-7_4
8. Biere, A.: Pre, icosat@sc'09. In: SAT 2009 Competitive Event Booklet, pp. 42–43 (2009)
9. Biere, A.: Yet another local search solver and lingeling and friends entering the sat competition 2014. Sat Competition **2014**(2), 65 (2014)
10. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Paracooba, Plingeling and Treengeling entering the SAT Competition, Kissat, pp. 51–53 (2020)
11. Cai, S., Luo, C., Su, K.: CCAnr+glucose in SAT Competition 2014. In: Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions, p. 17 (2014)
12. Cai, S., Luo, C., Su, K.: CCAnr: a configuration checking based local search solver for non-random satisfiability. In: Proceedings of SAT 2015, pp. 1–8 (2015)
13. Cha, B., Iwama, K.: Adding new clauses for faster local search. In: Proceedings of AAAI, vol. 96, pp. 332–337 (1996)
14. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
15. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
16. Gershman, R., Strichman, O.: Haifasat: a new robust SAT solver. In: Ur, S., Bin, E., Wolfsthal, Y. (eds.) Proceedings of Haifa Verification Conference 2005, pp. 76–89 (2005)
17. Goldberg, E.I., Novikov, Y.: Berkmin: a fast and robust sat-solver. In: Proceedings of DATE (2002), pp. 142–149 (2002)
18. Gomes, C.P., Selman, B., Kautz, H.A.: Boosting combinatorial search through randomization. In: Proceedings of AAAI/IAAI 1998, pp. 431–437 (1998)
19. Habet, D., Li, C.M., Devendeville, L., Vasquez, M.: A hybrid approach for SAT. In: Proceedings of CP 2002, pp. 172–184 (2002)
20. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 228–245. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_15
21. Kautz, H.A., Selman, B.: Planning as satisfiability. In: Proceedings of ECAI 1992, pp. 359–363 (1992)
22. Kochemazov, S., Zaikin, O., Kondratiev, V., Semenov, A.: Maplelcmdistchronobt-dl, duplicate learnts heuristic-aided solvers at the sat race 2019. In: Proceedings of SAT Race, pp. 24–24 (2019)
23. Kroc, L., Sabharwal, A., Gomes, C.P., Selman, B.: Integrating systematic and local search paradigms: a new strategy for maxsat. In: Proceedings of IJCAI 2009, pp. 544–551 (2009)
24. Letombe, F., Marques-Silva, J.: Improvements to hybrid incremental SAT algorithms. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 168–181. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79719-7_17
25. Li, C.M., Habet, D.: Description of RSeq2014. In: Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions, p. 72 (2014)
26. Li, C.M., Li, Yu.: 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). https://doi.org/10.1007/978-3-642-31612-8_43

27. Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 123–140. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40970-2_9
28. Lorenz, J.-H., Wörz, F.: On the effect of learned clauses on stochastic local search. In: Pulina, L., Seidl, M. (eds.) SAT 2020. LNCS, vol. 12178, pp. 89–106. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51825-7_7
29. Luo, M., Li, C., Xiao, F., Manyà, F., Lü, Z.: An effective learnt clause minimization approach for CDCL SAT solvers. In: Proceedings of IJCAI 2017, pp. 703–711 (2017)
30. Mazure, B., Sais, L., Grégoire, É.: Boosting complete techniques thanks to local search methods. Ann. Math. Artif. Intell. **22**(3–4), 319–331 (1998)
31. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, DAC 2001, pp. 530–535 (2001)
32. Newman, N., Fréchette, A., Leyton-Brown, K.: Deep optimization for spectrum repacking. Commun. ACM **61**(1), 97–104 (2018)
33. Oh, C.: Between SAT and UNSAT: the fundamental difference in CDCL SAT. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 307–323. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_23
34. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 294–299. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72788-0_28
35. Selman, B., Kautz, H.A., McAllester, D.A.: Ten challenges in propositional reasoning and search. In: Proceedings of IJCAI, vol. 97, pp. 50–54 (1997)
36. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Proceedings of ICCAD 1996, pp. 220–227 (1996)
37. Silva, J.P.M., Sakallah, K.A.: Boolean satisfiability in electronic design automation. In: Proceedings of the DAC 2000, pp. 675–680 (2000)