SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

July 2019, Vol. 62 072103:1–072103:13
https://doi.org/10.1007/s11432-017-9467-7

A branching heuristic for SAT solvers based on complete implication graphs

Fan XIAO[1], Chu-Min LI[2*], Mao LUO[1], Felip MANYÀ[3], Zhipeng LÜ[1] & Yu LI[2]

[1]*School of Computer Science, Huazhong University of Science and Technology, Wuhan* 430074*, China;*
[2]*Modélisation Information et Systèmes* (*MIS*)*, University of Picardie Jules Verne, Amiens* 80090*, France;*
[3]*Artificial Intelligence Research Institute* (*IIIA-CSIC*)*, Barcelona* 08193*, Spain*

Received 24 November 2017 /Revised 5 February 2018 /Accepted 20 March 2018/Published online 17 April 2019

Abstract The performance of modern conflict-driven clause learning (CDCL) SAT solvers strongly depends on branching heuristics. State-of-the-art branching heuristics, such as variable state independent decaying sum (VSIDS) and learning rate branching (LRB), are computed and maintained by aggregating the occurrences of the variables in conflicts. However, these heuristics are not sufficiently accurate at the beginning of the search because they are based on very few conflicts. We propose the distance branching heuristic, which, given a conflict, constructs a complete implication graph and increments the score of a variable considering the longest distance between the variable and the conflict rather than the simple presence of the variable in the graph. We implemented the proposed distance branching heuristic in Maple_LCM and Glucose-3.0, two of the best CDCL SAT solvers, and used the resulting solvers to solve instances from the application and crafted tracks of the 2014 and 2016 SAT competitions and the main track of the 2017 SAT competition. The empirical results demonstrate that using the proposed distance branching heuristic in the initialization phase of Maple_LCM and Glucose-3.0 solvers improves performance. The Maple_LCM solver with the proposed distance branching heuristic in the initialization phase won the main track of the 2017 SAT competition.

Keywords SAT, branching heuristic, conflict-driven clause learning, implication graph

Citation Xiao F, Li C-M, Luo M, et al. A branching heuristic for SAT solvers based on complete implication graphs. Sci China Inf Sci, 2019, 62(7): 072103, https://doi.org/10.1007/s11432-017-9467-7

1 Introduction

In propositional logic, a variable x may take the truth value 0 (false) or 1 (true). A literal l is a variable x or its negation $\neg x$, a clause is a disjunction of literals, a CNF (conjunctive normal formula) formula ϕ is a conjunction of clauses, and the size of a clause is the number of literals in it. An assignment of truth values to the propositional variables satisfies a literal x if x takes the value 1 and satisfies a literal $\neg x$ if x takes the value 0, satisfies a clause if it satisfies at least one its literals, and satisfies a CNF formula if it satisfies all of its clauses. The empty clause contains no literals and is unsatisfiable; it represents a conflict. A unit clause contains exactly one literal and is satisfied by assigning the appropriate truth value to the variable. A variable is free if it has not been assigned a value. Initially, all variables are free. A literal is free if its variable is free. For convenience, when variable x is assigned a truth value, we also say that literals x and $\neg x$ are assigned. An assignment for a CNF formula ϕ is complete if each variable in ϕ has been assigned a value; otherwise, it is considered partial. The SAT problem for a CNF formula ϕ is to find an assignment to the variables that satisfies all clauses of ϕ.

* Corresponding author (email: chu-min.li@u-picardie.fr)

The SAT problem is a classical NP-complete problem. Thanks to advances in conflict-driven clause learning (CDCL) SAT solvers [1–6], reducing combinatorial problems to SAT is a powerful generic problem solving approach. Branching heuristics are one of key elements of the success of CDCL SAT solvers. A SAT solver roughly works as follows. The solver repeatedly selects a free literal according to the branching heuristic, satisfies it by assigning the appropriate truth value to its variable, and propagates all unit clauses implied by its satisfaction until the empty clause is deduced or all variables are assigned a truth value. The selected literal is referred to as the decision literal, and the decision literal variable is called the decision variable. The decision literal and all literals satisfied by the subsequent propagation constitute a decision level. If the empty clause is derived, its derivation is analyzed to learn a new clause. The new clause is added to the current CNF formula to avoid the same conflict in the remaining search and determine the decision level to which the solver has to backtrack [5, 7]. If all variables are assigned a truth value without deriving the empty clause, the problem is satisfiable and the complete assignment is returned as the solution. Otherwise, the solver derives the empty clause in all branches of the search tree, thereby proving the unsatisfiability of the problem. Thus, the branching heuristic influences how the solver derives the empty clause. A clever branching heuristic allows the solver to derive the empty clause quickly and consequently a learned clause of good quality, which reduces the search space significantly.

State-of-the-art branching heuristics such as variable state independent decaying sum (VSIDS) [7] and learning rate branching (LRB) [8] maintain a score for each variable during the search process by aggregating its occurrences in previous conflicts. Then, the variable with the greatest score is selected as the next decision variable. With the exception of parameters that can take different values, these heuristics compute the score of a variable in the same manner at the beginning, middle and end of the search process. Specifically, the scores of all variables in a conflict are incremented by a value without considering how these variables contribute to the conflict. As the search proceeds, the score of a variable computed in this manner more accurately reflects its power in leading to a conflict in the near future. However, at the beginning of the search, the score of a variable is not sufficiently accurate because it is based on aggregated occurrences in very few conflicts. Therefore, a branching heuristic should consider more information in the initialization phase of a CDCL SAT solver.

In this paper, we propose a branching heuristic, which we refer to as distance, that can be used by CDCL SAT solvers in the initialization phase. Similar to LRB and VSIDS, the distance heuristic initializes the score of all variables to 0. Then, each time the solver derives an empty clause, the heuristic constructs a complete implication graph starting from all decision literals implying the empty clause. It then increments the score of each variable in the implication graph by a value that depends on the longest distance between the variable and the conflict. In this manner, information other than the simple occurrence of the variable is considered to compute its score.

We implemented the distance heuristic in Glucose-3.0 [9] and Maple_LCM [10] (LCM is learned clause minimization), which are two of the best CDCL SAT solvers. We replaced, in the initialization phase, the original branching heuristics in these solvers with the distance heuristic and used the resulting solvers to solve the instances[1)] from the application and hard combinatorial tracks of the 2014 and 2016 SAT competitions and the main track of the 2017 SAT competition[2)]. The experimental results demonstrate that the distance heuristic significantly improves the performance of Glucose-3.0 and Maple_LCM. In particular, the solver Maple_LCM_Dist, which is an implementation of Maple_LCM that incorporates the distance heuristic [11], was ranked first in the main track of the 2017 SAT competition.

The significance of our contributions is supported by this statement of two leading SAT solver developers [12]: "We must also say, as a preliminary, that improving SAT solvers is often a cruel world. To give an idea, improving a solver by solving at least ten more instances (on a fixed set of benchmarks of a competition) is generally showing a critical new feature. In general, the winner of a competition is decided based on a couple of additional solved benchmarks."

The above statement was also quoted in [8, 10] to acknowledge the advances brought about by the conflict history-based branching (CHB) heuristic and a learned clause minimization approach for CDCL

1) http://www.satcompetition.org/.

2) In the main track of the 2017 SAT competition, the application and hard combinatorial instances are mixed.

SAT solvers.

The remainder of this paper is organized as follows. Section 2 reviews existing branching heuristics for SAT solvers. Section 3 describes the proposed distance branching heuristic. Section 4 presents experimental results, and conclusion is given in Section 5.

2 Existing branching heuristics in CDCL SAT solvers

Intensive efforts have been devoted to devising clever branching heuristics for SAT solvers [8, 13–16]. In this section, we review some of the most representative branching heuristics.

2.1 VSIDS in Chaff and BerkMin's heuristic

The VSIDS heuristic was originally introduced in Chaff [5]. VSIDS maintains a score (referred to as activity) initialized to 0 for each literal of a given SAT instance. Each time the solver derives a conflict, the VSIDS heuristic increments the score of each literal in the corresponding learned clause by 1. Periodically, the score of each literal is divided by a constant to favor the literals in recently learned clauses. The literal with the greatest score is selected as the next decision literal to satisfy recent learned clauses.

BerkMin's heuristic can be considered an improvement of the VSIDS heuristic in Chaff. First, BerkMin maintains a score activity$[x]$ for each variable x (rather than each literal of x) of the SAT instance. Each time the solver derives an empty clause, activity$[x]$ is incremented for each occurrence of a literal of x in a clause responsible for the conflict. In other words, if k clauses containing a literal of x are responsible for the conflict, activity$[x]$ is incremented by k. Thus, while Chaff only increments the scores of literals occurring in the learned clause, BerkMin increments the scores of all variables involved in the conflict, including variables in the learned clause. Second, BerkMin selects the next decision variable in the most recent learned clause that has not been satisfied. In other words, the free variable with the greatest score in the most recent unsatisfied learned clause is selected as the next decision variable. If all learned clauses are satisfied, the free variable with the greatest score is selected. Third, BerkMin assigns the value to the selected decision variable x to balance the power of unit propagation after setting x to 0 and 1, respectively.

2.2 VSIDS in recent CDCL SAT solvers

The VSIDS heuristic used in recent CDCL SAT solvers, such as MiniSat [3], Glucose [1], Lingeling [2] and CryptoMiniSat [6], can be considered an improvement of BerkMin's heuristic. The VSIDS heuristic maintains a score (activity$[x]$) initialized to 0 for each variable x of a given SAT instance. Each time the solver derives an empty clause, VSIDS increments the score of all variables involved in the conflict by a value var_inc, which is initialized to 1 and updated to var_inc/var_decay after each conflict, where var_decay is a value in the real interval $(0, 1)$. In MiniSat, var_decay $= 0.95$.

In Glucose, if a variable is assigned in the last decision level by a learned clause with small literal blocks distance (LBD) and contributes to the conflict, its score is incremented once again by var_inc, where the LBD of a learned clause is the number of different decision levels in which the literals of the clause are assigned. Here, the purpose is to favor the generation of learned clauses with small LBD [17].

Note that var_inc increases after each conflict, which means that a score that has increased due to a recent conflict has greater importance than a score increased due to an older conflict. In this manner, a solver does not need to periodically divide the score of each variable by a constant, and the variables contributing to recent conflicts are naturally favored. The intuition behind the VSIDS heuristic is that the variable with the greatest score (i.e., the variable occurring most frequently in recent conflicts) should be selected as the next decision variable. By default, to inherit the essence of the history, the last value assigned to this variable is reassigned to this variable for branching.

As indicated in a previous study [8], VSIDS and its variants have been the most effective and widespread decision heuristics for many years, and the success of VSIDS has dramatically raised the bar for new heuristics. However, VSIDS scores are insufficiently accurate to select variables at the beginning of

a search process because they are based on very few conflicts. In Glucose-2.3 [18] and its variants, this situation is remedied by varying the value of parameter var_decay at the beginning of the search. Glucose-2.3 initializes var_decay to 0.8 (rather than 0.95), and then increments var_decay by 0.01 every 5000 conflicts until var_decay reaches a cruise value of 0.95 (after 75000 conflicts). Consequently, the variable scores increase more quickly at the beginning of the search in Glucose-2.3 and its variants. In Glucose-2.3, after each conflict, the scores of all variables involved in the conflict are increased without considering how these variables contribute to the conflict.

2.3 LRB

The LRB heuristic [19] was recently proposed as an improvement to the CHB heuristic [8]. Similar to the CHB heuristic, LRB is based on the exponential recency weighted average algorithm used in non-stationary multi-armed bandit problems, and the fact that CDCL SAT solvers frequently and repeatedly assign a truth value to a variable x and cancel the value assigned to x upon backtracking. Note that the score $\mathrm{LRB}(x)$ of variable x is initialized to 0. Here, let p be the period between the time at which x is assigned a value and the time at which that value is canceled. Let age be the number of conflicts that the solver derives during period p, and let nbConflicts be the number of conflicts wherein x is involved during p. When the value of x is canceled, the heuristic updates $\mathrm{LRB}(x)$ as follows:

$$\mathrm{LRB}(x) \leftarrow (1-a) \times \mathrm{LRB}(x) + a \times \mathrm{nbConflicts/age},$$

where parameter a is a value in the real interval $(0, 1)$. In another study [8], a is initialized to 0.4 and decreased by 10^{-6} after each conflict until it reaches 0.06. As in the VSIDS heuristic, a conflict at the beginning of the search (i.e., one of the first 34×10^4 conflicts) contributes to the LRB score of a variable more than a conflict of the cruise search because the score is not sufficiently accurate at the beginning of the search process.

The intuition behind the LRB heuristic is that a free variable contributing to most conflicts during the period in which it was assigned a value is likely to help derive new conflicts quickly if it is assigned a value again. In particular, the variable contributing to all conflicts during the periods in which it was assigned a value has an LRB score close to the maximum value 1 and should be selected as the next branching variable. Note that, as in the VSIDS heuristic, the change of the LRB score of a variable due to a conflict is not dependent on how the variable contributes to the conflict.

The MapleCOMSPS and MapleCOMSPS_LRB solvers combine LRB and VSIDS heuristics and were ranked first in the main and industrial tracks of the 2016 SAT competition, respectively. In these solvers, the VSIDS heuristic is used to select decision variables for initialization when the number of conflicts is less than 10000. Then, MapleCOMSPS maintains the LRB heuristic and uses it to select decision variables for 2500 s. It then maintains the VSIDS heuristic and uses it to select decision variables for the remaining time. MapleCOMSPS_LRB regularly switches between LRB and VSIDS heuristics and allocates the same amount of time to LRB and VSIDS. See [20] for details.

The Maple_LCM solver was developed from MapleCOMSPS by incorporating an effective learned clause minimization technique [10]. In [10], Maple_LCM was referred to as Maple+. Maple_LCM inherits the initialization phase of MapleCOMSPS, wherein the VSIDS heuristic is used to select decision variables when the number of conflicts is less than 10000. Then, it uses the LRB heuristic for 2500 s before switching to the VSIDS heuristic for the remaining time. Note that Maple_LCM was the second-best solver in the main track of the 2017 SAT competition. It was beaten by Maple_LCM_Dist, which was developed from Maple_LCM by incorporating the proposed distance branching heuristic.

2.4 Look-ahead heuristics and their integration into CDCL SAT solvers

VSIDS and LRB heuristics compute the score of a variable based on past events; thus, they can be referred to as "look-back" heuristics. Differing from look-back heuristics, look-ahead heuristics compute the score of a variable by considering what will happen if this variable is assigned a truth value. A typical look-ahead heuristic is Jeroslow-Wang [15], which selects the next decision variable based on the number

of its occurrences in unsatisfied clauses (i.e., clauses not containing any satisfied literal but containing at least two free variables). We are unaware of any state-of-the-art CDCL SAT solver that combines a look-back heuristic and a look-ahead heuristic, likely because it is unclear how this could be accomplished. The first difficulty in integrating a look-ahead heuristic into a CDCL SAT solver is that, for efficiency, the solver typically does not maintain the number of occurrences of each variable in unsatisfied clauses. In fact, a CDCL SAT solver usually does not even maintain the list of unsatisfied clauses; thus, it must check the literals of a clause to know if the given clause is satisfied by the current partial assignment.

The second difficulty is how a look-ahead heuristic score is combined with a VSIDS or LRB score because these different scores do not vary at the same scale. For example, assume that the VSIDS score of a variable x is 1. Then, if x is involved in the 200th conflict occurring later, the VSIDS score of x will be increased by more than 28528 ($\approx (1/\text{var\_decay})^{200}$) if var_decay = 0.95, while a look-ahead score of x will not change to the same extent.

Thus, we have implemented another look-ahead strategy in Glucose-3.0 to break ties among variables with the greatest VSIDS score. Let ϕ be a CNF that does not contain any unit clause, let x be a free variable with the greatest VSIDS score in ϕ, and let v be its last assigned value. If unit propagation derives the empty clause from ϕ after assigning v to x, then x is selected as the next decision variable with the value v. Otherwise, $\text{UP}(\phi \wedge x = v)$ denotes the number of variables assigned in ϕ by unit propagation after assigning v to x, and S denotes the set of free variables with the greatest VSIDS score. We compute $\text{UP}(F \wedge x = v)$ for each x of the first $\min(|S|, T)$ variables in S, where T is an integer parameter. If one of these unit propagations derives the empty clause, the decision variable is selected accordingly; otherwise, the variable x with the greatest $\text{UP}(\phi \wedge x = v)$ is selected as the next decision variable with the value v. The intuition behind this strategy is to branch on the variable that allows the immediate derivation of a conflict or a CNF formula containing the lowest number of variables among the $\min(|S|, T)$ possible branches while maintaining the effectiveness of VSIDS.

We have tested this strategy in Glucose-3.0 with parameter values $T = 2, 3, \ldots, 20$. The best results were obtained with $T = 12$. Unfortunately, even with $T = 12$, this strategy did not improve the performance of Glucose-3.0, thereby confirming the difficulty of combining look-ahead and look-back heuristics in CDCL SAT solvers.

3 The proposed distance branching heuristic based on implication graphs

A common feature of the LRB and VSIDS heuristics is that they count the simple occurrences of variables in previous conflicts to compute variable scores, regardless of how these variables contribute to the conflict. Intuitively, variable scores computed in this manner are only sufficiently accurate to distinguish the variables when a large number of conflicts is considered.

In this section, we describe the proposed distance branching heuristic, which computes the score of a variable according to its distance to the empty clause rather than its simple presence in the complete implication graphs of the empty clause. The proposed distance heuristic is intended to be used at the beginning of the search process when there are very few conflicts.

3.1 Complete implication graph of the empty clause

When a CDCL SAT solver derives an empty clause during the search process, the complete implication graph $G = (V, E)$ of the empty clause can be defined recursively as follows:

(1) Graph G contains the special vertex $\square \in V$, which represents the empty clause.

(2) If c is the clause from which the empty clause is derived (i.e., the empty clause is derived because all literals of c are assigned false), then each literal l of c is a vertex of G and $(l, \square)$ is an arrow of G (i.e., $l \in V$ and $(l, \square) \in E$).

(3) If $l \in V$ and clause c' is the reason for l in the current partial assignment (i.e., $\neg l$ is a literal of c' and each other literal l' of c' is assigned false), then l' is a vertex of G and (l', l) is an arrow of G (i.e., $l' \in V$ and $(l', l) \in E$).

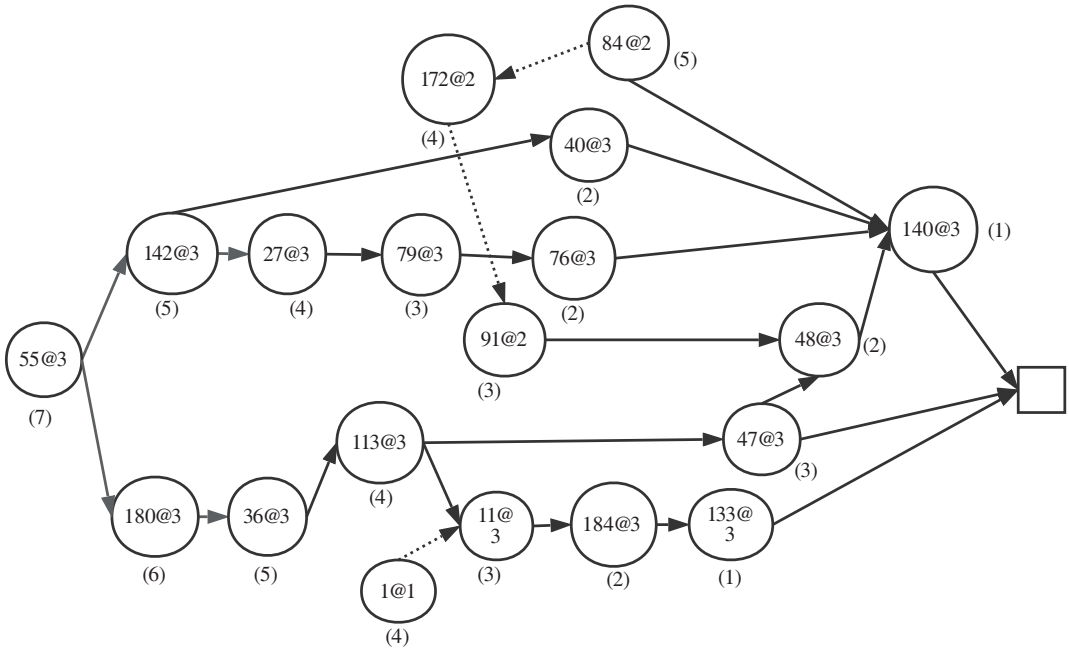

Figure 1 Complete implication graph implying the empty clause from clause $x_{70} \vee \neg x_{24} \vee \neg x_{67}$. Numbers in parentheses below vertices represent the longest distance from the vertex to $\square$.

The above definition also provides a way to construct the complete implication graph of the empty clause from clause c. Note that an implication graph is a directed acyclic graph, where vertices without preceding vertices are decision literals and vertex $\square$ has no successor. The graph depicts how decision literals imply the empty clause.

Figure 1 shows a complete implication graph implying the empty clause $\square$ from clause $140@3 \vee 47@3 \vee 133@3$. Following the terminology of [4], the vertex name $a@b$ means that literal l_a is assigned at decision level b. An even number a represents a positive literal and an odd number a negative literal; thus, $l_a = x_{\frac{a}{2}}$ if a is even and $l_a = \neg x_{\frac{a+1}{2}}$ if a is odd. For example, clause $140@3 \vee 47@3 \vee 133@3$ in Figure 1 is in fact clause $x_{70} \vee \neg x_{24} \vee \neg x_{67}$.

Note that each vertex in an implication graph also identifies a clause. A vertex without any predecessor identifies a unit clause (i.e., a variable with a truth value). For example, the vertex 55@3 identifies the unit clause $\neg x_{28}$. A vertex with predecessor identifies a non-unit clause. For example, the vertex 140@3 identifies, together with the incoming edges, the clause $\neg x_{24} \vee \neg x_{38} \vee \neg x_{20} \vee \neg x_{42} \vee x_{70}$. Note that the set of all clauses in an implication graph is clearly unsatisfiable. It has been stated that this set is minimally unsatisfiable [21] in the sense that, if any clause is removed from the set, then it will become satisfiable.

A literal and its negation cannot both occur in an implication graph. We replace each literal with its variable in the implication graph because the purpose is to determine how to increase the score of a variable by analyzing the implication graph. We say that a variable x contributes to a conflict if there is at least a path from the variable to the conflict in the implication graph, and we say that x depends on clause c to contribute to the conflict if there is a path from x to the conflict in the implication graph containing clause c.

3.2 Distance score based on implication graphs

All variables of a complete implication graph are involved in the derivation of the empty clause and contribute to the conflict. Our purpose is to estimate, from the current conflict, how an involved variable contributes to a future conflict. In fact, for each variable x in a complete implication graph of the empty clause, a CDCL SAT solver can compute the set of clauses on which x depends to contribute to the conflict, which could be considered when computing the score of x under the following hypothesis.

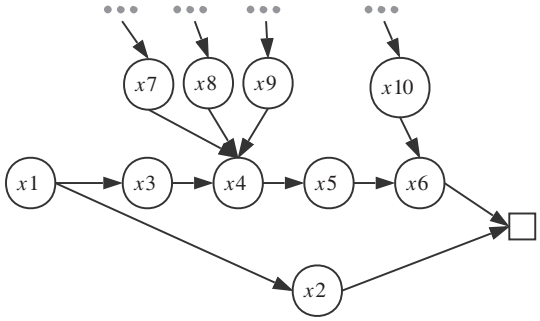

Figure 2 Example to illustrate that, when a variable needs more clauses to imply a conflict, it has lower probability of being involved in a future conflict. For simplicity, each vertex represents a positive literal x_i in the implication graph.

Dependence hypothesis. A variable x that contributes to the current conflict and depends on fewer clauses than variable x' has higher probability of contributing to a future conflict than x'.

The intuition behind the dependence hypothesis can be summarized as follows. The involvement of a clause in unit propagation (i.e., the fact it becomes unit) depends on many factors and is somewhat random. If a variable depends on few clauses to contribute to a conflict, the probability that these few clauses are simultaneously involved in future unit propagations to imply a future conflict is greater than if the variable depends on more clauses. To observe this, we consider two examples. In set $\{x_1, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2\}$, variable x_1 depends only on two clauses to contribute to a conflict. Its probability to contribute to a future conflict is high because a conflict is derived once x_1 is assigned true. However, the probability that variable x_3 in Figure 2 contributes to a future conflict is significantly less because x_3 depends on four clauses to contribute to the conflict. In fact, x_3 does not contribute to the conflict if clause $x_4 \vee \neg x_7 \vee \neg x_8 \vee \neg x_9$ is learned but is removed when the learned clause database is reduced, or when x_{10} is not assigned because one decision literal implying it is not assigned. This holds because the set of clauses of the implication graph is minimally unsatisfiable and becomes satisfiable if any clause is not included in the implication graph (i.e., it does not become unit) for some reason during the search.

The number D of clauses on which a variable depends to contribute to a conflict can be considered a measure of the probability by which the variable contributes to the conflict. Unfortunately, we are unaware of any algorithm that can compute D for all variables in time $O(|C_{\max}| \times n)$, where $|C_{\max}|$ and n are the length of the longest clause and the number of vertices in the implication graph, respectively. Therefore, we construct the complete implication graph and compute the number of vertices in the longest path, denoted longDist$[x]$, for each variable x with a path to the conflict using Algorithm 1. The intuition behind longDist$[x]$ is that a variable depending on many clauses likely requires a long path to reach the conflict. Note that the complexity of Algorithm 1 is $O(|C_{\max}| \times n)$, which is the same complexity as the conflict analysis in CDCL SAT solvers.

The distance score of variable x, denoted distAct$[x]$, based on the dependence hypothesis is defined as follows. The score distAct$[x]$ is initialized to 0. Then, when x contributes to a conflict (i.e., when unit propagation derives the empty clause and x occurs in the corresponding complete implication graph), Algorithm 2 increments distAct(x) by inc$\times 1/$longDist$[x]$, where inc is a global variable initialized to 1 and dist_Decay is intended to assign more importance to recent conflicts, similar to the var_decay parameter in the VSIDS heuristic. The default value of dist_Decay is 0.95, which is the default value of var_decay in MiniSat.

Algorithm 3 shows a solver that selects decision variables in the initialization phase according to the proposed distance score. It employs the distance score in the first α conflicts, where the value of parameter α is fixed empirically. For each of the first α conflicts, the solver calls Algorithm 2 to construct a complete implication graph and update the distance score of each variable in the graph. All other aspects of the solver are unchanged. After the initialization phase, the solver employs its own branching heuristic (e.g., VSIDS or LRB) to select decision variables.

Algorithm 1 computeLongestdistances(C)

Input: C, a clause in which all literals are falsified by the current partial assignment

```
for each variable x in C do
    longDist[x] ← 0; seen[x] ← 1;
end for
for each assigned variable x in the decreasing order of their assigning time do
    if seen[x] == 1 then
        R ← reason[x];
        if R ≠ no_reason then
            for each variable y in clause R do
                if x ≠ y then
                    if seen[y] == 0 then longDist[y] ← longDist[x] + 1, seen[y] ← 1;
                    else if longDist[y] < longDist[x] + 1 then longDist[y] ← longDist[x] + 1;
                    end if
                end if
            end for
        end if
    end if
end for
for each assigned variable x such that seen[x] == 1 do
    seen[x] ← 0;
end for
```

Algorithm 2 updatedistanceScore(C)

Input: C, a clause in which all literals are falsified by the current partial assignment

```
Call Algorithm 1 to construct the complete implication graph G that falsifies C and compute longDist[x] for each variable x occurring in G;
for each variable x occurring in G do
    distAct[x] ← distAct[x] + inc × 1/longDist[x];
end for
inc ← inc/dist_Decay;
```

Algorithm 3 Solver_Dist(α): a CDCL SAT solver using the distance branching heuristic for the first α conflicts

```
dist_Decay ← 0.95; nbConflicts ← 0;
for each variable x do
    distAct[x] ← 0;
end for
while true do
    cl ← unitpropagate();
    if all literals in cl are falsified then
        nbConflicts ← nbConflicts +1; /* a new conflict is derived */
        if the current decision level is 0 (i.e., there is no decision literal) then
            return UNSAT;
        end if
        if nbConflicts ⩽ α then
            updatedistanceScore(cl); /* call Algorithm 2 */
        end if
        Conflict analysis to learn a new clause;
        Backtrack to the second highest level in the new learned clause;
    else
        if nbConflicts ⩽ α then
            x ← the free variable with the greatest distance score distAct[y];
        else
            x ← the free variable selected using the solver's own branching heuristic;
        end if
        Assign x to true or false based on a polarity heuristic such as phase saving;
    end if
end while
```

4 Empirical evaluation

We implemented Algorithm 3 in the Glucose-3.0 [9, 18] and Maple_LCM [10] CDCL SAT solvers under the dependence hypothesis. The resulting solvers are referred to as Glucose-3.0_Dist(α) and Maple_LCM_Dist(α), respectively. In other words, Glucose-3.0_Dist(α) (Maple_LCM_Dist(α)) is Glucose-3.0 (Maple_LCM) with the exception that it uses the distance heuristic under the dependence hypothesis to select decision variables when the number of conflicts (nbConflicts) is not greater than α.

The set of clauses on which a variable x depends to contribute to a conflict or the longest distance of x to the conflict is not the only thing a SAT solver can learn from an implication graph. In fact, Algorithm 1 can be easily adapted to compute the number of vertices in the shortest path, denoted shortDist[x], from x to the conflict in the implication graph, which could be considered to compute the score of x under the following hypothesis.

Nearness hypothesis. A variable closer to the conflict in the implication graph has higher probability of contributing to a future conflict.

To evaluate the nearness hypothesis, we replaced longDist[x] with shortDist[x] in Algorithm 2 to compute the short distance score for each variable. We then implemented Glucose-3.0_shDist(α) and Maple_LCM_shDist(α), i.e., Glucose-3.0 and Maple_LCM, respectively, with the exception that the short distance score is used under the nearness hypothesis to select decision variables when the number of conflicts is not greater than α.

We compiled the above solvers using the Makefile of their base solver, and then solved the instances from the application and hard combinatorial tracks of the 2014 and 2016 SAT competitions, as well as the main track of the 2017 SAT competition. The experiments were performed using Intel Xeon CPUs E5-2680 v4@2.40 GHz under Linux with 20 GB of memory.

Table 1 compares Glucose-3.0, Glucose-3.0_shDist(5×10^4), Glucose-3.0_Dist(5×10^4), Maple_LCM, Maple_LCM_shDist(5×10^4), Maple_LCM_Dist(10^4), Maple_LCM_Dist(2.5×10^4), Maple_LCM_Dist(5×10^4), Maple_LCM_Dist(7.5×10^4) and Maple_LCM_Dist(10×10^4) with a cutoff time of 5000 s for each solver and instance as in the SAT competition. Note that Maple_LCM_Dist(10^4) corresponds to Maple_LCM_Dist, which was the best solver at the 2017 SAT competition. For each solver and group of benchmarks, Table 1 shows the total number of instances (#instances) in each group, the number of (satisfiable + unsatisfiable) instances solved within the cutoff time, and the mean run time of the solved instances, as well as the score of the solver in the main track (application + crafted) of each year, computed as follows. The score is initialized to 0. Then, for each instance in the main track, if the instance is solved within the cutoff time, the score is increased by the solving time in seconds; otherwise, the score is increased by 10^4 (i.e., two times the cutoff time). This score scheme is referred to as PAR-2 and was used in the 2017 SAT competition to rank solvers. Here, the lower the score, the better the solver. The best results are shown in bold in Table 1.

Three observations can be made from the results given in Table 1.

• The distance heuristic based on the dependence hypothesis, given in Algorithm 3 and used in the initialization phase of Glucose-3.0 and Maple_LCM, improves the performance of Glucose-3.0 and Maple_LCM, particularly in terms of the PAR-2 score. The solvers with the distance heuristic outperformed their counterpart base solvers without the distance heuristic in the considered tracks of the 2014, 2016, and 2017 SAT competitions. Maple_LCM_Dist(10^4) significantly outperformed Maple_LCM on the instances of the 2017 SAT competition, and Maple_LCM_Dist(5×10^4) performed better than Maple_LCM_Dist(10^4). Note that Glucose-3.0 and Maple_LCM are highly efficient solvers.

• The Glucose-3.0_shDist(5×10^4) and Maple_LCM_shDist(5×10^4) solvers, which are based on the nearness hypothesis, demonstrated worse performance than Glucose-3.0_Dist(5×10^4) and Maple_LCM_Dist(5×10^4), indicating that a solver should use the longest rather than shortest distance in the complete implication graph to compute the score of a variable.

• The performance of the distance heuristic is not very sensitive to α. In fact, Maple_LCM_Dist(10^4), Maple_LCM_Dist(2.5×10^4), Maple_LCM_Dist(5×10^4), Maple_LCM_Dist(7.5×10^4), and Maple_LCM_Dist(10×10^4) solved more instances than their base solver, i.e., Maple_LCM. Nevertheless, when $\alpha \leqslant$

Table 1 Comparison of different implementations of the proposed distance branching heuristic on instances of different SAT competitions with cutoff time of 5000 s for each solver and instance

		SAT 2014		SAT 2016		SAT 2017	Total
		App.	Crafted	App.	Crafted	Main	
Solver	#Instances	300	300	300	200	350	1450
Glucose-3.0	#solved	229	180	159	58	179	805
	#Sat + #Unsat	102+127	84+96	67+92	5+53	77+102	–
	Mean time	625 s	848 s	771 s	1644 s	1071 s	–
	Score	2205765		3047941		1901709	–
Glucose-3.0_shDist(α) $\alpha = 50000$	#Solved	230	183	157	**60**	171	801
	#Sat + #Unsat	103+127	88+95	64+93	8+52	70+101	–
	Mean time	686 s	834 s	791 s	1740 s	1078 s	–
	Score	2180402		3058587		1974338	–
Glucose-3.0_Dist(α) $\alpha = 50000$	#Solved	226	188	156	**60**	178	**808**
	#Sat + #Unsat	101+125	91+97	64+92	8+52	77+101	–
	Mean time	596 s	936 s	751 s	1458 s	990 s	–
	Score	**2170664**		**3044636**		**1896220**	–
Maple_LCM	#Solved	265	218	175	56	219	933
	#Sat + #Unsat	121+144	93+125	74+101	8+48	103+116	–
	Mean time	1054 s	779 s	1023 s	1619 s	878 s	–
	Score	1619132		2959689		1502282	–
Maple_LCM_shDist(α) $\alpha = 50000$	#Solved	259	220	170	56	222	927
	#Sat + #Unsat	113+146	96+124	72+98	8+48	108+114	–
	Mean time	1000 s	808 s	906 s	1664 s	906 s	–
	Score	1646760		2987204		1481132	–
Maple_LCM_Dist(α) $\alpha = 10000$	#Solved	266	219	174	59	**226**	944
	#Sat + #Unsat	119+147	96+123	72+102	8+51	110+116	–
	Mean time	1075 s	767 s	1050 s	1773 s	889 s	–
	Score	1603923		2957307		1440914	–
Maple_LCM_Dist(α) $\alpha = 25000$	#Solved	265	**223**	173	**60**	225	946
	#Sat + #Unsat	118+147	98+125	74+99	7+53	109+116	–
	Mean time	1028 s	799 s	977 s	1693 s	925 s	–
	Score	1570597		2940601		1458125	–
Maple_LCM_Dist(α) $\alpha = 50000$	#Solved	**272**	220	174	58	225	**949**
	#Sat + #Unsat	125+147	96+124	72+102	7+51	109+116	–
	Mean time	1012 s	758 s	936 s	1620 s	847 s	–
	Score	**1522024**		2936824		1440575	–
Maple_LCM_Dist(α) $\alpha = 75000$	#Solved	263	221	**177**	56	223	940
	#Sat + #Unsat	116+147	97+124	75+102	7+49	109+114	–
	Mean time	1040 s	798 s	984 s	1538 s	817 s	–
	Score	1609878		**2930296**		1452191	–
Maple_LCM_Dist(α) $\alpha = 100000$	#Solved	259	220	**177**	56	223	935
	#Sat + #Unsat	114+145	95+125	73+104	8+48	107+116	–
	Mean time	934 s	732 s	1057 s	1529 s	757 s	–
	Score	1612946		2942713		**1438811**	–

50000, the heuristic appears to give the best results, indicating that when the number of conflicts is greater than 50000, the VSIDS and LRB heuristics in Glucose-3.0 or Maple_LCM already perform well and there is less need to construct a complete implication graph to use the proposed distance heuristic.

Table 2 compares Glucose-3.0, Glucose-3.0_shDist(5×10^4), Glucose-3.0_Dist(5×10^4), Maple_LCM, Maple_LCM_shDist(5×10^4), and Maple_LCM_Dist(5×10^4) with a cutoff time of 5 h for each solver and instance, in the same manner as Table 1. Note that the percentage taken by the first 50000 conflicts in the five-hour search is substantially less than that of 5000 s search. The results in Table 2 confirm

Table 2 Comparison of different implementations of the distance branching heuristic on instances of different SAT competitions with a cutoff time of 5 h for each solver and instance

		SAT 2014		SAT 2016		SAT 2017	Total
		App.	Crafted	App.	Crafted	Main	
Solver	#Instances	300	300	300	200	350	1450
Glucose-3.0	#Solved	244	199	183	77	208	911
	#Sat + #Unsat	108+136	92+107	70+113	7+70	88+120	–
	Mean time	1311 s	1737 s	1977 s	3131 s	2293 s	–
	Score	6317547		9242878		5588944	–
Glucose-3.0_shDist(α) $\alpha = 50000$	#Solved	242	199	183	77	**212**	913
	#Sat + #Unsat	109+133	94+105	71+112	8+69	89+123	–
	Mean time	1156 s	1400 s	2338 s	3030 s	2654 s	–
	Score	6282352		9301164		**5530648**	–
Glucose-3.0_Dist(α) $\alpha = 50000$	#Solved	**246**	**202**	**187**	**82**	208	**925**
	#Sat + #Unsat	112+134	95+107	72+115	9+73	89+119	–
	Mean time	1440 s	1528 s	2449 s	3120 s	2343 s	–
	Score	**6134896**		**9029803**		5599344	–
Maple_LCM	#Solved	277	226	202	82	253	1040
	#Sat + #Unsat	127+150	99+127	78+124	11+71	117+136	–
	Mean time	2332 s	1836 s	2972 s	3952 s	2338 s	–
	Score	4552900		8700408		4083514	–
Maple_LCM_shDist(α) $\alpha = 50000$	#Solved	275	**230**	203	82	253	1043
	#Sat + #Unsat	125+150	102+128	80+123	10+72	117+136	–
	Mean time	2506 s	1964 s	2905 s	4059 s	2270 s	–
	Score	4560870		8662553		4066310	–
Maple_LCM_Dist(α) $\alpha = 50000$	#Solved	**278**	229	**204**	**84**	**254**	**1049**
	#Sat + #Unsat	128+150	102+127	79+125	12+72	118+136	–
	Mean time	2078 s	1874 s	3096 s	4142 s	2262 s	–
	Score	**4354830**		**8611512**		**4030548**	–

the performance of the proposed distance heuristic based on the dependence hypothesis, even when it affects a very small percentage of the search. The performance of Glucose-3.0_shDist(5×10^4) and Maple_LCM_shDist(5×10^4) is somewhat surprising with the five-hour cutoff time because these solvers perform worse than their base solvers when the cutoff time is 5000 s. This phenomenon might be explained as follows. The search is diversified due to the distance heuristic based on the nearness hypothesis, and this diversification is not profitable with a short cutoff time but is profitable with a long cutoff time. Anyway, the results of Table 2 show that using the longest distance to increase the score of a variable during the first 50000 conflicts always performs better than using the shortest distance.

When a CDCL SAT solver derives an empty clause, we say that a variable is in the conflict level if it is assigned after the last decision variable. The conflict analysis of a CDCL SAT solver is typically performed in the conflict level. Here, let n_1 denote the number of variables in the conflict level and let n_2 denote the total number of variables in the completed implication graph. For the 1450 instances tested in this paper, n_2 is approximately equal to $30 \times n_1$ on average. We implemented the distance branching heuristic based on the dependence hypothesis in Maple_LCM_Dist(α) using a partial implication graph with $\min(n_2, k \times n_1)$ vertices for $k = 1, 2, 5, 10, 15$, respectively. The partial implication graph was constructed in a backward manner using Algorithm 1 as the complete implication graph, with the exception that the construction stopped when the limited number of vertices was reached. The resulting solver is denoted Maple_LCM_Dist(α, k). Table 3 compares the results of these solvers with a cutoff time of 5000 s for each solver and instance.

Table 3 shows that the complete implication graph in Maple_LCM_Dist(50000) gives the best results because it solves more instances than Maple_LCM_Dist($50000, k$) for all values of k (except for $k = 10$), and the mean solving time of Maple_LCM_Dist(50000) is less than that of of Maple_LCM_Dist($50000, 10$).

Table 3 Comparison of solvers with complete and partial implication graphs

		SAT 2014		SAT 2016		SAT 2017	Total
		App.	Crafted	App.	Crafted	Main	
Solver	#Instances	300	300	300	200	350	1450
Maple_LCM_Dist(α)	#Solved	**272**	220	174	58	225	**949**
$\alpha = 50000$	#Sat + #Unsat	125+147	96+124	72+102	7+51	109+116	–
Complete graph	Mean time	1012 s	758 s	936 s	1620 s	847 s	**937 s**
Maple_LCM_Dist(α,k)	#Solved	264	220	173	58	**229**	944
$\alpha = 50000$	#Sat + #Unsat	119+145	95+125	73+100	8+50	114+115	–
$k = 1$	Mean time	990 s	743 s	872 s	1614 s	863 s	–
Maple_LCM_Dist(α,k)	#Solved	266	219	177	58	223	943
$\alpha = 50000$	#Sat + #Unsat	121+145	94+125	76+101	8+50	108+115	–
$k = 2$	Mean time	1004 s	779 s	1044 s	1633 s	886 s	–
Maple_LCM_Dist(α,k)	#Solved	267	218	**178**	54	224	941
$\alpha = 50000$	#Sat + #Unsat	123+144	94+124	75+103	7+47	109+115	–
$k = 5$	Mean time	1022 s	758 s	1038 s	1593 s	912 s	–
Maple_LCM_Dist(α,k)	#Solved	265	**221**	**178**	**60**	225	**949**
$\alpha = 50000$	#Sat + #Unsat	122+143	96+125	74+104	9+51	107+118	–
$k = 10$	Mean time	964 s	731 s	1007 s	1692 s	955 s	962 s
Maple_LCM_Dist(α,k)	#Solved	258	219	**178**	57	227	939
$\alpha = 50000$	#Sat + #Unsat	114+144	94+125	74+104	7+50	111+116	–
$k = 15$	Mean time	961 s	775 s	1049 s	1633 s	848 s	–

Moreover, Maple_LCM_Dist(50000) does not need to adjust the value of k.

5 Conclusion

State-of-the-art CDCL SAT solvers typically compute the score of a variable based on its simple presence in previous conflicts. When there are few previous conflicts, as in an initialization phase of the search, a CDCL SAT solver must learn more from the implication graph of a conflict to maintain the score of a variable. In this paper, we tested two hypotheses to design a branching heuristic for CDCL SAT solvers in the initialization phase: (1) The dependence hypothesis stating that a variable depending on fewer clauses to contribute to the current conflict has higher probability of contributing to a future conflict; and (2) the nearness hypothesis stating that a variable closer to the current conflict in the implication graph has higher probability of contributing to a future conflict. When an empty clause is derived, the dependence hypothesis uses the longest distance of variable x to the conflict in the complete implication graph to update the score of x, whereas the nearness hypothesis uses the shortest distance of x to the conflict. We refer to the heuristic based on the dependence hypothesis as the distance branching heuristic.

We conducted experiments on 1450 instances of the main tracks of three recent SAT competitions. The results demonstrate that using the proposed distance branching heuristic in the initialization phase significantly improves the performance of two of the best CDCL SAT solvers.

It is important to highlight that Maple_LCM_Dist, which is a preliminary implementation of the distance branching heuristic in the Maple_LCM solver, was ranked first in the main track of the 2017 SAT Competition, and the new Maple_LCM_Dist(5×10^4) solver further improves the performance of Maple_LCM_Dist. Therefore, the gains obtained due to the distance branching heuristic are important, and the distance branching heuristic offers a significant contribution to CDCL SAT solvers.

Acknowledgements This work was partially supported by National Natural Science Foundation of China (Grant Nos. 61370183, 61370184, 61472147), Matrics Platform of the Université de Picardie Jules Verne, and MINECO-FEDER Project RASO (Grant No. TIN2015-71799-C2-1-P).

References

1 Audemard G, Simon L. Glucose 2.3 in the SAT 2013 competition. In: Proceedings of SAT Competition, 2013. 42–43
2 Biere A. Lingeling, Plingeling, Picosat and Precosat at SAT Race 2010. FMV Report Series, Technical Report 10/1, 2010
3 Eén N, Sorensson N. An extensible SAT solver. In: Proceedings of SAT, 2003. 502–518
4 Marques-Silva J, Sakallah K A. GRASP — a new search algorithm for satisfiability. In: Proceedings of the International Conference on Computer-Aided Design, 1996. 220–227
5 Moskewicz M, Madigan C, Zhao Y, et al. Chaff: engineering an efficient SAT solver. In: Proceedings of the Design Automation Conference, 2001. 530–535
6 Soos M. CryptoMiniSat v4. In: Proceedings of SAT Competition, 2014. 23
7 Marques-Silva J, Lynce I, Malik S. Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability. Washington: IOS Press, 2009. 131–153
8 Liang J H, Ganesh V, Poupart P, et al. Exponential recency weighted average branching heuristic for SAT solvers. In: Proceedings of AAAI, 2016. 3434–3440
9 Audemard G, Simon L. Predicting learnt clauses quality in modern SAT solvers. In: Proceedings of IJCAI, 2009. 399–404
10 Luo M, Li C M, Xiao F, et al. An effective learnt clause minimization approach for CDCL SAT solvers. In: Proceedings of IJCAI, 2017. 703–711
11 Xiao F, Luo M, Li C M, et al. MapleLRB LCM, Maple_LCM, Maple_LCM_Dist, MapleLRB_LCMoccRestart and Glucose-3.0+width in SAT Competition 2017. In: Proceedings of SAT Competition, 2017. 22–23
12 Audemard G, Simon L. Refining restarts strategies for SAT and UNSAT. In: Proceedings of International Conference on Principles and Practice of Constraint Programming, 2012. 118–126
13 Biere A, Fröhlich A. Evaluating CDCL variable scoring schemes. In: Proceedings of SAT, 2015. 405–422
14 Goldberg E, Novikov Y. BerkMin: a fast and robust Sat-solver. Discrete Appl Math, 2007, 155: 1549–1561
15 Jeroslow R G, Wang J. Solving propositional satisfiability problems. Ann Math Artif Intell, 1990, 1: 167–187
16 Marques-Silva J. The impact of branching heuristics in propositional satisfiability algorithms. In: Proceedings of EPIA, 1999. 850
17 Audemard G, Simon L. GLUCOSE: a solver that predicts learnt clauses quality. In: Proceedings of SAT Competition, 2009. 7–8
18 Audemard G, Simon L. Glucose in the SAT 2014 competition. In: Proceedings of SAT Competition, 2014. 31–32
19 Liang J H, Ganesh V, Poupart P, et al. Learning rate based branching heuristic for SAT solvers. In: Proceedings of SAT, 2016. 123–140
20 Liang J H, Oh C, Ganesh V, et al. MapleCOMSPS, MapleCOMSPS LRB, MapleCOMSPS CHB. In: Proceedings of SAT Competition, 2016. 52–53
21 Li C M, Manyá F, Mohamedou N O, et al. Resolution-based lower bounds in MaxSAT. Constraints, 2010, 15: 456–484