An Effective Sharpsat Solution Method Based on Learnt Clause Minimization Approach

Xi Chen\textsuperscript{a}, Weikang Du\textsuperscript{b}, Yingjie Xu\textsuperscript{c}, Mingdian Ma\textsuperscript{d}

Department of Software, Jilin University, Changchun, China

\textsuperscript{a}2114118585@qq.com, \textsuperscript{b}592262731@qq.com, \textsuperscript{c}408054109@qq.com, \\
\textsuperscript{d}460937922@qq.com

Abstract. The problem of satisfiability of the propositional logic formula (SAT problem) in computer science is an important and difficult problem. Learning clauses often contain redundant literal in CDCL SAT solving, which may have a negative impact on the performance of the solver. To overcome this shortcoming, we show a new sharpSAT solver based on learning clause minimization is proposed. By recoding the CNF formula to reduce the storage space, and applying Boolean constraint propagation to eliminate redundant literals in the learning clause, the algorithm reduces the time cost. The complexity is compared with the existing solver. The experimental results show that the solver has significant application value, reduces the time of instance solution, and increases the number of maximum solvable instances.

1. Introduction

The problem of satisfiability of the propositional logic formula (SAT problem) \cite{1} is a question of determining whether a Conjunctive Normal Form has a satisfactory assignment. It is the first proven NP-Complete problem \cite{2} in history and a core issue in computer science and engineering and artificial intelligence research. Therefore, it is extremely important to find a fast and effective algorithm for solving SAT problems not only in theoretical research but also in many application fields. However, there are many other classic problems in the field of artificial intelligence research, such as Bayesian network inference problem, probability programming problem, confidence analysis problem, etc. Their computational complexity is higher than NP. After turning these problems into propositional formulas, it is impossible to solve the problem simply by judging the satisfiability of the formula, and the number of these assignments (i.e., models) that the mission-solving formula can satisfy is the key to solving the problem. This extends the #SAT problem, which is the problem of solving the number of models for a given propositional formula. The computational complexity of the model counting problem is \#P-complete. For other \#P problems, the problem can be solved in the polynomial time. Therefore, the research on the #SAT problem solving method has been done in the field of artificial intelligence research. Quite far-reaching theoretical and practical significance.

#SAT, also known as the model counting problem, is the problem of solving the model of a given propositional formula (generally, this mission-solving formula is a model of true assignment and called a propositional formula). The #SAT problem requires finding out the number of models that can satisfy the problem, so the computational complexity of the #SAT problem is \#p complete, the computational
complexity of the problem is higher than the proposition can satisfy the problem, and the proposition can satisfy the problem only needs to find out a model that fits the problem is fine.

The characteristic of the complete algorithm is that when the problem is satisfied, the assignment (i.e. the model) that a mission problem formula can satisfy is given; when it is not satisfied, the proof is given. Prior to this, the complete algorithm based on Davis Putnam Lagemann Loveland Algorithm (DPLL) [3] was the main direction. At present, the mainstream SAT solver based on the complete algorithm for solving SAT problem is Conflict Driven Clause Learning (CDCL) SAT solver. However, the learning clause in the CDCL SAT solver usually contains redundant literal. Redundant literal may reduce the effectiveness of Boolean Constraint Propagation (BCP) and the quality of subsequent learning clauses, which may have a negative impact on performance. At this time, the learning clause minimization algorithm came into being. The learning clause is minimized before the SAT solver triggers certain selected restarts, applying BCP to eliminate redundant literal in the learning clause. Related experiments have proved the efficiency of the learning clause minimization algorithm and have been widely used. But the #SAT problem is a new field and the main content of this paper. Reducing the redundant literal in the learning clause plays a crucial role in improving the efficiency of the solver.

2. Preliminaries

Let \( Q = \{ x_1, \ldots, x_n \} \) be a set of propositional variables. The literal is positive \( x \) or its negation \( \neg x \). A clause is the disjunction of the literals. The size of a clause is the number of literals it contains. A unit clause is a clause that contains only one literal. The Conjunctive Normal Form (CNF) formula is the conjunction of the clauses. A CNF formula consists of a series of clauses, and a clause consists of a series of literals.

Suppose there is a set of assignments \( \rho: Q = \{0, 1\} \). The assignment satisfies the negation of the literal \( x \) or \( \neg x \) if and only if \( \rho(x) = 1 \) or \( \rho(x) = 0 \); the set of values satisfies a clause if and only if the group is assigned satisfying any of the literals in the sentence, the set assignment satisfies a CNF formula if and only if the set assignment satisfies all clauses in the CNF formula. We say that a CNF formula is achievable if and only if there is at least one set of assignments that satisfy the CNF formula. An empty clause is always unsatisfiable or represents a conflict or failure. The SAT question is to prove whether a CNF formula is satisfactory. The two CNF formulas are equivalent if and only if only two formulas can be satisfied by the same set of values. If a new CNF formula obtained by deleting a literal \( l \) from a clause of a CNF formula is equivalent to the original CNF formula, the literal \( l \) is said to be redundant.

Given a CNF formula \( F \) and a literal \( l \), we defined \( F|_l \) is removing all clauses which contain \( l \) from formula \( F \) and all the literals that appear in the clause \( l \). The CNF formula obtained later. First define:

1. An empty clause is always unsatisfiable, and it indicates a conflict or failure.
2. In clause \( c \) of formula \( f \), if deleted from \( c \) will cause the formula to be logically equivalent to \( f \), then the literal it is superfluous.

If wanting to recursively define Boolean constraint propagation (BCP), or more precisely, Unit Propagation (UP), use the following two rules:

1. \( UP(F) = F \) when \( f \) does not contain any unit clauses;
2. \( UP(F) = UP(F|_l) \) When there is a unit clause is \( \{l\} \) in \( F \).

In the latter case, we say literal it is asserted because it must be assigned a value of 1. If the unit clause \( \{l\} \) is obtained by applying rule (2), then \( UP(F) \) will infer or assert literal \( l \). When calculating \( UP(F \cup \{l\}) \), we say that \( l \) spreads in \( f \).

We use \( UP(F) \) to denote the reuse rule (2) until we get an empty clause or a CNF formula obtained after there is no unit clause in \( f \). Specifically, if we can use the rule (2) multiple times to get an empty clause, then \( UP(F) = \square \). Otherwise, \( UP(F) \) indicates that all unit clauses have been used (2) the CNF formula after propagation.

The CDCL solver performs a non-time backtracking search in a partial truth value assignment space. Specifically, the solver repeatedly selects a decision variable \( l \) and application unit propagation \( UP(F \cup \{l_i, l_2, \ldots, l_j\}) \) until the formula is empty or empty clause. If you derive an empty clause, you use a special method (usually the only implication point) to get the learning clause, and then add the learning clause to the clause database.
Suppose we have learned some clauses through conflicts, but the excessive number of literals in each learning clause will affect the efficiency of the solver. Our main goal is to design and implement an efficient and efficient processing method to minimize the use of single-literal rules.

The literal in the learning clause can be divided by the number of layers declared by the variable. The variables declared in the same layer are divided into one class. The number of these classes is called the Literal Block Distance (LBD). Experiments show that the smaller the value of LBD (Literal Block Distance, LBD), the higher the efficiency of using single-literal rules. We can delete some unnecessary literals by some algorithm to achieve the purpose of simplifying the learning clause.

3. Related Work on Clause Minimization and SharpSAT Solver

Davis and Putnam first proposed the DP algorithm. In 1962, G Logemann and D Loveland further described the DP algorithm, and used Loveland's problem subdivision instead of decomposition to form the DPLL algorithm. Many of the later algorithms were improved on this basis. For example, rel SAT [4], GRASP [5] and SATO [6] added intelligent backtracking search strategy, restart technique and conflict clause analysis. There are of course other backtracking SAT algorithms that are competitive with specific classes of SAT, such as SATZ [7], POSIT [8], NTAB [9], and CSAT [10].

The Chaff [11] solver proposed in 2001 has the significance of mileage cards. The main technology introduced is the two monitoring literal data structures, which greatly optimize Boolean Constraint Propagation (BCP). The conflict clause drives the search process, and the VSIDS decision heuristic takes into account the information of the recent conflict clause. Therefore, this decision strategy can be regarded as directly combining this driving force with the decision process. The Berkmin [12] solver, introduced in 2002, inherits the GRAP, SATO and Chaff solver clause records, fast BCP, restart techniques and conflict clause deletion techniques, and introduces a new list-based decision process and new clause database management method. Minisat [13] proposed in 2003 is the basis of glucose, which mainly introduces conflict-driven clause learning and dynamic variable ordering. The glucose [14-15] solver proposed by Gilles Audemard and Laurent Simon in 2009 is an enhancement of the Minisat version. They use the Literals Blocks Distance (LBD) to predict the quality of the learning clause, delete the learning clause, and solve the glucose solution. The device is more efficient than other complete solvers.

From 2011 to 2017, Jingchao Chen proposed MPhase SAT [16], improved Glucose [17], Glue bit [18], Glu vc [19] and other solvers, mainly to study the assignment of decision variables, introducing different kinds of phase selection heuristics.

The most widely used method of precision algorithms is still based on methods. The method firstly extends the algorithm into the algorithm, and obtains the prototype-algorithm of the algorithm based on it. The main idea of the algorithm is that in the process of branching, when selecting a variable from the formula to branch, we set the number of models representing the given formula, indicating the number of models of the formula obtained by assigning the variable to true, indicating that the variable is assigned the number of models of the formula obtained after the false value, because all the intersections of the two formulas that satisfy the assignment set are empty, so they are available. According to this formula, by continuing the recursive backtracking, the number of acceptable values of the entire formula can be obtained. In order to improve the efficiency of the solution, strategies such as clause learning and component caching can also be added to the method. There is also an accurate algorithm. When the number of unsatisfiable assignments of the propositional formula is relatively small, the algorithm that is very efficient is called the extended rule method. The basic idea of the method is to extend all the clauses in the formula according to the extension rule, and then using the principle of tolerance and repulsion, the number of unsatisfiable assignments of the extended propositional formula is calculated, so that the inverse solution can satisfy the number of assignments.
4. Learning Clause Minimization in SharpSAT Solver

First consider the encoding process. The sharpSAT solver uses bounded component analysis and caching techniques. Each variable in the Conjunctive Normal Form is represented by a vertex, and then the variables appearing in the same clause are connected by edges to form a no constraint diagram of the direction. Each disconnected branch in the constraint graph is called a connected component. During the construction process, the positive and negative literals are not distinguished.

When assigning a variable to a formula, a conflict may occur. A new clause is obtained through conflict analysis, and the clause is classified into the original formula. This clause is called a learning clause. If the learning clause obtained through conflict analysis contains too much literal, it will affect the efficiency of the solver. At this time, we can minimize the learning clause by algorithm 4 and delete a part of the literal to minimize the use of single-literal rules.

In the process of learning clause minimization, there are several problems to be solved:

1. When is the activation learning clause minimized?
   Activated before restart, but not activated before each restart. Since the number of new learning clauses at each restart may not be sufficient for cell propagation to infer conflicts. Therefore, in order to determine whether the learning clause must be minimized at the time of restart, a function is defined, which indicates the number of learning clauses after the last execution of the learning clause minimization function, and refers to the number of times the function has been executed so far.

2. What kind of learning clause should be minimized?
   Previous experiments have shown that learning clauses with higher LBD have little effect on solving the sat problem. And reducing the LBD of a clause is much more difficult than reducing its length. At the same time, the high LBD learning clause is minimized and usually requires more unit spread. Therefore, we should try to reduce the learning clause with a lower LBD value. More precisely, we define functions to determine if the learning clause is to be minimized according to its LBD.

3. What is the best order of literal propagation in a learning clause during the process of learning clause minimization?
   The minimization of clauses based on unit propagation depends on the order in which the selected literal is selected. In the process of minimizing clauses, we need to know the best order in which the literals in the learning clause are propagated. So we define a sorting function that sorts the clauses before the learning clause is minimized. The function returns without changing the order of the literal in the clause.

Algorithm 1: minimize (R, S)

**Input:** R: A CNF formula; S: a series of learnt clauses such that \( L \cap F = \phi \)

**Output:** A series of learnt clauses

```plaintext
begin
for each \( F = l_1 \lor l_2 \lor \ldots \lor l_k \in S \) do
  if ! liveClause(F) then continue;
  \( L \leftarrow L \setminus \{ F \} \); \( F \leftarrow \text{sort}(F) \);
  \( F' \leftarrow \phi \);
  for \( i = 1 \) to \( k \) do
    if \( (R \leftarrow \text{UP}(R \cup S \cup \neg\bar{F'} \cup \{ \bar{l}_i \})) = \square \) then
      \( F' \leftarrow \text{conflAnalysis}(R, S, \neg\bar{F'} \cup \{ \bar{l}_i \}, C) \);
      break;
    else if \( \text{UP}(R \cup S \cup \neg\bar{F'} \cup \{ \bar{l}_i \}) \neq \square \) then
      \( S \leftarrow S \cup \{ F' \} \);
  return S;
end
```
Let $F = l_1 \lor l_2 \lor \ldots \lor l_k$ is a learning clause in $L$, and makes liveClause($F$) true. If $UP(R \cup \{\check{l}_1, \check{l}_2, \ldots, \check{l}_{i-1}, \check{l}_i\}) = \Box$, then $R \cup \{\check{l}_1, \check{l}_2, \ldots, \check{l}_{i-1}, \check{l}_i\}$ is unsatisfiable. Clause $l_1 \lor l_2 \lor \ldots \lor l_i$ is the logical result for $R$, which is included in $F$, can be added to $R$ to affect the satisfiability of the CNF formula. If $UP(R \cup \{\check{l}_1, \check{l}_2, \ldots, \check{l}_{i-1}, \check{l}_i\}) = \Box$, then $F'' = l_1 \lor l_2 \lor \ldots \lor l_{i-1} \lor \check{l}_i$ can be derived from $R$. In addition, $F = l_1 \lor l_2 \lor \ldots \lor l_{i-1} \lor \check{l}_i$ and the decomposition of $C''$ does not contain $l_i$ but included in $F$. So algorithm 1 does not add $l_i$ to new clause $C''$. And for efficiency reasons, the 10th line of Algorithm 1 calculate the $UP(R \cup S \cup \check{F} \cup \{\check{l}_i\})$ by examining whether $UP(R \cup S \cup \check{F})$ can infer $\check{l}_i$ instead of spreading $l_i$.

Let $\check{F} = \{\check{t}_1, \check{t}_2, \ldots, \check{t}_{i-1}\}$ and function conflAnalysis($R, S, \check{F} \cup \{\check{l}_i\}, R$) is from the conflict clause $C$ obtained by the unit to trace back the hidden graph, returning the analysis of disjunction of the negation of the literal. Pay attention to the function conflAnalysis($R, S, \check{F} \cup \{\check{l}_i\}, C$) does not use the first implication point. Line 7 of Algorithm 1 applies $UP(R \cup S \cup \check{F} \cup \{\check{l}_i\})$ makes $\check{l}_i$ be propagated by the returned formula of $UP(R \cup S \cup \check{F})$, before which needs checking if $\check{l}_i$ or $l_i$ can be asserted. Notify that $UP(R \cup S \cup \check{F} \cup \{\check{l}_i\})$ is implemented using the unit propagation function and the data structure of the solver.

5. Experimental Investigation

In order to test the new solver in the solution time and efficiency is better, randomly select different examples in the SAT competition library for test description. This paper mainly based on the solver Glucose3.0 version of the experimental test.

In the previous study, the solution process based on the CDCL algorithm can be formalization, which is the process of adding the learning and learning clause strategy. Based on this, a learning clause evaluation method based on deductive length is proposed and combined with the existing literal block distance based evaluation method, including four separate uses. Different methods of improved solver [21]. The test solver version is 5, Glucose_LBD, Glucose_length, Glucose_LBD+len, Glucose_len+LBD, and our solver. The same example was used to compare several different solvers and the solvers mentioned in this article. The test examples used in the experiment were from 300 Main-track group instances of the 2015 SAT competition and Main- of the 2016 SAT competition. Track

An instance of 300 of the Application type of the group. These examples come from different practical issues, such as software testing, hardware circuit testing, graph coloring

Problems, network security, etc. The solution time for each instance in the experiment is no more than 4 000 s.

It can be seen from Table 1 that from the comparison of these solvers, the proposed performance of the sharpSAT solver based on the learning clause minimization method is better than the existing solver, and the number of solutions can be solved. And the solution time has certain advantages.

Figure 1 shows the solver Glucose_LBD, Glucose_length, Glucose_LBD+len, Glucose_len+LBD, and the solver proposed in this paper solve the runtime comparison of 600 instances. The closer the curve in Figure 1 is to the right and closer to the x-axis, the smaller the solution time represented by this curve and the more solved. It can be seen that our solver is more efficient.
Figure 1 performance of different solvers

Table 1 Number of solution examples for different

| Test case | category | Glucose_LBD | Glucose_length | Glucose_LBD + len | Glucose_len + LBD | Our Solver |
|-----------|----------|-------------|----------------|-------------------|-------------------|------------|
| SAT 2015(300) | SAT      | 132         | 135            | 140               | 142               | 143        |
|           | UNSAT    | 98          | 99             | 94                | 95                | 96         |
| SAT 2016(300) | SAT      | 54          | 54             | 55                | 57                | 60         |
|           | UNSAT    | 78          | 81             | 82                | 84                | 85         |
| Total (600)   | SAT      | 186         | 189            | 195               | 199               | 203        |
|           | UNSAT    | 176         | 180            | 176               | 179               | 181        |
| sum               |          | 362         | 369            | 371               | 378               | 384        |

6. Conclusion
We describe a new method for processing learning clause minimization based on the application of Boolean constraint propagation to solve the SAT problem and apply it to the sharpSAT solver. When restarting, the number of learning clauses is changed. Cut back. Thus a new sharpSAT solver is defined, which is simpler and has improved performance. We use a lot of data to test the performance of the solver, and the test results show that the solver performs better than the previous solver. We believe that this technology can be used in the future.

Acknowledgments
We would like to extend our sincere gratitude to our tutors, for their instructive advice and useful suggestion on our thesis. We are deeply grateful of their help in the completion of this thesis.

We are also deeply indebted to all the other tutors and teachers in Translation Studies for their direct and indirect help to us.

References
[1] Biere A, Heule M, van Maaren H, et al. Handbook of satisfiability [M]. Washington D C: IOS Press, 2009: 131-153
[2] Cook S A. The complexity of theorem-proving procedures [C] //Proceedings of the 3rd Annual ACM Symposium on Theory of Computing. New York: ACM Press, 1971: 151-158
[3] Davis M, Logemann G, Loveland D. A machine program for theorem-proving [J]. Communications of the ACM, 1962, 5(7): 394-397
[4] Roberto J. Bayardo, Jr, Schrag R C. Using CSP look-back techniques to solve real-world T instances [C]. In Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence(AAAI'97/IAAI'97). AAAI Press, 1997: 203-208.
[5] Marques-Silva J P, Sakallah K A. GRASP: a search algorithm for propositional satisfiability [J]. Computers IEEE Transactions on, 1999, 48(5): 506-521.
[6] Zhang H. SATO: International Conference efficient propositional prover [C],Proceedings of the Automated Deduction, 1997: 272-275.
[7] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems [J].Lecture Notes in Computer Science, 1997, 1330: 341-355.
[8] Freeman J W. Improvements to propositional satisfiability search algorithms [M].University of Pennsylvania, 1995.
[9] Crawford J M, Auton L D. Experimental results on the crossover point in satisfiability problems [C]. Eleventh National Conference on Artificial Intelligence. AAAI Press,1993:21-27.
[10] Dubois O, Andre P, Boufkhad Y, et al. SAT versus UNSAT [J]. Dimacs Challenge workshop.1996.
[11] Madigan C F, Madigan C F, Zhao Y, et al. Chaff: engineering an efficient SAT olver [C].Design Automation Conference. ACM, 2001: 530-535.
[12] Goldberg E, Novikov Y. Berk Min: A Fast and Robust Sat-Solver [C]. Conference on design, Automation and Test in Europe. IEEE Computer Society, 2002: 142.
[13] Eén N, Sörensson N. An Extensible SAT-solver [M]. Theory and Applications of Satisfiability Testing. Springer Berlin Heidelberg, 2003: 502-518.
[14] Audemard G, Simon L. GLUCOSE: a solver that predicts learnt clauses quality [J]. SAT Competition.2009: 7-8.
[15] Audemard G, Simon L. Predicting Learnt Clauses Quality in Modern SAT Solvers.[J].Proc. IJCAI-2009, 2009, 38(4): 399-404.
[16] Chen J. Phase Selection Heuristics for Satisfiability Solvers [J]. ar Xiv preprintar Xiv: 1106.2011, 1372: 1-12.
[17] Chen J. A Dynamic Phase Selection Strategy for Satisfiability Solvers [J]. ar Xiv preprintar Xiv: 1208. 2012, 1613: 1-9.
[18] Chen J. A Bit-Encoding Phase Selection Strategy for Satisfiability Solvers [C].International Conference on Theory and Applications of Models of Computation. Springer, Cham, 2014: 158-167.
[19] Chen J. Glu_vc: Hacking Glucose by Weighted Variable State Independent Decay Sum Branching Policy [J]. Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions, volume B-2017-1 of Department of Computer Science Series of Publications .B, University of Helsinki 2017.
[20] Gu J. Global Optimization for Satisfiability (SAT) Problem [J]. Knowledge & Data Engineering IEEE Transactions on, 1994, 6(3): 361-381.
[21] CHANG Wenjing, XU Yang, WU Guanfeng. Learned clauses reduction strategy based on length of deduction. Computer Engineering and Applications, 2018, 54(16): 30-36