An Efficient Greedy Local Search Algorithm for Boolean Satisfiability Based on Extension Rules

Huanhuan Peng, Liming Zhang and Ziming Ye

College of Software, Jilin University, Changchun 130012, China

Keywords: Boolean Satisfiability (SAT) problem, local search, extension rule, subScore, clause weighting

Abstract: The extension rule is a method for solving Boolean Satisfiability (SAT) problems by using maximum terms, but it is not mature enough at present. Many efficient local search algorithms apply true value assignment search solutions. Based on the relationship between extension rules and truth value assignment and some efficient heuristic strategies, this paper mainly does the following three aspects: 1) analyze two different methods of extension rule and truth value assignment and compare them. On this basis, a local search algorithm based on extension rules (LSER) is proposed. 2) Apply configuration checking strategy, propose the greedy search algorithm method (GSER) of extension rules. 3) We also improve the GSER by the subScore strategy and the clause weighting strategy, and design a new strategy called maximum score upper limit strategy. And then a new extension rule method (IGSER) based on greedy local search is proposed. Experiments show that the proposed algorithm outperforms the general extension rule inference method and algorithm GSER on 3-SAT and CBS SAT instances.

1. Introduction

SAT is the first proved NP-complete problem. And almost all NP-complete problems from a variety of domains can be transformed into SAT problem. The SAT problem is now widely used in the field of artificial intelligence, and many efficient solvers have emerged.

In 1992 Selman proposed the greedy local search algorithm GSAT [1] for the SAT problem. GSAT algorithm proves that the random local search algorithm can effectively solve the SAT problem and becomes the groundbreaking algorithm of the random local search algorithm.

Cai et al. presented configuration strategy in the field of local search algorithm in recent years. This strategy is one of the most influential strategies proposed by Chinese in the international SAT competition [2-7]. In 2011, Cai et al. used configuration strategy for the local search for the SAT problem for the first time, and the designed SWCC achieved amazing results [2]. Subsequently, based on the strategy of configuration, Cai et al. successively designed many powerful solvers such as SWCCA [3], CCASat [4], CScoreSAT [5], CCAnr [6] and CSCCSat [7]. The configuration strategy has greatly improved the performance of the SAT local search algorithm.

In 2003, Lin et al. proposed a new rule of automatic reasoning, called extension rule [8]. Martin Davis, an expert in artificial intelligence, regards it as a "complementary" reasoning method, which shows that this method has been widely recognized all over the world.

Yang et al. proposed a new extension rule reasoning method based on local search, applying local search to extension rules, and constructing an incomplete reasoning framework based on extension rules for the first time [9]. On this basis, this paper proposes LSER, GSER, IGSER algorithm by using some existing local search strategies and some new strategies, hoping to further improve the efficiency of SAT problem.

2. Preliminaries

In this section, we will introduce the basic concepts, including SAT problem, extension rule, etc.

2.1 Basic concept of the SAT problem

Definition 1 (CNF formula): Give a set of n Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$, a literal is either a Boolean variable x or its negation ¬x. A clause is a disjunction of literals. A Conjunctive Normal Form (CNF) formula $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ is a conjunction of clauses. For a literal $v \in X$, we use $ref(C_i, v)$ to represent the truth value of variable v in C_i.

Definition 2 (SAT problem): The Boolean satisfiability problem (SAT) is to find a set of truth assignment so that all clauses in a CNF formula F can be satisfied.

2.2 Extension rule

Definition 3 (Extension rule) [8]: Given a clause C and a variable set X, $D = \{C \vee x, C \vee \neg x\}$ where x is a variable that doesn't appear in C and $x \in X$. At this time we call the operation proceeding from C to D is using the extension rule on C. D is the result of using the extension rule on C.

Obviously, the clause C is logically equivalent to the result D in truth assignment.

2.3 Comparison of extension rule with truth assignment

Applying extension rule to solve the SAT problem is to find a non-expandable maximum term for all clauses, that is, to find a truth value assignment, so that the truth value assignment has at least one variable with different values for each clause. Using truth value assignment to determine whether the SAT problem is satisfiable is to find a true value assignment with at least one variable having the same value for each clause. We can find that the two thoughts are just opposite but equivalent. Therefore, many heuristic strategies applied in the truth assignment algorithm can also be applied to the extension rule.

2.4 Local Search

Although the local search algorithm has many shortcomings in principle, such as the possibility of falling into the local best, it is still simple and effective.

Applying the idea of classical local search algorithm, Yang proposed a framework of local search algorithm based on extension rules, as follows [9]:

Algorithm Local Search in Extension Rule (LSER) [9]

Input: CNF formula $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$
maxTries

Output: Non-expandable maximum term $T = \{x_1, x_2, \ldots, x_n\}$
or no solution found

```
1.attempt ← 0
2.T ← randomly generated maximum term
3.WHILE attempt < maxTries DO
4.    IF CanBeExpand(T, F)
5.    THEN T ← Transform(T)
6.        attempt++
7.    ELSE
8.        RETURN T
9.RETURN "no solution found"
```

Among them, the function "Transform" is the neighborhood conversion function of local search, and the neighborhood is 1 neighborhood, that is, only two maximum terms with one different value of variables are adjacent; the objective function is the number of non-expandable clauses under the value of the current maximum term. If the objective function takes the value m (the number of clauses), then the maximum term T is directly returned.

2.5 Configuration checking [4]

Definition 4 (configuration)[4]: Given a CNF formula F and an truth assignment α, the configuration of a variable x under α is a vector called $Conf_\alpha(x)$,which contains a set of truth values of all variables in N(x) under α, i.e., $Conf_\alpha(x) = \alpha|_{N(x)}$, where N(x) is the set of neighboring variables of x.

Applying the concept of configuration to an extension rule, you only need to think of assignment α as a maximum term T.

Definition 5 (configuration checking strategy) [4]: For a SAT local search algorithm solving a CNF formula F, if the configuration of the variable x has not been changed (which means none of its neighboring variables has been flipped) since it was last selected to flip, then it isn't allowed to be flipped. The above is called configuration checking strategy.

In the following description, we use conf array to represent a change in the configuration of a variable. If the configuration of variable x has been changed, we use conf [x] =1 to indicate it, otherwise set conf[x] to 0 [4].

The algorithm first initializes the whole conf array to 1 for each variable. In every local search process, when the variable x is selected to flip, conf[x] is reset to 0, and for each variable $y \in N(x)$, conf[y] is set to 1[4].

In our proposed algorithm, this strategy is not strictly followed. If the configuration of x has not been changed, then we use some other strategies to select the variable that will be flipped.

3. Greedy Local Search and Improvement

In this part, we will introduce greedy strategies and propose a greedy local search algorithm based on extension rule and improve it.

3.1 Greedy search algorithm under extension rule

In the algorithm LSER, we don't give a specific meaning of the Transform function. In fact, it's related to the score of each variable (which is given below) [9]:

For a formula F and a maximum term T, $\mathrm{cost}(F, T)$ represents the total weight of extensible clauses under maximum term T. The evaluation function of each variable is defined as $\mathrm{Score}(x) = \mathrm{cost}(F, T) - \mathrm{cost}(F, T')$, where T' is obtained from T by flipping x [9]. In the local search process, we try to select variables with highest scores and satisfying conf [v] =1.

Based on the evaluation function and the configuration checking strategy, it is easy to define the local search algorithm in greedy mode. The algorithm is as follows:

Algorithm Greedy Search under Extension Rule (GSER) [9]

Input: CNF formula $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$
maxTries

Output: Non-expandable maximum term $T = \{x_1, x_2, \ldots, x_n\}$ or no solution found

1. attempt ← 0;
2. T ← initializeMaxTerm(F)
3. WHILE attempt < maxTries DO
4. IF ~CanBeExpand(T, F) THEN RETURN T
5. compute $P = \{v | \mathrm{Score}(v) > 0 \text{ and conf}[v] = 1\}$
6. IF $P \neq \emptyset$ THEN
7. x ← v ∈ P with the highest score
8. T ← T with x flipped
9. ELSE
10. use other rules to select the flipped variable x
11. attempt++
12. update the configuration of each variable
13. RETURN "no solution found"

3.2 Use subScore-assisted greedy search [4]

Definition 6 (critical variable): If $x \in V(C)$, then x is a critical variable in C, or x is an unrelated variable in C.

The critical variable of a clause is all the variables it contains, and such a variable will really determine whether the clause is true or false.

Obviously, the search direction of the algorithm is to make the value of more clause critical variables different from the maximum term.

We divide the clauses into three categories: extensible, critical and stable. If critical variables of the clause are as same as in the maximum term, then it's extensible. If there is only one critical variable that differs from the maximum term, then it's critical. In other cases, the clause is stable, that is, there are at least two of the critical variables that differ from the maximum term.

For example, $T = \{x_1, x_2, \neg x_3, \neg x_4\}, C_1 = \{x_1, x_2, x_3\}$, it can be said that C_1 is critical in T.

In the following discussion, we use flag[i] to represent the number of variables in the clause C_i that differ from the maximum term under the current maximum term, $\text{flag[i]} = 0$ means that C_i is extensible, $\text{flag[i]} = 1$, indicating that C_i is critical, $\text{flag[i]} \geq 2$, indicating that C_i is stable.

With the above definition, we introduce the subScore function as an auxiliary scoring function to help in the following situation. In the previous algorithm GSER, there may be a case where there are multiple variables satisfying $\text{conf[v]} = 1$ and score(v) are the greatest. At this time, a random selection strategy can be enabled, but from the perspective of the whole algorithm, we hope that the search direction of the maximum term proceeds along the direction that makes more clauses become critical or even stable. Therefore, we use subScore to record this tendency.

The definition of $\text{subScore}(v, C_i)$ is given as follows:

Given a maximum term T, if the variable v is flipped, flag[i] is incremented, then $\text{subScore}(v, C_i) = 1$,if flag[i] is decreased, then $\text{subScore}(v, C_i) = -1$. If flag[i] is unchanged, then $\text{subScore}(v, C_i) = 0$.

It can be seen that when v is not a critical variable of C_i, $\text{subScore}(v, C_i) = 0$; when v is a critical variable of C_i and the value of v in T is different from C_i, flipping it will reduce flag[i], $\text{subScore}(v, C_i) = -1$; $\text{subScore}(v, C_i) = 1$ when v is a critical variable of C_i and the value of v in T is the same as C_i.

So, the above definition can be written as:

$$\text{subScore}(v, C_i) = \begin{cases} 0, v \text{ is not a critical variable in } C_i \\ 1, v \text{ is a critical variable in } C_i \\ \quad \text{and } ref(T, v) = ref(C_i, v) \\ -1, v \text{ is a critical variable in } C_i \\ \quad \text{and } ref(T, v) \neq ref(C_i, v) \end{cases} \tag{1}$$

$$\text{and } \text{subScore}(v) = \sum_{i=1}^{m} \text{subScore}(v, C_i). \tag{2}$$

When there are multiple variables with $\text{conf[v]} = 1$ and the same score value, calculate their subScore value, flip the subScore maximum variable, if there are multiple subScore maximum variables, flipped according to the following strategy.

3.3 Maximum Score Upper Limit Strategy

In Algorithm GSER using subScore strategy, there may still be multiple variables with the largest Score and subScore. We propose the maximum score upper limit strategy to deal with this situation. The basic idea is to assume that the flipped variable is v, and the maximum term after flipping v is T'. Calculate the maximum value of Score under T' as the upper limit of Score after flipping v. For each variable, calculate the upper limit of the Score after the flip to help select the current variable, that is, select the variable flipped with the maximum Score limit.

Algorithm MaxScoreUpperLimit

Input: CNF formula $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$

Current maximum term T $= \{x_1, x_2, \ldots, x_n\}$
Candidate Set S $= \{x_p, x_{p+1}, \ldots, x_q\}$
Output: flipped variable x
1.max ← 0 x ← 0
2.FOR x_i in S DO
3. T' ← T with x_i flipped
4. max$[x_i]$ ← 0
5. FOR j ← 1 TO n DO
6. compute Score(x_j)
7. IF Score(x_j) > max$[x_j]$ THEN max$[x_j]$ ← Score(x_j)
8. IF max$[x_i]$ > max THEN max ← max$[x_i]$ x ← x_i
9.RETURN x

3.4 Clause weighting strategy [4]

The previous algorithm is performed when there is always a variable that its configuration is satisfied. When there is no variable that satisfies conf [v] =1, the algorithm uses clause weighting strategy which is used to update the weight of some clauses. We first increase the weights of extensible clauses by one, and then a clause that can be expanded by T is randomly picked, lastly select the variable that has not been flipped for the longest time in C_i to flip [4].

The framework of the overall algorithm is as follows:

Algorithm Improved Greedy Search algorithm based on Extension Rules(IGSER)

Input: CNF formula F $= C_1 \wedge C_2 \wedge \ldots \wedge C_m$
maxTries
Output: Non-expandable maximum term T $= \{x_1, x_2, \ldots, x_n\}$
or no solution found
1.attempt ← 0;
2.T ← initializeMaxTerm(F)
3.WHILE attempt < maxTries DO
4. IF ~CanBeExpand(T, F) THEN RETURN T
5. compute P $= \{v|\text{Score}(v) > 0 \text{ and conf}[v] = 1\}$
6. IF P ≠ ∅ THEN
7. compute Q $= \{v|\text{Score}(v) \text{ is the highest and } v \in P\}$
8. IF |Q| $= 1$ THEN T ← T with x ∈ Q flipped
9. ELSE
10. FOR v ∈ P DO compute the subScore(v)
11. compute S $= \{v|\text{subScore}(v) \text{ is the highest and } v \in Q\}$
12. IF |S| $= 1$ THEN T ← T with x ∈ S flipped
13. ELSE x ← MaxScoreUpperLimit(F, T, S) T ← T with x ∈ S flipped
14. ELSE
15. update the weights of extensible clauses
16. x ←the oldest variable in a random extensible clause
17. attempt++
18. update the configuration of each variable
19. RETURN "no solution found"

4. Evaluations of IGSER

In this subsection, we carry out experiments to evaluate the performance of IGSER on random 3-SAT instances.

4.1 Benchmarks and experiment preliminaries

Uniform Random-3-SAT, phase transition region, unforced filtered: $100 \leq$ variables ≤ 200, $300 \leq$ clauses ≤ 500. All instances used here are CNF formula encoded in DIMACS CNF format. This format is supported by most of the solvers provided in the SATLIB Solvers Collection[1].

Random-3-SAT Instances with Controlled Backbone Size: in order to fully test the performance of the IGSER, we choose some SAT problems with Controlled Backbone Size instances to test and verify the performance of IGSER.

IGSER is implemented in C++ and compiled by g++. Experiments in this section are run on a machine with a 4 cores of 3.0 GHz Intel(R) Core(TM) I5-7400 CPU and 4 GB RAM under Linux (Ubuntu 14.4). The cutoff time is set to 10000ms for the Uniform Random-3-SAT and CBS SAT. For each instance, each SAT solver is performed 100 times with a cutoff time of 10000ms, and we take the average to each of them.

4.2 Comparing IGSER on random 3-SAT

Table I presents the results of comparing these algorithms on the random 3-SAT (Comparative performance results of IGSER and NER, SER, GSER on the Uniform Random-3-SAT (variables =100, clauses=430).

Table.1. Uniform Random-3-SAT

Problems	Algorithm (CPU Time /ms)			
	NER	***SER***	***GSER***	***IGSER***
Uf100-01	-[a]	7899	1455	273
Uf100-02	-	-	-	422
Uf100-03	-	8080	1003	501
Uf100-04	-	-	-	785
Uf100-05	-	-	1369	1010
Uf100-06	-	-	988	674
Uf100-07	-	-	-	795
Uf100-08	-	-	1500	952
Uf100-09	-	-	1678	814
Uf100-10	-	-	1270	378

[a.] *- means time out.*

GSER shows a substantial improvement over NER and SER on these random 3-SAT instances [10-11]. On most instance classes, GSER achieves a higher effectiveness than NER and SER does. Table I also indicates that IGSER significantly outperforms GSER in terms of running time. And it can success in finding the result on all test cases. So, to great extent it means IGSER has been improved from GSER.

4.3 Comparing IGSER on CBS SAT problems

Table II presents the results of comparing these algorithms on the CBS SAT problems (Comparative performance results of IGSER and NER, SER on the CBS SAT). IGSER shows an absolute advantage over NER and SER on these CBS 3-SAT instances. On all instances, IGSER achieves a maximum term which meets the requirements, but NER and SER are all time out.

Table II also indicates that IGSER has the ability to solve different kinds of SAT problems and at the same time, it also runs successfully with good performance.

[1] https://www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html

4.4 The summary of this experiment

The experiments demonstrate that IGSER consistently outperforms its competitors (NER, SER) on both random 3-SAT and CBS SAT instances. And it also shows superiority over GSER, because in some solutions GSER only finds the local optima resulting in looping without the real maximum term but IGSER can do it by Maximum Score Upper Limit Strategy and subScore-assisted greedy search. So IGSER gives the best performance on those 3-SAT instances and CBS SAT instances.

5. Conclusion and Future Work

This paper proposes a new extension rule method (IGSER) based on greedy local search. Experiments show that the proposed algorithm outperforms the general extension rule inference method and algorithm GSER on 3-SAT and CBS SAT instances. In future work, we will continue to improve the effectiveness of our algorithm. And we can parallelize the algorithm to improve the efficiency of the SAT problems.

Table.2. Cbs Sat Problems

Problems	**Algorithm (CPU Time /ms)**		
	NER	***SER***	***IGSER***
CBS_k3_n100_m403_b10_0	-[a]	-	143
CBS_k3_n100_m403_b10_1	-	-	193
CBS_k3_n100_m403_b10_2	-	-	140
CBS_k3_n100_m403_b10_3	-	-	560
CBS_k3_n100_m403_b10_4	-	-	193
CBS_k3_n100_m403_b10_5	-	-	2082
CBS_k3_n100_m403_b10_6	-	-	588
CBS_k3_n100_m403_b10_7	-	-	349
CBS_k3_n100_m403_b10_8	-	-	480
CBS_k3_n100_m403_b10_9	-	-	671

[a.] *- means time out.*

References

[1] Selman B, Levesque H J, Mitchell D G, “A new method for solving hard satisfiability problems,” Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, USA, 1992, pp. 440-446.

[2] Cai S W, Su K L, “Local search with configuration checking for SAT,” Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence. Boca Raton, USA, 2001, pp.59-66.

[3] Cai S W, Su K L, “Configuration checking with aspiration in local search for SAT,” Proceedings of the 26th AAAI Conference on Artificial Intelligence and the 24th Innovative Applications of Artificial Intelligence Conference, Toronto, Canada, 2012, 1: 434-440.

[4] Cai S W, Su K L, “Local search for Boolean Satisfiability with configuration checking and subscore,” Artificial Intelligence, 2013, 204: 75-98.

[5] Cai S W, Luo C, Su K L, “Scoring functions based on second level score for k-SAT with long clauses,” Journal of Artificial Intelligence Research, 2014, 51: 413-441.

[6] Cai S W, Luo C, Su K L, “CCAnr: A configuration checking based local search solver for non-random satisfiability,” Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing, Austin, TX, USA, 2015, pp.1-8.

[7] Luo C, Cai S W, Wu W, Su K L, “Clause states based configuration checking in local search for satisfiability,” IEEE Transactions on Cybernetics, 2015,45 (5): 1014-1027.

[8] Lin H, Sun J G, Zhang Y M, “Theorem proving based on the extension rule,” Journal of Automated Reasoning, 2003, 31 (1): 11-21.

[9] Yang Y, Liu L, Li G L, Zhang T B, Lü S, “A novel local search-based extension rule reasoning method,” Journal of Computers, 2018, 41 (4): 825-839 (in Chinese)

[10] Sun J G, Li Y, Zhu X J, Lü S. “A novel theorem proving algorithm based on extension rule,” Journal of Computer Research and Development, 2009, 46 (1): 9-14 (in Chinese)

[11] Zhang L M, Ouyang D T, Bai H T, “Theorem proving algorithm based on semi-Extension rule,” Journal of Computer Research and Development, 2010, 47 (9): 1522-1529 (in Chinese)