Karlsruhe Institute of Technology

SAT Solving with distributed local search

Master Thesis of

Guangping Li

At the Department of Informatics
Institute of Theoretical informatics, Algorithmics II

Advisors: Dr. Tomáš Balyo
Prof. Dr. Peter Sanders

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, 1th September 2018

Abstract

Stochastic local search (SLS) is an elementary technique for solving combinational problems. Probsat is an algorithm paradigm of the simplest SLS solvers for Boolean Satisfiability Problem (SAT), in which the decisions only based on the probability distribution. In the first section of this paper, we introduce an efficient Probsat heuristic. We experimentally evaluate and analyze the performance of our solver in a combination of different techniques, including simulated annealing and tabu search. With the approach of formula partition, we introduce a parallel version of our solver in the second section. The parallelism improves the Efficiency of the solver. Using different random generator and other parameter settings in solving the sub-formula can bring further improvement in performance to our parallel solver.

Zusammenfassung

Stochastische lokale Suche (SLS) stellt eine elementare Technik zur Lösung von komplizierten kombinatorischen Problemen dar. Probsat ist einer der einfachsten SLS-Solver für das Erfüllbarkeitsproblem der Aussagenlogik (SAT), bei dem die Entscheidungen nur auf der Wahrscheinlichkeitsverteilung basieren. Im ersten Teil dieser Arbeit stellen wir eine effiziente Probsat-basierte Heuristik vor. Die Leistung unseres Algorithmus in einer Kombination verschiedener Techniken, einschließlich simulierter Abkühlung und Tabu Search wurde auch experimentell bewertet und analysiert. Mit dem Ansatz der Formelpartition wird im zweiten Teil eine parallele Version unseres Algorithmus eingeführt, die die Effizienz des Lösers verbessert. Die flexible Parametereinstellungen bei der Lösung der Teil-formeln kann eine weitere Verbesserung unseres Algorithmus bringen.

Contents

1 Introduction

1.1 Problem/Motivation

The ***propositional satisfiability problem*** (***SAT***) is the first proven NP-complete problem [1]. The problem is to determine whether an assignment of Boolean values to variables in a Boolean formula such that the expression evaluates to true. As one of the most studied problems in computer science, the SAT problem has many applications. Hard combinational problems can be resolved with appropriate Encoding as a sat problem. The SAT problem has many applications in computer science like chip model checking [2], software verification [3] or in automated planning and scheduling in artificial intelligence [4]. Formula partition is one of the promising approaches in DPLL-like solvers [5]. By giving the order to the variables according to a good formula partition, the search gets a relatively balanced decision tree. But formula partition is rarely used in a local search for the SAT problem. How to combine the formula partition with local search, will the local search benefit from the partitioning, if the formula partitioning can guide a parallel local search, are still open questions.

1.2 Content

The SAT problem, as a well-known NP-complete problem, has received a great deal of attention and different local search heuristics have been developed. This paper is a survey on the stochastic local search on SAT problem with a guide of formula partition.
In section 1, we summarize the formal concept and introduces techniques used in this paper. One class of the most straightforward but efficient stochastic local search algorithms Probsat is the algorithm basic in our paper. Probsat was proposed in 2012 by Adrian Balint and Uwe Schoening [6]. Section 2 describes our Probsat algorithm and discusses our attempts to improve the original algorithm. By experimentally evaluation and comparison, some techniques turned out to be more efficient than the simple Probsat search. With the partition of variables and its corresponding formulas, the problem can be separated into two subproblems of similar size. In section 3, we search the potential benefit of formula partition in a parallel search. Section 4 describes the details in experiments and several empiric results mentioned in section 2 and section 3. Section 5 concludes the paper with further works.

1.3 Definitions and Notations

Propositional Satisfiability Problem
A variable with two possible logical values *TRUE* or *False* is a***propositional variable***, which will be referred to as ***variable*** in this paper. A ***literal*** is an atomic formula in propositional logic. A literal can either be a ***positive literal*** v as the variable v or a ***negative literal*** $\bar{v}$ as negation of v. A ***clause*** is a disjunction of literals. A formula in conjunctive normal form (CNF) is a conjunction of clauses. We refer it as ***CNF-formula*** or simply as ***formula*** in this paper. An ***assignment*** a: assigns the truth value to each variable v in the formula. We say the assignment satisfies a formula if the truth value of the formula with this assignment turns out to be true. Specifically, an assignment satisfies a clause, if one literal in the clause with value *True* in this assignment. A formula is a satisfying formula if one assignment exists satisfies all its clauses. We say an assignment a satisfying assignment if it satisfies the formula. Otherwise, we say there are conflicts in some clauses with this assignment, or some clauses

are unsatisfying clauses with this assignment. The SAT problem is to determine whether a satisfying assignment exists for the given formula. If so, we denote the formula a satisfiable formula.

Set
A set is a container of unique elements. A set of 3 objects a, b, c is written as a, b, c. The size of a set is the number of elements in the set.

Local Search
For instance I of hard combinational Problem, there is a set of solutions. According to the constraints of the problem, an object function (score) is used to evaluate the candidate solutions. The Goal of the local search is to find the solution of minimum cost (the solution with the maximal score). A local search starts with an initial complete solution. According to some heuristic, the local search makes local changes to its current solution iteratively, hence the name local search. Starts from an initial solution, the search will evaluate the solutions which can be reached by applying a local change to the current solution and choose one of the neighbor solutions with local optimization. The search applies local moves until the optimal solution is reached, or in some cases, a generally good solution is reached. Local search is widely used in hard combinational problems such as the traveling salesman problem [13] and the graph coloring problem [14]. In the Boolean satisfiability problem, a local search operates primarily as follows: The search start from a randomly generated assignment as the initial solution. If this current assignment satisfied the formula, the search stops with success. Otherwise, a variable is chosen depends on some criterion, which we further call pickVal. By change the assignment of the selected variable, a neighbor assignment of our current solution is reached in next step, which is also called variable flipping. A local search will move in the space of the assignments by making the variable flipping until a satisfying assignment is reached by the search. The heuristic used for the pickVal selection is based on some scores of the variables in the current assignment.Consider the assignment reached by taking a flip of the variable in the current assignment. The number of clauses satisfied in B,but not in A is called the breakcout of the local move from to. Accordingly, the number of clauses, which become satisfying because of the flipping, is the makecount. The number of newly satisfying clauses (make) minus the number of newly unsatisfying clauses (break), which is denoted as diffscore, represents the local improvement of the corresponding flipping. Apart from this, other aspects like the repetition number of one flip or the number of occurrences of the variables can be considered in a selection heuristic. An example is the unit propagation embedded local solver EagleUp, which prefers flipping of variable with the highest number of occurrences in a formula to creates new unit clauses sooner. To get local improvement effectively, man can only consider variables in unsat clauses for the flipping selection. This process is called a focused local search and commonly used. The initial hope of the local search is that through iterative greedy local improvement the optimal global solution can be found. The typical problem of the local search is that the greedy local improvement searches be trapped in local unattractive local optimal solution. To avoid this, a worse solution then the current solution will be chosen for the next step (uphill moves). There are some techniques following used in local search with occasional uphill moves.

Stochastic Local Search
The stochastic local search will use the probability distribution of the scores of candidate solutions instead of the static decision. For the candidate moves, the probability of being chosen p(T(s)) corresponds to the scores of the solution. In this way, the advantage a move is, the probability of choosing it as the next step is higher. This randomization will avoid the stuck of the search in a local minimum and decrease the misguiding of the heuristic in specific situations.

Tabu Local Search
Tabu search is created by Fred W. Glover in 1986 [15] and formalized in 1989. For recognize the loop in a suboptimal region, the search trace is recorded in the process by mark the recently reached neighboring assignments as tabu. The tabu moves will not be touched in the further search to discourage getting stuck in a region.

Simulated Annealing
Simulated Annealing is an approach of SLS solver to difficult combinational optimization problems proposed by Kirkpatrick, Gelatt, and Vecchi. This approach is inspired by the metallic process annealing of shaping the material by heating and then slowly cooling the material. This approach works as a local optimization algorithm guided by a controlling parameter temperature. By high temperature, an uphill move is allowed with high probability while only small steps are allowed in low temperature. The temperature is varying according to the score of the current situation. For a current solution with a nearly optimal score, the temperature is near zero. For an unattractive local extreme with a poor score, the active search is tending to make uphill moves in high temperature.

The Probsat
Probsat is a class of SLS sat solver, which was introduced in 2012 by Adrian Balint and Uwe Schoening. In a probsat solver, the score of a candidate flip is solely based on the make and break score. The paradigm is as follows: At first, a completely random assignment is set as the initial assignment. In the initial solution, the truth value of The algorithm performs local moves by flip a variable in a random chosen unsatisfying clause and stops as soon as there are no unsatisfying clauses exists, which means the satisfying assignment is found. The flip probability p of the variables in the chosen clause is calculated in a function p(x,s) based on make and break score of the variable in the current assignment. The idea behind the probability function is to give the advantageous flipping relative high probability, but the other flipping has small chance to be chosen. There are two kinds of functions are considered in the paper of Adrian Baliant: exponential function and a polynomial function. An exponential function f is an exponential 2-parameter (cb and cm) function: Or a function with polynomial decay, which is called a polynomial function in the original paper: As mentioned in the probsat paper, it turns out in experiments that the influence or make is rather weak, so the one parameter function like the following can lead to an efficient algorithm. The pseudo code of a typical Probsat is shown below: Algorithm 1: ProbSAT Input: Formula F, maxTries, maxFlips Output: satisfying assignment a or UNKNOWN For I =1 to maxTries do: a<- randomly generated assignment For I =1 to maxFlips do: If(a satisfies F) then Return a C

1.4 The algorithms for comparison

2 Our local Solver

2.1 Probsat

2.2 Improvement through randomly generated solution

2.3 Improvement through random generator

2.4 Improvement through statistic

2.5 try star version

3 Our Parallel Algorithm

3.1 1st Approach: The pure portfolio approach (no partition)

3.2 2nd Approach: Star

3.3 3nd Approach: Hope

3.4 4th Approach: future

4 Evaluation

4.1 DIMACS standard format

All the benchmark CNF formula used in experiments are encoded in the DIMACS standard format [**?**]. This format is used to test and compare SAT solver in SAT competition. A DIMACS file contains the description of an instance using three types of lines[1]:

1. Comment line: Comment lines give information about the graph for human readers, like the author of the file or the seed used in generation. A comment line starts with a lower-case character c and will be ignored by programs:

$$\mathbf{c} \quad \textit{this is an example of the comment line}$$

2. Problem line: The problem line appears exactly once in each DIMACS format file. The problem line is signified by a lower-case character p. For a formula with nV variables and nC clauses ,the problem line in its DIMACS file is:

$$\mathbf{p} \text{ cnf } nV \ nC$$

[1]Only clause unweighted simple instances are tested in our experiments. For other descriptors and details of the DIMACS format, see [**?**].

3. Clause Descriptor: An clause $\{v_1, v_2, .., v_n\}$ in the graph is described in an edge Descriptor:
e v_1 v_2 .. v_n

```
c This is a DIMACS file of the graph in Figure 6
p edge 7 11
e 1 3
e 1 2
e 2 7
e 2 6
e 1 6
e 1 4
e 1 5
e 4 5
e 3 5
e 5 6
e 6 7
```

Figure 1: A DIMACS file example of the problem in Figure **??**

4.2 Benchmarks

The benchmark instances used in experiments are the 180 uniform instances (unif) in random benchmark categories in SAT competition 2017 [7]. In an unif problem file, all the clause have the same length. The suffix 'k' denotes the length of clauses. The r indicates the clause-to-variable ratio. The c and v are for the number of clauses and variables, while s is for the seed used in the generation process. Without flitering, there are at least 60 (33%) problems form our 180 benchmark collections are unsatisfiable.

4.3 Used plots and tables

Different plots and tables are used to illustrate the results of the following experiments.
Comparison Table
Scatter Plot
Cactus Plot
Advantage Plot

4.4 Automatic parameter optimization

4.5 Experiments

5 Conclusion

5.1 Further work

6 Bibliography

References

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, ACM, 1971. (Page 1).

[2] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal methods in system design*, vol. 19, no. 1, pp. 7–34, 2001. (Page 1).

[3] F. Ivančić, Z. Yang, M. K. Ganai, A. Gupta, and P. Ashar, "Efficient sat-based bounded model checking for software verification," *Theoretical Computer Science*, vol. 404, no. 3, pp. 256–274, 2008. (Page 1).

[4] H. Kautz and B. Selman, "Unifying sat-based and graph-based planning," in *IJCAI*, vol. 99, pp. 318–325, 1999. (Page 1).

[5] Z. Á. Mann and P. A. Papp, "Guiding sat solving by formula partitioning," *International Journal on Artificial Intelligence Tools*, vol. 26, no. 04, p. 1750011, 2017. (Page 1).

[6] A. Balint and U. Schöning, "Engineering a lightweight and efficient local search sat solver," in *Algorithm Engineering*, pp. 1–18, Springer, 2016. (Page 1).

[7] T. Balyo, M. J. Heule, and M. Jarvisalo, "Proceedings of sat competition 2017: Solver and benchmark descriptions," 2017. (Page 5).