A Distribution Method for Solving SAT in Grids

Antti E. J. Hyvärinen, Tommi Junttila, and Ilkka Niemelä

Helsinki University of Technology, Laboratory for Theoretical Computer Science
{Antti.Hyvarinen,Tommi.Junttila,Ilkka.Niemela}@tkk.fi

Abstract. The emerging large-scale computational grid infrastructure is providing an interesting platform for massive distributed computations. In this paper the problem of exploiting such computational grids for solving challenging propositional satisfiability problem (SAT) instances is studied. When designing a distributed algorithm for a large loosely coupled computational grid, a number of grid specific problems need to be tackled including the heterogeneity of the resources, inherent communication delays, and high failure probabilities of grid jobs. In this work a novel distribution method for solving SAT problem instances, called scattering, is introduced. The key advantages of scattering are that it can be used in conjunction with any sequential SAT solver (including industrial black box solvers), the distribution heuristic is strictly separated from the heuristic used in sequential solving, and it requires no communication between processes solving subproblems but still allows coordination of such processes. An implementation of the method has been developed for NorduGrid, a large widely distributed production-level grid running in Scandinavia. The implementation has been benchmarked with test cases including random 3SAT and challenging industrial benchmarks used in previous SAT competitions.

1 Introduction

We study the *propositional satisfiability problem* (SAT) of determining whether a given propositional formula has a satisfying truth assignment. Decision methods for SAT and their implementation techniques have advanced considerably during the last decade and SAT based techniques have been applied successfully in several areas including planning [1,2], model checking of finite state systems [3,4], testing [5], hardware verification [6], VLSI-routing [7], and scheduling [8].

An interesting approach to boosting the applicability of SAT based problem solving is to exploit parallel computation to solve SAT problem instances. In particular, the emerging large scale computational grids make this approach increasingly attractive. For example, the largest software project currently funded by the European Union is the EGEE project (Enabling Grids for E-sciencE; `http://public.eu-egee.org/`).

In this paper we study how to exploit the grid infrastructure in solving challenging SAT instances. Compared to more tightly coupled parallel computing architectures, grids have properties that need to be taken into account when designing distributed algorithms. In particular, (i) the available resources can be quite heterogeneous in a grid, (ii) no shared memory is available and communication delays are significant, and (iii) individual jobs executed in grid nodes have non-negligible failure rates. The goal

is an approach where we can exploit the best SAT solving techniques and the grid resources. For this we are developing distributed SAT solving methods that satisfiy the following set of requirements.

- Any SAT solver can be exploited in solving a problem without changes.
- If the used SAT solvers are complete, then the distributed method is complete.
- The methods are usable in a wide variety of grid infrastructures.
- There is no communication between grid nodes during solving. This is important as support for inter-node communication is limited due to security reasons.
- The methods are fault-tolerant and recover and maintain completeness even if grid processes fail.

Several parallel SAT solvers designed to work in a distributed environment have been described [9,10,11,12,13,14]. However, in these approaches it is not possible to exploit a chosen sequential SAT solver directly but they are based on developing a special purpose SAT solver for the distributed case. Moreover, distribution of work and load balancing are fairly tightly coupled with the decisions made in the SAT solver and a significant amount of inter-process communication is needed.

A straightforward way to satisfy the requirements above is to use an approach we call *Simple Distributed SAT* (SDSAT) where a SAT instance is solved by running a number of SAT solvers on the same instance as independent jobs and waiting until one of them solves the problem. If a randomized SAT solver is available (for example, `Satz-rand`, `WalkSAT`, or `AdaptiveNovelty+`), this solver can be used with different seeds. This approach has potential as results, e.g., in [15,16] indicate.

However, an obvious weakness of SDSAT is that there is no cooperation between the independent jobs, i.e., there is no way that progress and partial results obtained in one job can contribute to completing another job. This is a serious shortcoming because each process needs to be given resource bounds (CPU time, memory) when the process is sent to a grid. Thus, the natural way of running SDSAT by starting jobs without resource bounds and waiting until one of them succeeds is not available in grids. Moreover, job management in a grid typically takes into account the resource requirements of a job, implying lower priority to jobs with substantial resource demands and longer delays.

In this paper we present a novel distributed SAT solving method called *scattering*, which satisfies the requirements above but still allows for cooperation. The basic idea is to divide a given SAT instance gradually to increasingly more constrained subproblems that are sent to the grid to be solved using practically any available SAT solver. While this is somewhat similar to SDSAT, there are substantial differences.

- The subproblems to be solved become easier to solve as the computation proceeds.
- Learning techniques used in sequential solvers can be exploited when dividing a problem to subproblems.
- The division of a problem to subproblems is based on estimating the computational cost of the subproblems in order to achieve better load-balancing.

We have an implementation of the method, called SATU, for NorduGrid [17] (`http://www.nordugrid.org/`), a widely distributed Scandinavian computational grid.

The rest of the paper is structured as follows. The scattering method is described in Sect. 2 and Sect. 3 reports experiments on the feasibility of the approach and concludes.

2 The Algorithm

The goal is to develop an algorithm for solving challenging SAT instances on a variety of grids. In order to employ a wide range of grids, we make minimal assumptions regarding the *Distributed Execution Environment* (DEE) provided by a grid. The DEE is assumed to offer a simple interface between the client sending *executions* (executable programs together with their inputs), and the environment receiving and running the executions. The only functionalities available to the client are (i) Send, which sends an execution to the environment, (ii) Monitor, which reports the state of the execution, and (iii) Receive, which returns the result of an execution.

We do not assume that the executions are able to communicate directly with each other. We also assume that the DEE has some maximum of simultaneous executions it can hold. If this limit is reached, the environment is *saturated*, and any new executions may fail without a result. The executions must finish when some condition given at construction time is triggered, e.g, a CPU time or memory limit is exceeded.

The simple distributed SAT (SDSAT) scehme is not optimal for solving resource intensive problems in a grid because of the resource bounds and job management policies explained in the introduction. In order to address deficiencies of SDSAT we have developed a distribution method called *scattering*. The basic ideas underlying scattering are quite straightforward.

- A SAT problem instance is divided to a set of subproblems by adding new constraints (clauses) to the original problem to make the subproblems easier to solve.
- The division of a problem to subproblems (a scattering step) is done so that (i) if all subproblems have been solved, we get a solution to the original problem, and (ii) search spaces of the subproblems are disjoint.
- An execution of a subproblem is interrupted if a given resource bound is exceeded and the problem is divided further.

Next we explain (i) the basic scattering rule and the scattering tree, (ii) the technique employed to generate subproblems with comparable estimated computational cost, and (iii) the heuristic used to select constraints to be added.

2.1 The Basic Scattering Rule and the Scattering Tree

The scattering rule constructs a parameterized number sf of scattered formulas $F_1, \ldots, F_{sf}$ from a formula F such that

$$F_i = \begin{cases} F \wedge T_1 & \text{if } i = 1 \\ F \wedge \neg T_1 \wedge \cdots \wedge \neg T_{i-1} \wedge T_i & \text{if } 1 < i < sf \\ F \wedge \neg T_1 \wedge \cdots \wedge \neg T_{sf-1} & \text{if } i = sf. \end{cases} \quad (1)$$

T_i is a conjunction $l_1^i \wedge \cdots \wedge l_{d_i}^i$ of d_i literals heuristically selected (Sect. 2.3) and d_i is selected to yield comparably sized subproblems (Sect. 2.2). The expression $\neg T_i = \bar{l}_1^i \vee \cdots \vee \bar{l}_{d_i}^i$ is the negation of the conjunction T_i. Thus constructed propositional formulas have the properties that (i) the disjunction $F_1 \vee \cdots \vee F_{sf}$ is logically equivalent to the formula F, and (ii) no two formulas F_i, F_j, $i \neq j$, share a satisfying truth assignment.

We solve a SAT instance F_r by performing a distributed search in a *scattering tree* with root F_r . The nodes are formulas obtained by the scattering rule so that the children of a node F are the scattered formulas $F_1, \ldots, F_{sf}$. The search is implemented by sending formulas associated to nodes as jobs to be solved in the DEE. A part of a possible scattering tree is given in Fig. 1.

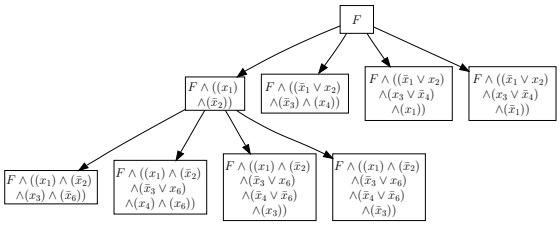

Fig. 1. A part of a scattering tree

A node of the tree is *computed satisfiable* if the corresponding job returns with this answer, or if at least one of the children is computed satisfiable. The node is *computed unsatisfiable* if the corresponding job returns unsatisfiable, or if all children have been computed unsatisfiable. The scattering tree is constructed incrementally while sending the jobs to the DEE until F_r is computed satisfiable or unsatisfiable.

The correctness of the computed answer follows from the following observations. Since scattered formulas of F are more constrained than F by (1), a satisfying truth assignment for one of the scattered formulas satisfies the formula F as well. Moreover, since the disjunction of the scattered formulas is logically equivalent to the formula F, if all scattered formulas are unsatisfiable, the formula F is also unsatisfiable. Furthermore, if a propositional formula is computed unsatisfiable, all possible scattered formulas, being more constrained, are also unsatisfiable.

An ancestor of a formula in the scattering tree is less constrained than the child. As a result, all clauses which are logical consequences of the ancestor are also logical consequences of the child. Learned clauses from formula F can be included to the formula F' only if F is ancestor of F' in the scattering tree.

2.2 Balancing the subproblems

The basic scattering rule on F will fix d_i literals by the conjunction T_i on each scattered instance F_i, $1 \leq i \leq sf - 1$. We try to divide F to F_i that have comparable estimated computational costs.

Let $t(F)$ denote the estimate of the time required to solve a formula F directly. Then the time required to solve each F_i should be comparable to $t(F)/sf$. Since the solution spaces of the instances are distinct by (1), we may assume that the solving times $t(F_j) = t(F)/sf, 1 \leq j < i$ of the previously constructed problems can be subtracted from the total solving time $t(F)$ to get the time required to solve the remaining problem. Since $t(F_i)$ should also be $t(F)/sf$, a proportion r_i should be constructed from the remaining problem of approximate run time $t(F) - (i-1)(t(F)/sf)$. This yields the equation $t(F)/sf = (t(F) - (i-1)(t(F)/sf))\, r_i$. When solved for r_i, this becomes $r_i = (sf - i + 1)^{-1}$.

As an approximation of the computational cost we assume the worst case behaviour, that is, $t(F) = \mathcal{O}(2^n)$ where n is the number of variables in F. If d_i is the number of literals fixed in the scattering rule to obtain F_i, then $t(F_i) = \mathcal{O}(2^{n-d_i})$, when assuming direct simplification by unit propagation. The problem is to minimize $|2^{-d_i} - r_i|$.

An example for $sf = 7$ is given in Table 1. The first column shows values of r_i for $i = 1, \ldots, 7$, the second column shows the values of d_i minimizing $|2^{-d_i} - r_i|$, and the third column shows the resulting estimated fractions of the full problem.

2.3 The Heuristic

The selection of literals for the scattered formulas in (1) is a heuristic process aiming at constructing scattered formulas which have comparable solving times and are more constrained than the formula from which they are scattered. We have implemented the heuristic selection of scattering literals on top of a SAT solver implementation [18] similar to `zChaff` [19]. When selecting the d_i literals for the scattered formula F_i, the procedure takes as input $F \wedge \neg T_1 \wedge \cdots \wedge \neg T_{i-1}$, runs the `zChaff` type algorithm until it reaches the decision level $d_i + 1$ and then selects as the literals $l_1^i, \ldots, l_{d_i}^i$ the d_i decision literals chosen on levels $1, \ldots, d_i$.

r_i	d_i	Est.
$\frac{1}{7}$	3	0.125
$\frac{1}{6}$	3	0.109
$\frac{1}{5}$	2	0.191
$\frac{1}{4}$	2	0.144
$\frac{1}{3}$	2	0.108
$\frac{1}{2}$	1	0.161
1	0	0.161

Table 1. a balancing example for $sf = 7$

Our modification of the original `zChaff` VSIDS heuristic [19] aims at finding variables that divide the remaining search space into parts of comparable size. In contrast to the greedy VSIDS heuristic, we rank each variable x with the scores of literals x and $\neg x$ and choose the variable which has the best lower score, and finally select the literal corresponding to the higher score of this variable as the decision literal.

3 Conclusions

The paper presents a novel method for distributed SAT solving, called scattering, and studies the performance of SATU using benchmarks from previous SAT competitions. Table 2 shows preliminary comparison between SATU and SDSAT on some SAT instances. The 4th column shows the number of SDSAT runs which were faster than SATU. The SDSAT runs were were computed by 200 MHz AMD Athlon(tm) 64 Processor 3200+. More results are available from [20].

Name	SATU (s)	SDSAT (s)	#	Result
cnt10	1121	540	57	sat
dp10u09	586	149	19	unsat
f2clk_40	8078	4831	34	unsat
lisa20_1_a	240	1	20	sat
lisa21_3_a	1017	206	5	sat
Mat26	1503	4151	0	unsat
vda_gr_rcs_w8	1160	1	5	sat

Table 2. comparison between SATU and SDSAT

Scattering differs from other distributed SAT solving methods, such as [9,10,11,12,13,14] in a number of ways: (i) any SAT solver, including industrial black box solvers, can be used with no modifications, (ii) it has modest requirements for communication but still allows process coordination, (iii) when the solving of a problem needs to be distributed, it is possible to divide the problem to an arbitrary number of subproblems, and (iv) heuristics for dividing a problem to subproblems is separated from the heuristic for solving individual subproblems.

Interesting topics of future work include the optimization of the scattering algorithm and the grid job management. Also a more thorough comparison between the SDSAT method and SATU might provide some useful hints on how to combine the benefits of both approaches.

References

1. Kautz, H., Selman, B.: Planning as satisfiability. In: ECAI 1992, John Wiley and Sons (1992) 359–363
2. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic, and stochastic search. In: AAAI/IAAI 1996, AAAI Press (1996) 1194–1201
3. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1) (2001) 7–34
4. Bjesse, P., Leonard, T., Mokkedem, A.: Finding bugs in an Alpha microprocessor using satisfiability solvers. In: CAV 2001. Volume 2102 of LNCS., Springer (2001) 454–464
5. Larrabee, T.: Test pattern generation using Boolean satisfiability. IEEE Transactions on Computer-Aided Design **11**(1) (1992) 6–22
6. Biere, A., Kunz, W.: SAT and ATPG: Boolean engines for formal hardware verification. In: ICCAD 2002, ACM (2002) 782–785
7. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: Solving difficult instances of boolean satisfiability in the presence of symmetry. IEEE Transactions on Computer Aided Design **22**(9) (2003) 1117–1137
8. Zhang, H., Li, D., Shen, H.: A SAT based scheduler for tournament schedules. In: SAT 2004. (2004) Online proceedings at `http://www.satisfiability.org/SAT04/programme/index.html`.
9. Zhang, H., Bonacina, M., Hsiang, J.: PSATO: A distributed propositional prover and its application to quasigroup problems. Journal of Symbolic Computation **21**(4) (1996) 543–560
10. Chrabakh, W., Wolski, R.: GridSAT: A chaff-based distributed SAT solver for the grid. In: SC 2003, IEEE (2003)
11. Jurkowiak, B., Li, C., Utard, G.: A parallelization scheme based on work stealing for a class of SAT solvers. Journal of Automated Reasoning **34**(1) (2005) 73–101
12. Sinz, C., Blochinger, W., Küchlin, W.: PaSAT — Parallel SAT-checking with lemma exchange: Implementation and applications. In: SAT 2001. Volume 9 of Electronic Notes in Discrete Mathematics., Elsevier (2001) 12–13
13. Blochinger, W., Sinz, C., Küchlin, W.: Parallel propositional satisfiability checking with distributed dynamic learning. Journal of Parallel Computing **29**(7) (2003) 969–994
14. Forman, S., Segre, A.: NAGSAT: A randomized, complete, parallel solver for 3-SAT. In: SAT 2002. (2002) Online proceedings at `http://gauss.ececs.uc.edu/Conferences/SAT2002/sat2002list.html`.
15. Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. Journal of Automated Reasoning **24**(1/2) (2000) 67–100
16. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126**(1-2) (2001) 43–62
17. Eerola, P., Konya, B., Smirnova, O., Ekelöf, T., Ellert, M., Hansen, J.R., Nielsen, J.L., Wäänänen, A., Konstantinov, A., Ould-Saada, F.: Building a production grid in Scandinavia. IEEE Internet Computing **7**(4) (2003) 27–35
18. Zhang, L.: SAT-solving: From Davis-Putnam to Zchaff and beyond, lecture notes (2003) Available online at `http://research.microsoft.com/users/lintaoz/SATSolving/satsolving.htm`.
19. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC 2001, ACM (2001) 530–535
20. Hyvärinen, A.E.J.: SATU: A system for distributed propositional satisfiability checking in computational grids. Research Report A100, Helsinki Univ. of Technology, Lab. for Theoretical Comp. Science, Espoo, Finland (2006)