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 a novel distribution method called scattering is introduced for solving SAT problem instances in grid environments. 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.

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 such as planning [1] and model checking [2].

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 to grid nodes are significant, (iii) inter-node communication is very limited and often not available, (iv) 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.

Several techniques for distributed SAT solving have been proposed [3–12]. 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 distributed

SAT solver. 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 use grid computing to solve SAT problems is to employ 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, this solver can be used with different seeds. This approach has potential as results, e.g., in [13, 14] indicate.

However, the natural way of running SDSAT by starting jobs without resource bounds and waiting until one of them succeeds is not available in grids. This is because each process needs to be given resource bounds (CPU time, memory) when the process is sent to a grid. Moreover, grid job management typically takes into account the resource requirements of a job, implying lower priority to jobs with substantial resource demands and, thus longer delays.

In this paper we present a novel distributed SAT solving method called *scattering*. The basic idea is to divide a given SAT instance gradually to increasingly more constrained subproblems which 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. (i) The subproblems to be solved become easier to solve as the computation proceeds. (ii) Learning techniques used in sequential solvers can be exploited when dividing a problem to subproblems. (iii) The division of a problem to subproblems is based on estimating the computational cost of the subproblems in order to achieve better load balancing.

Scattering has similar modest communication requirements as SDSAT and is thus different from typical guiding path based approaches, like [3], which need inter-process communication and special customized solvers for dynamic load balancing (and failure recovery). In scattering dynamic load balancing (and failure recovery) is handled without inter-process communication by (i) dividing a running job into subproblems and starting jobs on them even before the result of the running job is available and (ii) setting at job construction time resource bounds (time and memory) which when exceeded terminate the job.

Scattering is implemented in SATU, a SAT solver designed to work with NorduGrid [15] (`http://www.nordugrid.org/`), a widely distributed Scandinavian grid. SATU is available at `http://www.tcs.hut.fi/Software/satu/`.

2 The Algorithm

The goal is to develop an algorithm for solving challenging SAT instances on a wide range of grids. In order to support this, we make minimal assumptions regarding the *Distributed Execution Environment* (DEE) provided by a grid. 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, sending an execution to the environment, (ii) Monitor, reporting the state of the execution, and (iii) Receive, returning the result of an execution.

We assume that the executions are not able to communicate directly with each other. We also assume that 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) scheme 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 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) solution spaces of the subproblems are disjoint.
- Until a computed answer is obtained, subproblems are further divided by scattering steps whenever possible, i.e., when DEE is below the saturation limit. This means that the execution of a subproblem exceeding a given resource bound can be interrupted (or a grid node can suffer a failure) without compromising the completeness of the method.

2.1 The Basic Scattering Rule and the Scattering Tree

The scattering rule takes as input a formula F and a number $sf (\geq 2)$ (scattering factor) and constructs sf scattered formulas $F_1, \ldots, F_{sf}$ from F such that

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

Each 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 each number 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 DEE. A part of a possible scattering tree is given in Fig. 1.

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 according to the chosen search strategy while sending jobs to DEE whenever DEE is below its saturation limit until F_r is computed satisfiable or unsatisfiable.

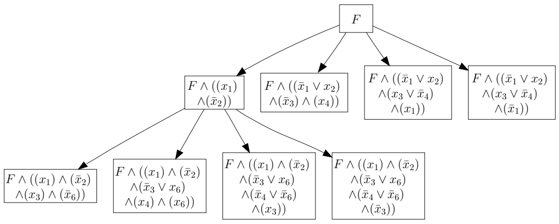

Fig. 1. A part of a scattering tree

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 F as well. Moreover, since the disjunction of the scattered formulas is logically equivalent to F, if all scattered formulas are unsatisfiable, F is also unsatisfiable. Furthermore, if a 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 thus be included to the formula F' if F is ancestor of F' in the scattering tree.

2.2 Balancing the subproblems

The goal in a scattering step is to divide a given formula F to scattered formulas $F_1, \ldots, F_{sf}$ that have comparable estimated computational costs by choosing d_i fixed literals for each F_i. This is done by selecting for each $i = 1, \ldots, sf - 1$, the number d_i such that $|2^{-d_i} - r_i|$ is minimized, where $r_i = (sf - i + 1)^{-1}$.

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 shows the values of d_i minimizing $|2^{-d_i} - r_i|$, and the third the resulting estimated fractions of the full problem.

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. d_i ($sf = 7$)

The idea behind this approach is to divide the solution space of F to sf portions of equal size. The solution spaces of different F_is are disjoint by construction. Hence, for each i, if $F_1, \ldots, F_{i-1}$ already cover $i - 1$ of these portions, then an equal size portion is obtained by taking the fraction r_i from the remaining solution space. By fixing d_i literals a fraction 2^{-d_i} of the solution space can be chosen. Hence, for each i, the numbers d_i are chosen so that 2^{-d_i} is as close as possible to r_i.

2.3 The Heuristic

The selection of literals for the scattered formulas in (1) is a heuristic process which we have implemented on top of a SAT solver implementation [16] similar to zChaff [17]. The SAT solver is run first on the formula F until a given number of conflicts is reached to initialize the heuristic scores of the literals as in zChaff and the initialization is continued as long as DEE is above the saturation limit. 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 $1, \ldots, d_i$. If the decision level $d_i + 1$ cannot be reached, this produces a computed solution for $F_i, \ldots, F_{sf}$.

We use a modification of the original zChaff VSIDS heuristic [17] which 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 Experimental Results and Conclusions

Some results, obtained with SATU 0.2 having zChaff as the SAT solver, are shown in Table 2. The scatter results are with scattering factor $sf = 7$, per job timeout ranging randomly between 60 and 90 minutes, memory limit of 1 GB and saturation limit of 64. The times are compared to minimum of randomized zChaff run times (out of 64). The fourth column indicates the number of cases run time of randomized zChaff was smaller than the wall clock time of SATU. We run zChaff on 2 GHz AMD Athlon(tm) 64 Processor 3200+ with 2 GB of memory. Note that the results of zChaff do not include the very substantial communication delays of a widely distributed grid. More experimental results are available in [18].

Table 2. Run times with SATU on formulas from SAT2002 competitions

Name	scatter (s)	$\texttt{zChaff}_{\text{min}}$ (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

Scattering differs from other distributed SAT solving methods, such as [3–12] 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) heuristic 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 job management. Also a more thorough comparison between scattering and other methods is needed.

References

1. Kautz, H., Selman, B.: Pushing the envelope: Planning, propositional logic, and stochastic search. In: AAAI/IAAI 1996, AAAI Press (1996) 1194–1201
2. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1) (2001) 7–34
3. Zhang, H., Bonacina, M., Hsiang, J.: PSATO: A distributed propositional prover and its application to quasigroup problems. J. Symbolic Computation **21**(4) (1996) 543–560
4. Chrabakh, W., Wolski, R.: GridSAT: A chaff-based distributed SAT solver for the grid. In: SC 2003, IEEE (2003)
5. 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
6. 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
7. Blochinger, W., Sinz, C., Küchlin, W.: Parallel propositional satisfiability checking with distributed dynamic learning. J. Parallel Computing **29**(7) (2003) 969–994
8. 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`.
9. Boehm, M., Speckenmeyer, E.: A fast parallel SAT-solver: Efficient workload balancing. Annals of Mathematics and Artificial Intelligence **17**(4-3) (1996) 381–400
10. Okushi, F.: Parallel cooperative propositional theorem proving. Annals of Mathematics and Artificial Intelligence **26**(1-4) (1999) 59–85
11. Speckenmeyer, E., Boehm, M., Heusch, P.: On the imbalance of distributions of solutions of CNF-formulas and its impact on satisfiability solvers. In: Satisfiability Problem: Theory and Applications, DIMACS (1997) 669–676
12. Blochinger, W., Westje, W., Küchlin, W., Wedeniwski, S.: ZetaSAT – Boolean satisfiability solving on desktop grids. In: CCGrid 2005, IEEE (2005) 1079–1086
13. Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. Automated Reasoning **24**(1/2) (2000) 67–100
14. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126**(1-2) (2001) 43–62
15. Eerola, P., et al.: Building a production grid in Scandinavia. IEEE Internet Computing **7**(4) (2003) 27–35
16. 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`.
17. 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
18. 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)