Counter Implication Restart for Parallel SAT Solvers

Tomohiro Sonobe and Mary Inaba

Graduate School of Information Science and Technology, University of Tokyo

Abstract. A portfolio approach has become the mainstream for parallel SAT solvers, making diversification of the search for each process more important. In the SAT Competition 2011, we proposed a novel restart method called counter implication restart (CIR), for sequential solvers and won gold and silver medals with CIR. CIR enables SAT solvers to change the search spaces drastically after a restart. In this paper, we propose an adaptation of CIR to parallel SAT solvers to provide better diversification. Experimental results indicate that CIR provides good diversification and its overall performance is very competitive with state-of-the-art parallel solvers.

1 Introduction

The Boolean satisfiability (SAT) problem asks whether an assignment of variables exists that can evaluate the given formula as true. A SAT problem is one of NP-complete problems. A formula is given in Conjunctive Normal Form (CNF), which is a conjunction of clauses. A clause is a disjunction of literals, where a literal is a positive or negative form of a variable. The solvers for this problem are called SAT solvers. The recent innovations in SAT solvers are significant and these solvers are used in many real applications, such as circuit design and software verification.

Many SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. In the last decades, conflict-driven learning and backjumping, Variable State Independent Decaying Sum (VSIDS) decision heuristic, and restart were added to DPLL, which improved the performance of DPLL solvers tremendously. These solvers are called Conflict Driven Clause Learning (CDCL) solvers. This kind of solver is now standard and it appears to be difficult to make a drastic improvement without a replacement of the fundamental algorithm.

Due to recent developments in multi-core hardware, we can easily run SAT solvers in parallel on standard PCs. However, there still appears to be a need for parallel SAT solvers. In the SAT Competition 2011, in the application category, the number of participants for the parallel category was about only ten, compared with more than 50 in the category of sequential solvers. Moreover, even though the parallel solvers were run on eight cores, the performance of the sequential solvers was very competitive with that of parallel solvers.

Many state-of-the-art parallel solvers are based on the portfolio approach [4]. In this approach, each solver runs competitively and they share learnt clauses

between them. Each solver uses a particular parameter set and conducts a differentiated but complementary search. This diversification is important for efficient searching [2]. Diversification is attained by employing, for example, differentiated restart policies [5], various strengths of saving literal polarity [6], decision heuristics, and so on.

In the SAT Competition 2011, we submitted a solver based on MiniSAT 2.2 [7] with our novel restart method, Counter Implication Restart (CIR). Our CIR enables SAT solvers to convert the search spaces by changing the decision-order after the restart, and thus enables an escape from desert search spaces [1]. This method is also valid for the diversification of the parallel SAT solver. In this paper, we propose the adaptation of CIR for use with parallel SAT solvers. Experimental results indicate that CIR also works efficiently in parallel solvers.

In Section 2, we explain the details of CIR. We show the experimental results in Section 3 and conclude the paper in Section 4.

2 Counter Implication Restart (CIR)

Existing restart policies only implement the restarting of the search from the beginning without changing anything. In many cases, this is sufficient to enable an escape from wrong branches. However, in some instances there are desert search spaces [1] where neither the solution nor the useful learnt clause exists. For such cases, it is difficult for SAT solvers to escape from these desert search spaces with a standard restart. Therefore, it is necessary to change the search activity after the restart drastically. CIR is a novel restart policy that consists of a standard restart and bumping the VSIDS scores to change the decision order after the restart. CIR traverses the implication graph [9] just before the restart, focusing on the indegrees of the variables.

A variable with a large indegree implies that this variable used to be the unit variable in a large clause. Let us consider the transformation from CSP to SAT. Suppose a variable a in the original CSP instance has a domain between 1 to n ($1 \leq a \leq n$), and its corresponding Boolean variables in the SAT instance are $a_1, a_2, ..., a_n$. There are clauses, $\prod_{1 \leq i < j \leq n} (\neg a_i \vee \neg a_j)$, that ensure at-most-one (AMO) constraint. In addition, there is one clause, $(a_1 \vee a_2 \vee ... a_n)$, that ensures at-least-one (ALO) constraint. In this setting, if any variables other than a are assigned to certain values and it causes the ALO clause for a to be unit clause by other constraints, such that only a variable $a_k (1 \leq k \leq n)$ is not assigned and the others are assigned to false, then a_k has $n - 1$ indegrees in the implication graph. Such variables like a are focused on by CIR and they are selected as decision-variables at early depth of the search tree. Before the execution of CIR, the assignments of such variables are forced by the values of other variables. However after CIR, they contribute early branching, and intuitively it enables the change of the search space.

The C-language-like pseudo code of the function of CIR is shown below. This function is called before the restart routine.

```
1. int run_count = 0;
```

```
 2. CounterImplicationRestart() begin
 3.   if (run_count++ % INTERVAL > 0)
 4.     int indegree[nVar] = {0};
 5.     int max_indegree = 0;
 6.     [calculate indegree for each variable and max_indegree]
 7.     for each variable var
 8.       bumpScore(var, BUMP_RATIO * indegree[var] / max_indegree);
 9.   restart();
10. end
```

The variable "run_count" stands for the number of times this function is executed. The main part of the function is run for every "INTERVAL" restart. In the seventh and eighth lines, all the VSIDS scores of the variables are bumped in proportion to their indegrees. To bump the VSIDS score drastically, the constant number of the "BUMP_RATIO" needs to be relatively large. So far, we have confirmed that the performance of CIR depends on the value of "INTERVAL" [8]. Fig. 1 shows the experimental result of various "INTERVAL" and fixed "BUMP_RATIO" using 200 instances from SAT Race 2008. From this result, We have found that small "INTERVAL" such as 3 is relatively better and it affects the total performance.

In the SAT Competition 2011, in the application category, we submitted MiniSAT 2.2 [7] with CIR, and won a gold medal in the minisat-hack track and a silver medal in the satisfiable problem track. Our solver could solve 202 instances in total - eight more than the original MiniSAT 2.2. The source code of this solver is available at http://www.cril.univ-artois.fr/SAT11/solvers/SAT2011-sources.tar.gz.

3 Experimental Results

We conducted experiments to confirm the performance of CIR for parallel solvers. In these experiments, the number of threads was set to four. As the first step, we implemented parallel settings of MiniSAT 2.2, called "para_minisat2.2", by using OpenMP. We modified the base number of Luby restart and the initial VSIDS scores of the variables for "para_minisat2.2", and added the function of learnt clause sharing. Then, we added the CIR top to "para_minisat2.2", called "para_cir_minisat". Three of the four threads used the CIR (the other ran as the default MiniSAT 2.2). In consideration of the previous results [8], the value of "INTERVAL" was set to 1, 2 and 3 respectively, and the "BUMP_RATIO" was fixed to 10000 for all of them.

The experiments were conducted on a Linux machine with an Intel Xeon quad-core CPU, running at 2.67 GHz and 24 GB of RAM. The benchmarks were 200 instances from SAT Race 2010. Timeout was set to 5000 seconds. We used six solvers: "para_minisat2.2", "para_cir_minisat", the latest version of Cryptominisat (denoted as "cryptominisat2.9.1"), the latest version of Plingeling (denoted as "plingeling276"), MiniSAT 2.2 in single thread (denoted as "minisat2.2_single"), and MiniSAT 2.2 with CIR whose "INTERVAL" is 3 in single thread (denoted as "cir_minisat_single").

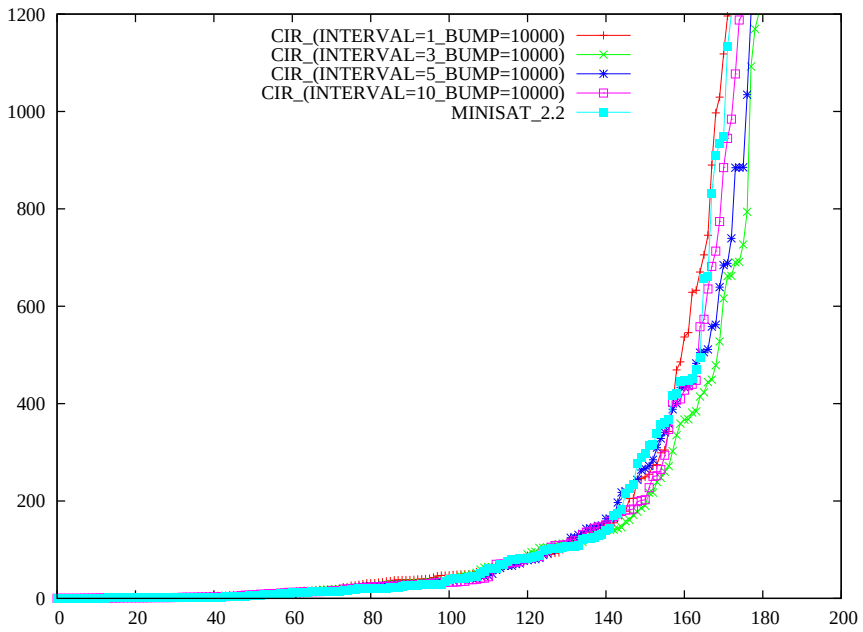

Fig. 1. The experimental result of various "INTERVAL" using 200 instances from SAT Race 2008.

The results are shown in Fig. 2 as a cactus plot and Table 1. Despite the naive and minimum configuration for parallel searching, "para_minisat2.2" provided good performance. In addition, "cir_minisat_single" is competitive with parallel solvers. The proposed solver, "para_cir_minisat" displayed relatively better performance than both "cryptominisat2.9.1" and "plingeling276", which won the first and second places in the SAT Competition 2011. This result indicates that CIR can also work in a parallel context and that CIR encourages the diversification of search activity in a portfolio approach.

4 Conclusion

CIR can convert the search space drastically after a restart. We propose the adaptation of CIR for parallel solvers in order to achieve good diversification. Experimental results show that CIR performed well for parallel settings, even though only simple functions were implemented. The vigorous conversion of the decision-order by CIR accelerates the diversification of search spaces. As future work, we will consider learnt clause sharing, such as clause length control [3] and arrange the scoring system of VSIDS so that it combines better with CIR.

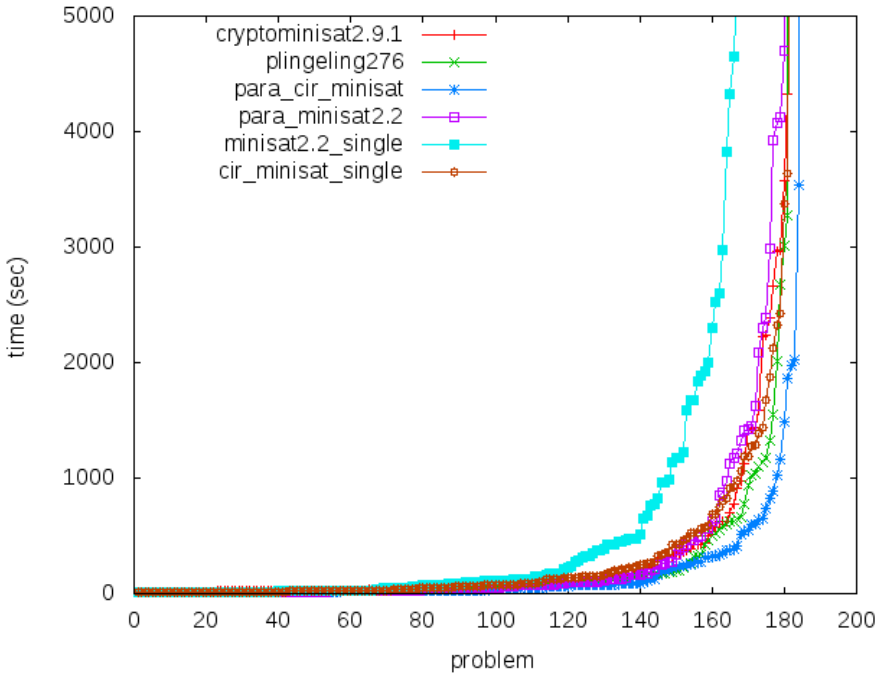

Fig. 2. The cactus plot of the experimental result using 200 instances from SAT Race 2010.

5 Acknowledgment

We appreciate the insightful comments from the reviewers in LION 6.

References

1. James M. Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. *In Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 1092–1097, 1994.
2. Long Guo, Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Diversification and intensification in parallel sat solving. In *Proceedings of the 16th international conference on Principles and practice of constraint programming*, CP'10, pp. 252–265, Berlin, Heidelberg, 2010. Springer-Verlag.
3. Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Control-based clause sharing in parallel sat solving. In *Proceedings of the 21st international joint conference on Artificial intelligence*, pp. 499–504, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
4. Youssef Hamadi and Lakhdar Sais. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2009.

Table 1. The number of solved instances for each solver.

	SAT	UNSAT	total
cryptominisat2.9.1	65	116	181
plingeling276	70	111	181
para_cir_minisat	69	115	184
para_minisat2.2	69	111	180
minisat2.2_single	66	106	172
cir_minisat_single	70	111	181

5. Jinbo Huang. The effect of restarts on the efficiency of clause learning. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2318–2323, 2007.
6. Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. *Proceedings of 10th International Conference on Theory and Applications of Satisfiability Testing(SAT)*, pp. 294–299, 2007.
7. Niklas Sorensson. Minisat 2.2 and minisat++ 1.1. *A short description in SAT Race 2010*, 2010.
8. Mary Inaba Tomohiro Sonobe and Ayumu Nagai. Counter implication restart. In *Pragmatics of SAT 2011*, 2011.
9. Lintao Zhang, Conor F. Madigan, and Matthew H. Moskewicz. Efficient conflict driven learning in a boolean satisfiability solver. In *ICCAD*, pp. 279–285, 2001.