2H5-OS-03b-3

Simple Method to Diversify Search Space in SAT Problems

Moon seongsoo Inaba Mary

Graduate School of Information Science and Technology, The University of Tokyo

Diversification of search space has contributed to the fast progress in SAT solving, and appears to be one of the most important keys in modern SAT solvers. It also plays an important role in portfolio-based parallel SAT solving. However, in portfolio solvers, maintenance of diversification among solvers is not that simple, especially for massively parallel machines. In this paper, we conducted several experiments to observe the variation of execution times when solving SAT problems. During the preprocessing, we shuffled indexes of variables randomly. This extremely simple preprocessing method shows some intriguing results, and we denote this with 'ISoV (index shuffling of variables)'.

1. Introduction

During the evolution of multicore hardware, many parallel SAT solvers have been proposed. The initial approach was divide-and-conquer[1], [2]. However it was difficult to find a successful load-balancing solution. In recent years, a portfolio-based approach[3], [4]. has become the mainstream of parallel SAT solvers. In portfolio approaches, the maintenance of the diversification and intensification tradeoff is very important[5]. Diversification is easily acquired with random noise, but it will lead to a different search space, and intensification will deteriorate. Our goal is to obtain extra diversity/intensity while maintaining the intensity/diversity of search. We propose a simple preprocessing method of index shuffling of variables, and we denote this with 'ISoV' in this paper. ISoV provides extra diversity, yet keeps intensity. We implemented this method in MiniSat 2.2[6], and checked variation of execution times to observe diversity. Intriguingly, this extremely simple method provided some diversification in execution time, and thus this might be helpful to create diversity for massively parallel environments.

2. Comparison of ISoV and MiniSat

2.1 About ISoV

Maintaining both diversification and intensification is the key in portfolio-based parallel SAT solvers. However, increased diversity in search may lead to weaken intensity. To obtain extra diversity, we tried an index shuffle of variables in preprocessing, and we denote this with 'ISoV'. The index shuffle method does not destroy the structure of the problem, but it still changes the order of the variables, which carries the possibility of changing results.

We implemented ISoV in MiniSat 2.2, the latest version of MiniSat released. We'd compared the execution time of ISoV with the original MiniSat, and obtained revised results for some problems. Though ISoV can be applied to any solver to diversify search space, we chose MiniSat to verify our method. As MiniSat is the most commonly used SAT solver, many solvers have been developed based on MiniSat. This means any refinements within MiniSat will be available to many other solvers.

Contact: Moon seongsoo, Creative Informatics (Univ of Tokyo), Bunkyo-ku, Tokyo Yayoi 1-1-1 University of Tokyo Graduate School of Information Science and Technology, Graduate School of Creative Information Department of I-REF Building 4F, Tel: 03-3812-2111, E-mail address: logic85@hotmail.com

2.2 Experimental Results

We implemented ISoV in MiniSat 2.2. We used a problem set of 300 industrial benchmarks from the SAT Competition 2014. We used Xeon X5680 3.3 GHz with 140 GB RAM for this experiment. Each instance was measured 12 times (1 MiniSat + ISoV 11 times), and if we choose best from the 12 results, this would indicate the approximate cost when this method is implemented on a parallel solver with 12 threads. The execution time-out limit was set to 3600 seconds on each instances. Figure 1 shows the comparison between MiniSat and ISoV on each problem. The X-axis represents each instance and the Y-axis represents the time needed to solve each instance. Each y value on the ISoV line represents each instance's best time from 11 ISoV results since we want to compare the execution time between the orginal MiniSat and ISoV. ISoV shows a faster time than MiniSat in many instances, but also exhibited slower time in some instances. This is because we simply shuffled indexes, meaning 12 cases are equivalent in performance. Consequently any one of the 12 cases could be the best solution for each instance. This figure's curve also indicates that ISoV was diversified in at least the execution time. MiniSat solved 162 problems in total. ISoV solved 183 problems with an additional 21 problems which were unsolved by MiniSat. The solving time was reduced in 105 problems. Figure 2 shows a comparison for MiniSat and the best time from 12 results (MiniSat + ISoV). This means each y value on the MiniSat + ISoV line represents the time needed to solve that instance if we ran in parallel. Thus, this figure compares 12 hypothetical processes with MiniSat, and the time amount of the space between the 2 lines will be improved if ISoV is parallelized.

In figure 3, we compare best time, worst time, and average time for each instance. We sorted data in ascending order along the average time. When average time is infinity, date is sorted in ascending order along how many

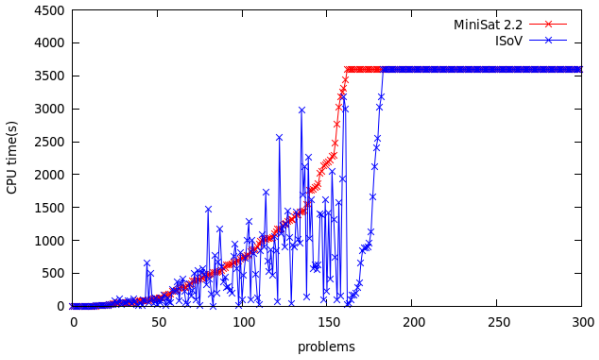

Fig. 1: Comparison between MiniSat and ISoV on each problem

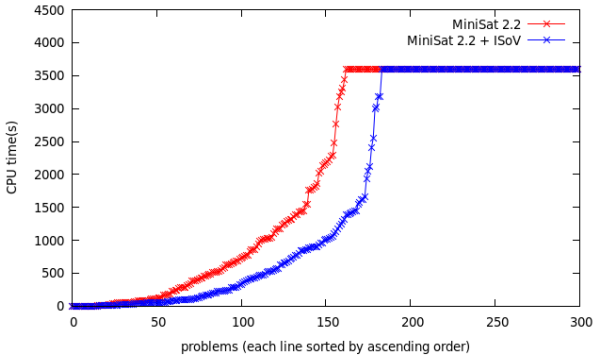

Fig. 2: Comparison between MiniSat and hypothetical 12 processes

times it reached the time-out limit. Variance seems very high when comparing the difference between the best time and worst time for each instance. For some instances, while the worst time reaches the time-out limit, but best time indicates within seconds. At the very least, we can assume execution time will be extremely different for some types of SAT problems when we use ISoV.

2.3 Why did ISoV change execution time?

Initially we didn't expect this simple method would change cost for proof, because most state-of-the-art SAT solvers are using VSIDS[7] . Therefore, decision-making will be made by the most frequently learnt order regardless of the variable's index. However, there exists the problem of choosing a variable from ones with the same VSIDS score. In MiniSat, the variable with the highest VSIDS score is chosen according to the order in which it is added to the heap. We assumed ISoV changes for this order, and the changed order resulted in diversity. Since ISoV does not deform the traits of the problem, it doesn't lower the performance of VSIDS. To demonstrate this assumption, we experimented with some instances, and the results are stated in the next section.

3. Picking a variable in MiniSat

3.1 Method in MiniSat2.2

VSIDS is a decision heuristics to pick a variable, and it is implemented in MiniSat. When a clause is added to learnt clauses, the score of each variable in the clause increases. The score order is managed through the heap array, and the highest one is chosen at each decision level. At this time, ties are broken randomly according to Chaff[7]. Although configurable, MiniSat simply choose the first index in the order heap.

3.2 Picking from ties

We modified MiniSat to choose a variable randomly from ties. We anticipated this would lead to results similar to

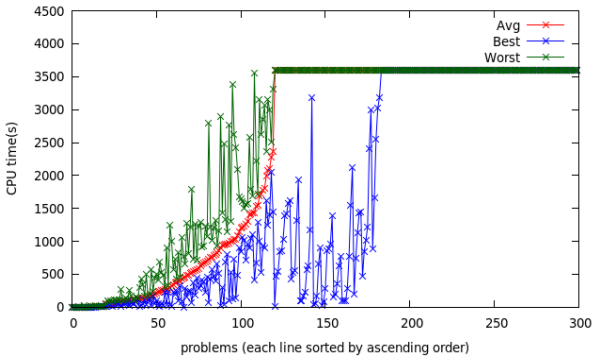

Fig. 3: Comparison among worst, best and average cases

the ISoV results. We used a problem set of 21 industrial benchmarks which are solved using ISoV and unsolved in the original MiniSat. Figure 3 shows the comparison between ISoV and the shuffle ties. We thought there would be an overhead time of O (ties number $\times$ log (heap_size)) to check ties. However according to these limited 21 results, there seems not to be a huge performance difference between these methods.

4. Concluding Remarks

We introduced ISoV and explained why this method diversifies execution time. This method seems to very similar to shuffling ties in VSIDS, and we verified this with several benchmarks.

The method of shuffling ties in VSIDS requires extra time on every decision of a variable, and especially the costs would be too big when a lot of ties happened. However,

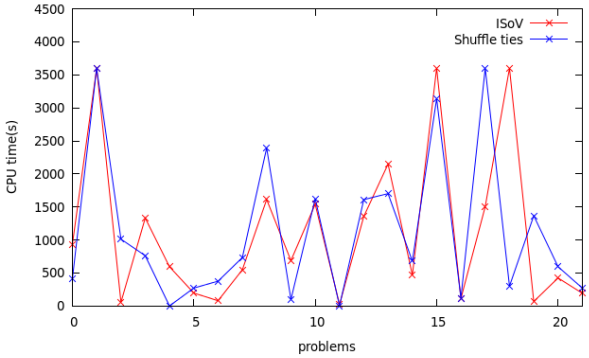

Fig. 4: Comparison between ISoV and shuffle ties

ISoV only shuffles once in pre-processing, and additional costs are unnecessary. Therefore, this method may be useful where the structure of massive ties happens frequently.

In our experience, the random strategy is inefficient for industrial problems because there is the possibility of ignoring the hidden structure in SAT problems. Our experiments are simply choosing a variable randomly. Our next step is the extraction of a meaningful variable from ties instead of operating on random decisions. In order to do this, we have to create a difference among VSIDS in ties which might require another feature extraction.

References

[1] Zhang, H., Bonacina, M. P., Hsiang, J.: PSATO: a distributed propositional prover and its application to quasigroup problems. Journal of Symbolic Computation 21, 543-560 (1996).

[2] Chrabakh, W., Wolski, R.: GrADSAT: A parallel SAT solver for the grid. Tech. rep., UCSB CS TR N. 2003-05 (2003)

[3] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. ManySAT: a parallel SAT solver. JSAT, Vol. 6, No. 4, 245-262 (2009)

[4] Gilles Audemard, Bewnoit Hoessen, Said Jabbour, Jean-Marie Lagniez, and Cedric Piette. Revisiting clause exchange in parallel SAT solving. In Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing, SAT'12, 200-213 (2012)

[5] Long Guo, The Diversification and Intensification in Parallel SAT Solving, Principles and Practice of Constraint Programming - CP 2010 (2010)

[6] Niklas Sorensson. MINISAT 2.2 and MINISAT++ 1.1. A short description in SAT Race 2010 (2010)

[7] Matthew W. Moskewicz, Chaff: Engineering an Efficient SAT Solver, DAC '01 Proceedings of the 38th annual Design Automation Conference (2001)