Generating Diverse Solutions in SAT: Paper Addendum

Alexander Nadel

Intel Corporation, P.O. Box 1659, Haifa 31015 Israel
alexander.nadel@intel.com

Abstract. This document is an addendum to [1]. We complement Section 4 of [1] in two ways. First, a detailed analysis of the randomized algorithms is provided. Second, an explanation of the behavior of the quality functions of PGUIDE and PBCPGUIDE_100 is proposed. The reader should be familiar with the content of [1] up to and including Section 4.

1 Analyzing Randomized Algorithms

This section complements the analysis presented in Section 4 of [1] by analyzing the behavior of three randomized algorithms and comparing them to PRAND.

DPLL-BASED SAMPLING invokes the SAT solver k times to generate k models for the same input formula. The first assignment to a variable is random for each invocation of the SAT solver. DPLL-BASED SAMPLING was mentioned in [2], but we did not find any reference to work introducing it.

XOR-SAMPLE [3] invokes the SAT solver at least k times to generate k models. For each invocation, the initial formula is augmented with random XOR constraints, where an XOR constraint includes variables and, optionally, the constant 1. Adding an XOR constraint means enforcing that an odd number of elements in the constraint are satisfied. In the original definition, a variable is added to the XOR constraint with probability q and 1 is added with probability 1/2. One can also use XOR sampling, where the length and number of XOR constraints are predefined.

PRANDWEAK [4] (AllSAT-Sampling in [4]) is a compact DIVERSEkSET algorithm. It randomizes the polarity of a new decision variable only when a variable is selected for the first time or for the first time after a model.

Let us analyze the behavior of the randomized algorithms for DIVERSEkSET, including PRAND, on our instances. Table 1 summarizes their behavior for a selected number of models and Fig. 1 presents their behavior as a function of the number of models.

In our experiments, we used 4 versions of XOR-SAMPLE, each one generating either 100 or 10 XOR constraints of length of 100 or 10. We generated the XOR constraints and translated them to clauses using the utilities of [3]. Note that additional variables must be introduced to translate XOR constraints to clauses. Compare the summary of the behavior of XOR-SAMPLE with that

Table 1: Mean quality and mean run-time for randomized DIVERSEkSET algorithms, given 100, 50 and 10 models on 66 benchmarks from semiformal verification of hardware. The algorithms are sorted by the quality obtained when generating 100 models.

	100		50		10	
Algorithm	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**
PRAND	0.1923	225	0.1915	206	0.1903	186
DPLL-BASED SAMPLING	0.1608	3988	0.1609	1992	0.1617	402
XOR-SAMPLE-100-100	0.1487	5067	0.1485	2559	0.1482	520
XOR-SAMPLE-100-10	0.1376	6855	0.1373	3430	0.1371	651
XOR-SAMPLE-10-100	0.1056	4117	0.1052	2059	0.1062	424
PRANDWEAK	0.09991	76	0.09405	55	0.08332	39
XOR-SAMPLE-10-10	0.08896	4126	0.08894	2076	0.08976	424

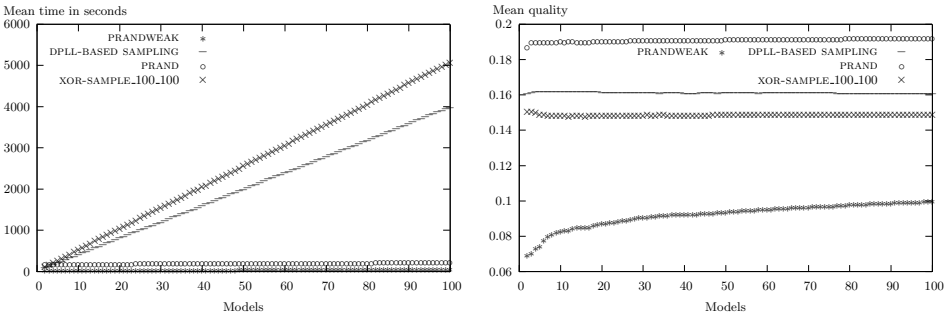

Fig. 1: Comparing randomized algorithms for DIVERSEkSET.

of DPLL-BASED SAMPLING in Table 1. The parameter values that follow XOR-SAMPLE in the first column of the table indicate the number of generated constraints and their length, respectively. The run-time of every configuration of XOR-SAMPLE is greater than that of DPLL-BASED SAMPLING and the quality is lower than that of DPLL-BASED SAMPLING. Compare the behaviour of XOR-SAMPLE_100_100 (the optimal configuration of XOR-SAMPLE in terms of quality) with that of DPLL-BASED SAMPLING on Figure 1. One can see that while the gap in the quality remains steady, the performance gap between the two algorithms grows. Our results reflect the following drawbacks of XOR-SAMPLE when applied to large instances of the DIVERSEkSET problem. First, adding clauses and variables to the instance makes the problem harder for the solver. Second, adding XOR constraints might disorient the SAT solver's heuristics in the sense that the solver would not be local with respect to the original structure of the input formula. Third, after XOR constraints are added the formulas might become unsatisfiable (in our experiments, we reinvoked the solver with a different set of XOR constraints on such occasions). In addition, it is unclear how to generalize XOR-SAMPLE to a compact algorithm. This is in contrast to DPLL-BASED SAMPLING that can easily be generalized. Indeed, PRAND and PRANDWEAK can be thought of as the result of such generalization.

Compare now the performance of DPLL-BASED SAMPLING and PRAND in Figure 1. The quality of PRAND is always better than that of DPLL-BASED SAMPLING. This behavior is expected, since PRAND's randomization strategy is much more aggressive; it randomizes the polarity of every selected variable, while DPLL-BASED SAMPLING randomizes only the first polarity for each variable. Now compare the run-time of both algorithms. Two factors are expected to impact the run-time. On the one hand, PRAND should be faster than DPLL-BASED SAMPLING since it invokes the SAT solver only once and hence keeps reusing all the learned clauses and heuristical information. On the other hand, DPLL-BASED SAMPLING could be faster than PRAND since PRAND's aggressive polarity randomization strategy effectively cancels the phase-saving polarity heuristic. In practice, the first factor clearly dominates. Indeed, DPLL-BASED SAMPLING runs faster only when the number of models is smaller than 4. The performance gap between PRAND and DPLL-BASED SAMPLING quickly increases; Table 1 shows that for 100 models PRAND is 18 times faster.

Consider the performance of PRANDWEAK. PRANDWEAK is undoubtedly the fastest algorithm. However, the quality of PRANDWEAK is worse than the quality of every other algorithm with the exception of one configuration of XOR-SAMPLE. These results are not surprising. PRANDWEAK is so fast because it makes only minimal changes to the run-time-efficient default strategies of the SAT solver. The quality of PRANDWEAK is low because its weak randomization strategy is insufficient for guiding the solver to different subspaces. As reported in [4], PRANDWEAK can be used for solving extremely difficult test-cases.

2 Explaining the Behavior of Polarity-based Guided Algorithms

This section complements the analysis presented in Section 4 of [1] by proposing an explanation of the behavior of the quality of PGUIDE and PBCPGUIDE_100 as a function of the number of models in Fig. 2 of [1]. First, we analyze the behavior of PGUIDE given a tautological formula. Second, we show how our analysis can be used for explaining the empirical behavior of PGUIDE on real world formulas. We then explain the difference between the empirical behaviors of PGUIDE and PBCPGUIDE_100. The following proposition determines the quality function of PGUIDE.

Proposition 1. *Suppose that* PGUIDE *is invoked on a tautological formula. Then it holds that:*

1. $Q_2 = 1$.
2. *For an odd* $m > 2$*:* $Q_m = \frac{m+1}{2m}$.
3. *For an even* $m > 2$*:* $Q_m = \frac{m}{2(m-1)}$.

Proof. Given a tautological formula, PGUIDE will assign each variable a value, independently of the other variables since there is no BCP or conflict analysis. Each variable assignment is guaranteed to be the eventual assignment of the

variable in a model. Consider the values given by PGUIDE to a variable v. For $m = 2$, the variable quality is clearly $Q_2 = 1$, since the second value assigned to v must be the negation of its value in the first model according to the definition of PGUIDE. One can see every odd value μ_i^u for $i > 2$ will be assigned randomly with probability 1/2, while every even value μ_i^u for $i > 1$ has to be the negation of the previous value μ_{i-1}^u. Hence the number of assigned 1's and 0's for any variable, given an even number of models is always equal.

For an even $m > 2$ we have: $S_m^u = p_m^u \times n_m^u = (m/2) \times (m/2) = m^2/4$, independently of u. Hence, for an even $m > 2$ it holds that $Q_m = \frac{n \times S_m^u}{\binom{m}{2} \times n} = \frac{m}{2(m-1)}$.

Consider an odd $m > 2$. It holds that $n_{m-1}^u = p_{m-1}^u$, since $m-1$ is even. By definition, we also have that $n_{m-1}^u = p_{m-1}^u = (m-1)/2$. The recursive definition of variable quality yields that $S_m^u = S_{m-1}^u + \frac{m-1}{2} = \frac{(m-1)^2}{4} + \frac{m-1}{2} = \frac{(m-1)(m+1)}{4}$. Hence, $Q_m = \frac{n \times S_m^u}{\binom{m}{2} \times n} = \frac{2(m-1)(m+1)}{4m(m-1)} = \frac{m+1}{2m}$. □

Prop. 1 implies that the image of the function Q_m for $m > 1$ of PGUIDE, given a tautological formula, looks as follows: $\{1, 2/3, 2/3, 3/5, 3/5, 4/7, 4/7, 5/9, 5/9, 6/11, 6/11, \ldots\}$. It is instructive that the behavior of the quality function of PGUIDE, given a tautological formula, seems to be similar to the empirical behavior of PGUIDE on our large well-structured real-world formulas. Indeed, the image is large for $m = 2$; it goes down quickly and approaches an asymptote in both cases (or at least seems to approach an asymptote in our experiments). We applied gnuplot's implementation of the nonlinear least-squares Marquardt-Levenberg algorithm to fit the quality function of PGUIDE into a curve over a space of parameters of the function $a + b\frac{x+1}{2x}$. The latter function was chosen, since it is similar up to parameters a and b to the quality function of PGUIDE, given a tautological formula. If our data fits into the curve well, it would be strong evidence that the behavior of the quality function of PGUIDE on tautological formulas is similar to its behavior on our real-wold formulas. The resulting function FITPGUIDE$(x) = 0.049 + 0.29(x + 1)/(2x)$ is shown in Fig. 2. One can see that the empirical data fits into the curve very well.

Apparently, Prop. 1 should be applicable to PBCPGUIDE, since BCP has no impact on a tautological formula. We tried to fit the quality of PBCPGUIDE_100 into the curve $a + b\frac{x+1}{2x}$. The resulting function FITPBCPGUIDE$(x) = 0.083 + 0.25(x + 1)/(2x)$ is displayed in Fig. 2. One can clearly see that the data of PBCPGUIDE_100 does not fit into the curve. In particular, FITPBCPGUIDE goes down asymptotically, while the real quality function goes up. This result is explained by the fact that unlike PGUIDE, PBCPGUIDE takes into consideration dependencies between variables.

Additional analysis to check whether or not the curves that characterize the behavior of PGUIDE/PBCPGUIDE_100 on tautological formulas fit the empirical data is beyond the scope of the present work. Our results indicate this to be a very interesting direction for future research.

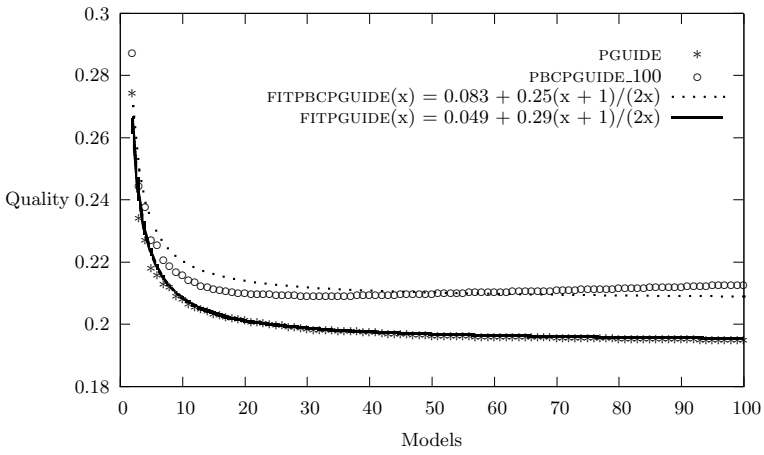

Fig. 2: Fitting polarity-based guided algorithms into curves.

References

1. Nadel, A.: Generating diverse solutions in SAT. In Sakallah, K.A., Simon, L., eds.: Theory and Applications of Satisfiability Testing - SAT 2011, 14th International Conference, Ann Arbor, USA, June 19-22, 2011, to Appear in Proceedings. (2011)
2. Kitchen, N., Kuehlmann, A.: Stimulus generation for constrained random simulation. In: ICCAD. (2007) 258–265
3. Gomes, C.P., Sabharwal, A., Selman, B.: Near-uniform sampling of combinatorial spaces using XOR constraints. In Schölkopf, B., Platt, J.C., Hoffman, T., eds.: NIPS. (2006) 481–488
4. Agbaria, S., Carmi, D., Cohen, O., Korchemny, D., Lifshits, M., Nadel, A.: SAT-based semiformal verification of hardware. In Bloem, R., Sharygina, N., eds.: Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2010). (October 2010) 25–32