Whose side are you on?
Finding solutions in a biased search-tree

Marijn J.H. Heule* and Hans van Maaren

Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Sciences
Delft University of Technology

marijn@heule.nl, h.vanmaaren@tudelft.nl

Abstract. We introduce a new jumping strategy for satisfiability (SAT) solvers that aims to boost their performance on satisfiable formulas, while maintaining their behavior on unsatisfiable instances. The direction heuristics used in various state-of-the-art SAT solvers bias - for some families in an observable useful way - the location of solutions in the search-tree. Based on the distribution of the solutions over the leaf nodes, we estimated - using small instances - the likelihood for each subtree to contain a solution. Larger formulas could then be solved by visiting these subtrees in descending order of this likelihood.
This paper describes experiments with this new strategy - *distribution jumping* - on hard random 3-SAT formulas using the march_dl solver. The results show a significant speed-up on satisfiable instances, while the performance on unsatisfiable formulas is not affected.

1 Introduction

Various state-of-the-art satisfiability (SAT) solvers use *direction heuristics* to predict the sign of the decision variables: These heuristics choose, after the decision of the branch variable, which Boolean value to examine first. On some families these heuristics bias the location of solutions in the search-tree. Given a large family with many satisfiable instances, this bias can be measured on small instances.

The usefulness of this depends on what we call the *bias-extrapolation property*: Given direction heuristics of a specified solver, the observed bias on smaller instances extrapolates to larger ones. Notice that this notion depends on the *action* of a particular solver on the family involved - e.g. a solver with random direction heuristics could probably also satisfy the bias extrapolation property, but not in a very useful way. The estimated bias then could be used to consider a jumping strategy that adapts towards the distribution of the solutions. We refer to this strategy as *distribution jumping*.

* Supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.306

Other jump strategies have been developed for SAT solvers. The most used technique is the *restart strategy* [4]: If after some number of backtracks no solution has been found, the solving procedure is restarted with different decision varables. This process is generally repeated for an increasing number of backtracks. This technique could fix an ineffective decision sequence. A disadvantage of restarts is its potential slowdown of performance on unsatisfiable instances.

Another jumping strategy is the *random jump* [8]. Instead of jumping all the way to the root of the search-tree, this technique jumps after some backtracks to a random level between the current one and the root. This technique could fix a wrong chosen sign of some of the decision variables. By storing the subtrees that have been visited the performance on unsatisfiable instances is only slightly reduced.

Both these techniques are designed to jump out of a huge subtree in which the solver gets "trapped". Our proposed technique does not only jump out of such a subtree but also aims jumping towards a subtree with a high likelihood of containing a solution. Like the random jump strategy, distribution jumping only alters the signs of decision variables and has a comparable performance on the unsatisfiable instances.

In this paper, we present the usefulness of distribution jumping on uniform random 3-SAT formulas close to the phase transition density using the march_dl solver - which direction heuristics are size independent. These formulas are well studied, obvious candidates to satisfy the bias-extrapolation property, and one can easily generate many hard instances for various sizes. Therefore, this family seems interesting and suitable as a first experimental environment for our approach.

2 Direction heuristics

All state-of-the-art complete SAT solvers are based on the DPLL architecture [2]. This recursive algorithm (see algorithm 1) first simplifies the formula by performing unit propagation (see algorithm 2) and checks whether it hits a leaf node. Otherwise, it selects a decision variable x_i and splits the formula into two subformulas where x_i is forced - one for positive (denoted by $\mathcal{F}(x_i = 1)$) and one for negative ($\mathcal{F}(x_i = 0)$).

Two important heuristics emerge for this splitting: *direction heuristics* and *variable selection heuristics*. Both heuristics are merged in procedure SELECTDECISIONLITERAL. Variable selection heuristics aim to select a decision variable in each recursion step yielding a relatively small search-tree. Direction heuristics aim to find an satisfying assignment as fast as possible by choosing which subformula - $\mathcal{F}(x_i = 0)$ or $\mathcal{F}(x_i = 1)$ - to examine first. In theory, direction heuristics could be very powerful: If one always predict the correct direction, all satisfiable formulas will be solved in a linear number of steps.

Traditionally, SAT research tends to focus on variable selection heuristics. Exemplary of the lack of interest in direction heuristics is the use in minisat [3]: While this solver is the most powerful on a wide range of instances, it always

Algorithm 1 DPLL($\mathcal{F}$)

```
1: F := UnitPropagation( F )
2: if F = ∅ then
3:    return satisfiable
4: else if empty clause ∈ F then
5:    return unsatisfiable
6: end if
7: l_decision := SelectDecisionLiteral( F )
8: if DPLL( F(l_decision = 1) ) = satisfiable then
9:    return satisfiable
10: else
11:    return DPLL( F(l_decision = 0) )
12: end if
```

Algorithm 2 UnitPropagation($\mathcal{F}$)

```
1: while F does not contain an empty clause and unit clause y exists do
2:    satisfy y and simplify F
3: end while
4: return F
```

branches negatively - computes $\mathcal{F}(x_i = 0)$ first. An explanation for the effectiveness of these heuristics is probably due to the general encoding of the most SAT formulas.

The march_dl SAT solver uses direction heuristics based on the reduction caused by the decision variable [5]. The reduction from $\mathcal{F}$ to $\mathcal{F}(x_i = 0)$ and from $\mathcal{F}$ to $\mathcal{F}(x_i = 1)$ is measured by the number of clauses that are reduced in size without being satisfied. In general, the higher this reduction the higher the chance the subformula is unsatisfiable. Therefore, march_dl always branches first on the subformula which has the smallest reduction.

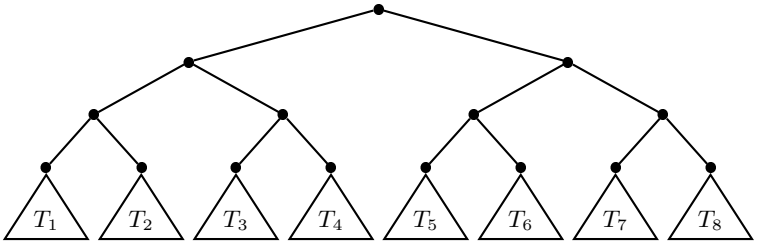

Fig. 1. A search-tree with jump depth 3 and thus 8 subtrees T_i

3 Distribution jumping

3.1 All subtrees distribution

We determined the biasedness of the location of solutions using the following experiment: Consider all the subtrees T_i which are at depth d. Assuming that the search-tree is big enough there are 2^d of these subtrees. Given a set of satisfiable formulas, how are the solutions distributed amongst the subtrees?

Let the left branch in a node denote the subformula - either $\mathcal{F}(x_i = 0)$ or $\mathcal{F}(x_i = 1)$ - which a solver decides to examine first. Consequently, we refer to the right branch as to the latter examined subformula. Subtrees are numbered from left to right (see figure 1 for an example with $d = 3$). We generated random 3-SAT formulas each with 350 variables and 1491 clauses (density = 4.26) and selected the first 5000 satisfiable instances for the test set.

Given a solver, jump depth d and a set of satisfiable formulas, we can compute for all 2^d subtrees the number of formulas that have at least one solution in T_i using the given solver. The histogram of march_dl, jump depth 12, and the test set is shown in figure 2. We refer to such a distribution as to the *all subtrees distribution*. The horizontal axis denotes the subtree number, while the vertical axis refers to the number of formulas which have at least one solution in T_i. The colors visualize the number of right branches that are required to reach a subtree: $T_1 = 0$, $T_2 = 1$, $T_3 = 1$, $T_4 = 2$, $T_5 = 1$ etc. The figure clearly shows that the distribution is biased towards the left branches: Most solutions are found in T_1 (zero right branches), followed by T_{2049} ($T_{2^{11}+1}$), T_{1025} ($T_{2^{10}+1}$), and T_{257} (T_{2^8+1}) - all reachable by one right branch. Based on the all subtrees distribution (figure 2) we can construct permutation $\pi_{\text{all}} = (1, 2049, 1024, 257, ...)$.

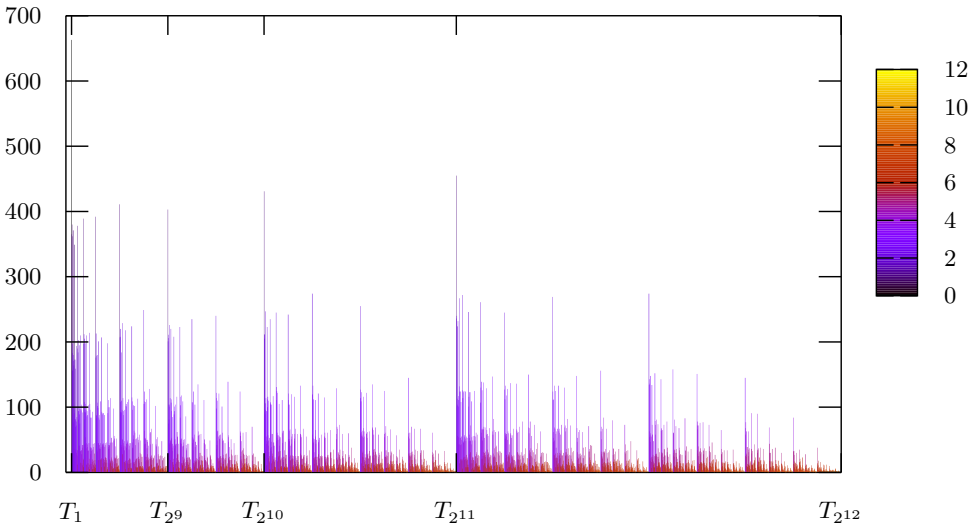

Fig. 2. All subtrees distribution with march_dl and jump depth 12 based on 5000 random 3-SAT formulas with 350 variables and 1491 clauses.

3.2 Implementation

Since we only computed π_{all} for jump depth 12, a more generalized permutation is required for an implementation with a flexible jump depth. An accurate approximation of π_{all} is obtained by visiting the subtrees in the following order: First, visit T_1, followed by all T_i's which can be reached in only one right branch. Continue with all T_i's that can be reached in two right branches, etc. All T_i's which can be reached in the same number of right branches are visited in decreasing order of i. We refer to this permutation as π_{dec}. Table 1 shows the order in which the first ten subtrees are visited with π_{all} and π_{dec} using jump depth 12.

Table 1. Permutations in which subtrees can be visited with jump depth 12.

	1	2	3	4	5	6	7	8	9	10
π_{all}	T_1	T_{2049}	T_{1025}	T_{257}	T_{513}	T_{129}	T_{65}	T_3	T_{33}	T_9
π_{dec}	T_1	T_{2049}	T_{1025}	T_{513}	T_{257}	T_{129}	T_{65}	T_{33}	T_{17}	T_9

We implemented distribution jumping with π_{dec} in march_dl and call the new version march_dj. A fixed jump depth is not useful for a general purpose approach: For some formulas this depth will be too large so it will never be reached, on others the depth will be too low, resulting in huge subtrees. Therefore, we implemented some elementary heuristics realizing an instance dependend jump depth: Let b be the depth at which the first backtrack occurs, then the jump depth is set to $d = b - 7$. Although far from optimal, these heuristics seem to work appropriate during our first experiments.

3.3 First results

Large random 3-SAT formulas with density 4.26 are still hard for our first implementation. Therefore, we experimented on two smaller densities, although we did not verify whether the all subtrees distributions of these densities are comparable with the computed one.

Table 2. Number of solved instances within a timeout of 600 seconds

	march_dl		march_dj		R+AdaptNovelty+	
size	#SAT	#UNKNOWN	#SAT	#UNKNOWN	#SAT	#UNKNOWN
600 vars, 2400 cls	86	14	100	0	100	0
600 vars, 2460 cls	75	25	100	0	100	0
700 vars, 2800 cls	73	27	100	0	100	0
700 vars, 2870 cls	15	85	90	10	100	0

For our experiments we generated random 3-SAT formulas for four different sizes - 600 and 700 variables both at density 4.0 and 4.1 - to test the improvement realized by march_dj. We compared the performance of our solvers with R$^+$AdaptNovelty$^+$ [1] - an incomplete solver which was the strongest on these kind of formulas at the SAT 2005 competition [7].

Table 2 shows the results of this experiment. The progress from march_dl to march_dj is clear. However, R$^+$AdaptNovelty$^+$ is still better on these instances.

4 Theoretical progress

4.1 First subtree distribution

We also studied the theoretical progress that could be realized by distribution jumping. Let march_dl(π_j) be a theoretical SAT solver which is similar to march_dl but visits the subtrees using permutation π_j. Let $\pi_0 = (1, 2, 3, 4, ...)$, so march_dl is equivalent to march_dl(π_0). Consider the *first subtree distribution* of a set of formulas: Count for all subtrees the number of formulas for which the (first) solution is found in T_i using march_dl(π_j). Figure 3 shows the *first subtree distribution* for the test set using march_dl.

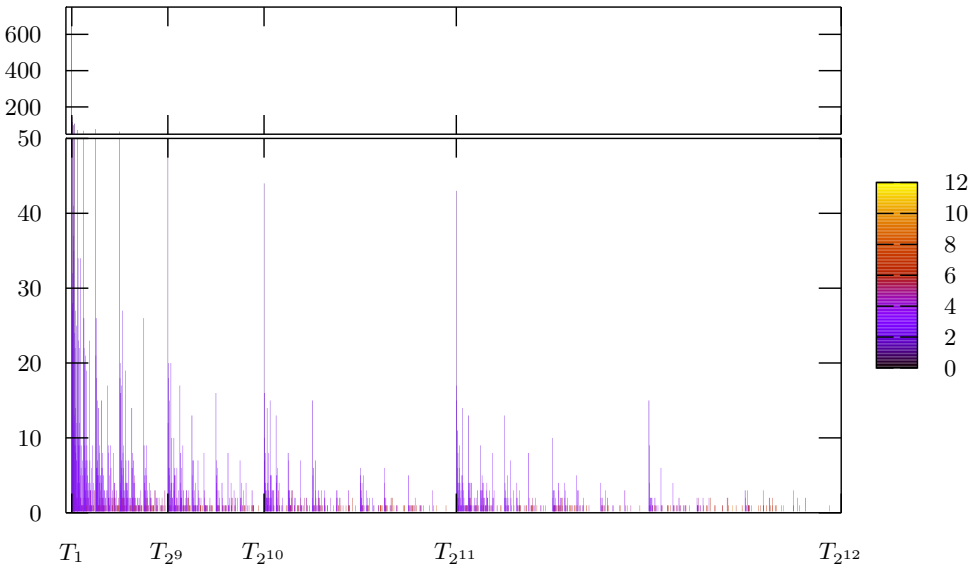

Fig. 3. The first subtree distribution using march_dl with jump depth 12 based on 5000 random 3-SAT formulas with 350 variables and 1491 clauses.

Notice that while using march_dl(π_j), the corresponding all subtrees distribution is a permutation of figure 2 with π_j. However, this does not hold for the first subtree distribution: The subtree in which the (first) solution will be found could be different for march_dl and march_dl(π_j).

Figure 4 shows the first subtree distribution of march_dl(π_{all}). Clearly, the (first) solutions are found far more to the left of the search-tree. Therefore, march_dl(π_{all}) is expected to be much faster on this test set compared to march_dl. However, the histogram is not strictly decreasing: Therefore, we can conclude that there is a permutation which has an even better expected performance.

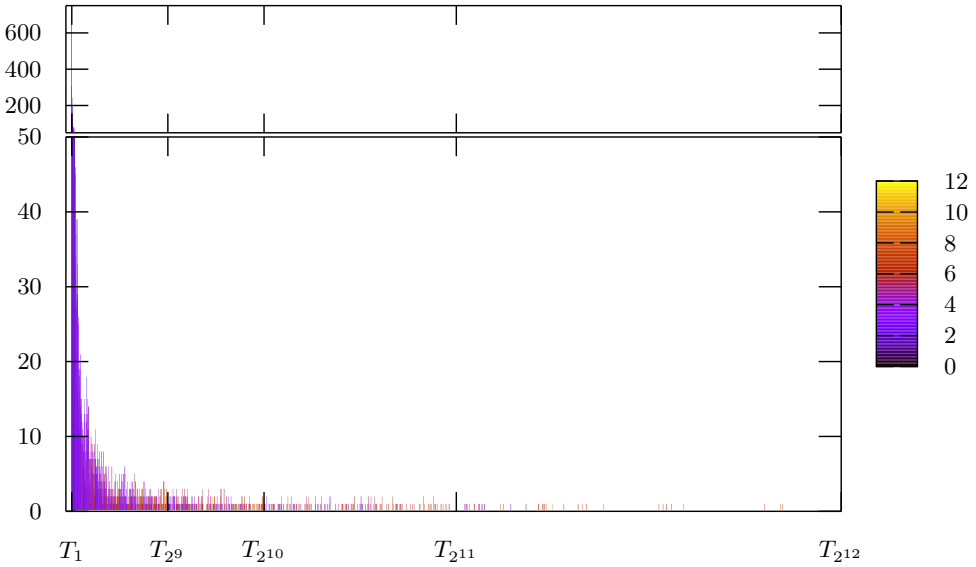

Fig. 4. The first subtree distribution using march_dl(π_{all}) with jump depth 12 based on 5000 random 3-SAT formulas with 350 variables and 1491 clauses.

4.2 Expected speed-up

The speed-up realized by distribution jumping can be approximated if the following two restrictions are met:

R_1) the size of the subtrees is about equal;
R_2) the jump depth is much smaller than the size of subtrees.

Due to R_1 the expected computational cost of each subtree is equal and R_2 marginalizes the overhead costs - getting from one subtree to another - of the distribution jumping technique. Using march_dl, the search-trees for hard random 3-SAT appear to be quite balanced (satisfying R_2). Given a relatively small jump depth the speed-up could be computed. However, R_1 and R_2 are probably hard to achieve for more structured formulas.

Given a large set of satisfiable benchmarks. Let $FT_{i,d}(\pi_j)$ denote the set of benchmarks which (first) solution is found in T_i using march_dl(π_j) with jump depth d. Now, we can compute the speed-up of using distribution jumping with jump depth d and permutation π_j:

$$\text{Speedup}_d(\pi_j) := \frac{\sum_{i=1}^{2^d} i \times |FT_{i,d}(\pi_0)|}{\sum_{i=1}^{2^d} i \times |FT_{i,d}(\pi_j)|} \quad (1)$$

On the test set $\text{Speedup}_{12}(\pi_{\text{all}}) = \frac{3197607}{810243} \approx 3.95$.

5 Conclusions

We explained the concept of distribution jumping. By using this technique in our solver march_dl, we are able to solve - within the experimented domain - satisfiable instances much faster, while the performance on unsatisfiable formulas remains practically equal.

Although we see significant progress on random 3-SAT instances, incomplete solvers are still much more powerful on these kind of benchmarks. However, on some structured families with many satisfiable instances, march_dl already outperforms the incomplete solvers, so even better results are expected using the new technique - if they would satisfy the bias-extrapolation property.

The usefulness of distribution jumping could be further increased by developing better direction heuristics: The more biased the distribution, the better the expected speed-up. Also some additional progress may be realized by using a permutation that has a higher expected speed-up than the implemented one.

Acknowledgments

This paper is a preliminary version of an extended paper of joint work with Samuel Burri, Stephan van Keulen, and Oana Dragomyr.

References

1. Anbulagan and Nghia Duc Pham, *The R+AdaptNovelty+ SAT solver*. http://nicta.com.au/director/research/programs/lc/people/a_anbulagan.cfm
2. M. Davis, G. Logemann, and D. Loveland, *A machine program for theorem proving*. Communications of the ACM **5** (1962), 394–397.
3. N. Een and N. Sorensson, *An extensible SAT-solver*. In SAT 2003, LNCS **2919** (2003), 502–518.
4. C.P. Gomes, B. Selman, and C. Crato, *Heavy-tailed Distributions in Combinatorial Search*. Principles and Practice of Constraint Programming, CP-97, LNCS **1330** (1997), 121–135.
5. M. Heule and H. van Maaren, *March_dl: Adding Adaptive Heuristics and a New Branching Strategy*. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006), 47–59.
6. H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, B. Selman, *Dynamic Restart Policies*. Proceedings of AAAI02 (2002), 674-682.
7. The SAT 2005 Competition. http://www.satcompetition.org/2005
8. H. Zhang, *A Complete Random Jump Strategy with Guiding Paths*. Proceedings of SAT 2006 (2006), 96–101.