Novelty Search for SAT

Roujia Wen
Minerva Schools at KGI
roujia@minerva.kgi.edu

Josh Grochow
Santa Fe Institute
jgrochow@santafe.edu

***Abstract*—In contrast to traditional local search methods that greedily optimize for fitness, which can sometimes be deceptive and misleading, novelty search is an alternative approach that looks for novel behaviors as a means to reach global optimums. We applied this idea to solving the boolean satisfiability problem. Incorporating multi-objective optimization, our algorithm outperformed the traditional local search algorithm (WalkSAT) in hard instances with different problem sizes.**

1. Introduction

In constraint satisfaction problems, stochastic local search algorithms perform optimization through randomly changing a small part of the existing solution and adopts it if it has a higher fitness value than existing one. Fitness is given by the fitness function, also called the objective function. This family of algorithms is generally good at finding local optimums or approximations for the global optimal. If the goal is to find the true global optimum, it becomes difficult as the search process easily gets stuck in local optimums due to its greedy nature. Thus, allowing "sideways" or even sometimes "downhill" moves helps with finding the global optimum [1]. Other improvements have been made to help with this process. For example, in the boolean satisfiability problem, it has been shown that adding random walk to local search makes the algorithm probabilistically approximately complete [2], which means that, as the runtime approaches infinity, the probability of finding global optimum approaches one. In this type of algorithms, for every search step, there is probability p that, instead of making a greedy choice, the algorithm makes a random choice. However, being able to find a solution in finite time in theory is not always practically useful. The algorithm can still spend a long time moving from one local optimum to another before reaching the true solution.

In recent years, we have started to see a new approach in evolutionary computation called *novelty search* [3]. It is a type of non-objective search, meaning that it does not explicitly try to search for the ultimate objective. Novelty search seeks the newest behaviors instead of the "best" behaviors. The idea is that phenotypic behaviors often collapse into a small number of typical behaviors. By searching for novelty, we gradually exhaust the possibilities of the behavioral space and are likely to come across the true solution. Inspired by this idea, we wanted to apply novelty search to the boolean satisfiability (SAT) problem, which is a classic NP-complete problem [4] that is very representative of hard combinatorial search landscape and captures complexity of many real world problems.

In the SAT problem, a literal is defined as a variable or the negation of a variable, where variables can take either True or False values. A clause is a number of literals connected by the and operators. The input for a SAT problem instance is usually a boolean formula given in the *conjunctive normal form*—a number of clauses connected by the or operators. One example could be $(x_1 \wedge x_2 \wedge \neg x_4) \vee (x_4) \vee (\neg x_3 \wedge \neg x_1) \vee (\neg x_2 \wedge x_3 \wedge \neg x_4)$, where $\neg$ is the not operator. A *complete assignment* is a set of assignments for all variables in the formula. A *satisfying assignment* is a complete assignment that makes the formula $True$. In the example above, one satisfying assignment might be $\{x_1 = True; x_2 = False; x_3 = False; x_4 = True\}$.

SAT instances where all clauses have exactly three literals are called *3-SAT*. All SAT instances can easily be converted to 3-SAT. Therefore, in this paper we are only concerned with the 3-SAT problem. In addition, problem size m is the number of variables appeared in the formula. It has been experimentally shown that for random 3-SAT, problem instances with *clause-to-variable ratio* α of 4.267 are the hardest to solve [5].

Two main families of existing SAT solvers are the DPLL-based algorithms and the stochastic local search algorithms. DPLL (Davis–Putnam–Logemann–Loveland) [6] family of algorithms are essentially depth-first search with various optimizations such as unit propagation, pure literal elimination, randomized restarts [7], [8] and backjumping [9]. These algorithms are *complete* methods because they are able to determine the unsatisfiability of an instance after exhausting all combinations of assignments. On the other hand, stochastic local search algorithms are *incomplete* because they cannot prove an instance unsatisfiable. However, they usually cost much less running time and tend to scale better as the problem size increases.

In this paper, we present an algorithm that combines novelty search with multi-objective optimization and performed a series of experiments. We found that our algorithm outperforms WalkSAT at solving random 3-SAT instances with different number of variables.

2. Local Search Algorithms

2.1. GSAT

GSAT is one of the first stochastic local search algorithms for solving the SAT problem. It starts with a random assignment. For each search step, it tries to flip all variables, and find the one that leads to the most increase in the number of satisfied clauses. It also allows "sideways" moves—those that lead to neither increase nor decrease in the number of satisfied variables. When a few variables are tied, the algorithm chooses randomly among them. If no satisfying assignment is found after a certain number of steps, the algorithm restart the the search process. It has been experimentally shown that GSAT outperforms DPLL algorithms by an order of magnitude on hard random SAT instances [1].

2.2. WalkSAT

WalkSAT is another family of stochastic local search algorithm. Rather than checking all variables each step as GSAT does, it only selects from variables inside a randomly chosen unsatisfied clause. There are different strategies for selecting a variable from the clause. The one that we are using in our experiments is the "SKC" strategy, introduced by Selman, Kautz and Cohen [10]. The strategy starts by determining how many satisfied clauses are "broken" by flipping each variable, meaning the satisfied clauses become unsatisfied due to the flip. If there is a case where no satisfied clauses are broken, this variable, called a "freebie," will be selected. Otherwise, for some probability p, the algorithm will select a variable randomly. For probability $1 - p$, the algorithm will select the variable that minimizes the number of broken clauses. It has been found that WalkSAT with SKC strategy can handle random SAT formulas that are much larger than those can be handled by basic GSAT, GSAT with noise, and simulated annealing solvers [10].

Apart from the "SKC" strategy, there are many other more recently introduced strategies for selecting a variable within the clause. WalkSAT with tabu search [11] improves performance by avoiding flipping variables that have been recently flipped. Novelty [12] adds a heuristic that when the best choice happens to be the most recently flipped one, it is only flipped with a fixed probability. With the rest of the probability, the second-best variable is selected.

3. The Novelty Search Approach

3.1. Algorithm Description

We will define a few concepts in novelty search and multi-objective optimization before discussing the specifics of the implementation of our SAT solver.

Novelty search in the context of the SAT problem is essentially the process of finding the most different-looking phenotype. Here we consider each complete assignment to

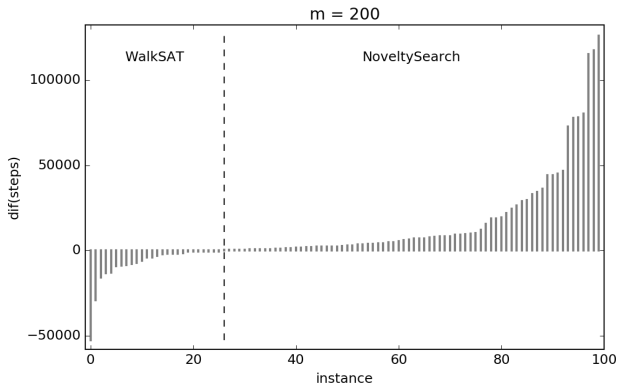

Figure 1: The difference of average number of steps between WalkSAT and novelty search for each distinct problem instance, based on 10 trials per instance. For a particular instance, a positive value indicates that novelty search outperforms WalkSAT. $m = 200$. $\alpha = 4.267$. Running time limit is fixed at 3000 seconds. WalkSAT: noise parameter $p = 0.55$. Novelty search: $c = 0.5$. $r = 30$.

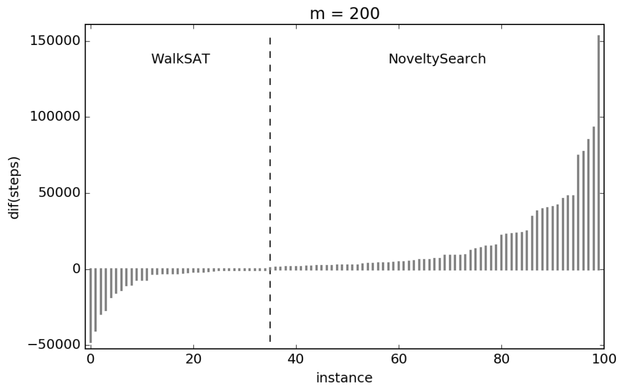

Figure 2: The difference of standard deviation.

be a genotype. The corresponding phenotype P is then defined as $[s_1, s_2, s_3 ... s_n]$ where s_i is the state of clause i given that specific assignment, with 1s being satisfied and 0s being unsatisfied. During the search process, we keep track of the most recently visited phenotypes and store them in the *phenotype library*, denoted by L. Then for any new phenotype P_{new} that we encounter, the degree of novelty N is the product of hamming distances d of P_{new} from each in the library. Hamming distance between two strings is defined as the number of positions at which the corresponding symbols differ. Formally, the degree of novelty is given by the following term:

$$N = \prod_{i=1}^{|L|} d(P_{new}, P_i) \tag{1}$$

In multi-objective optimization, a solution is said to dominate another solution when one is better than the other in every single objective. A solution is called *Pareto optimal*,

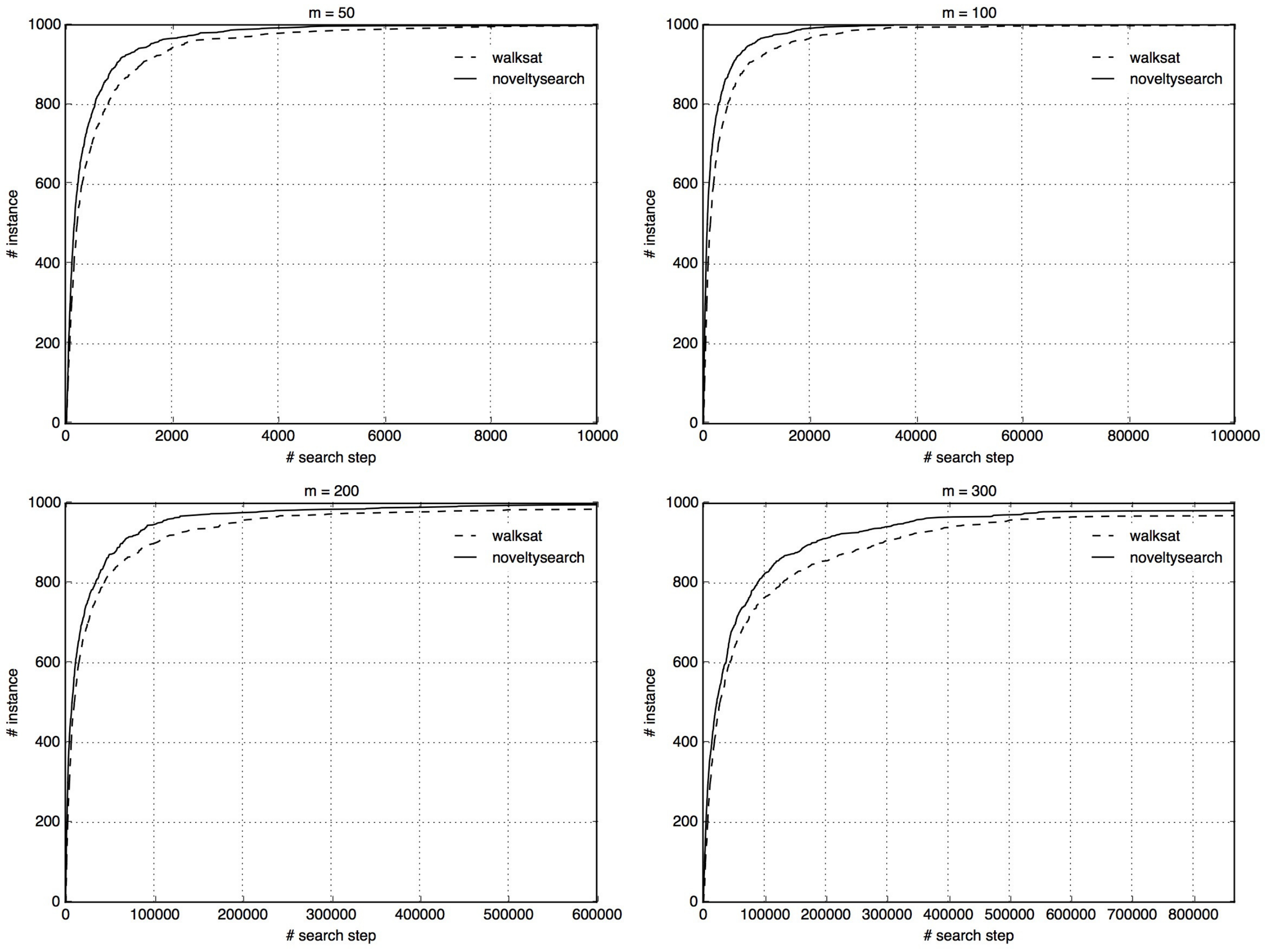

Figure 3: The number of random 3-SAT instances solved in a given number of steps, based on 100 different problem instances for each value of m and 10 trials per instance. $m = 50, 100, 200, 300$. $\alpha = 4.267$. Running time limit is fixed at 3000 seconds. WalkSAT: noise parameter $p = 0.55$. Novelty search: $c = 0.5$. $r = 30$.

if there is no other solution that dominates it. *Pareto front* is defined as the set of solutions that are Pareto optimal.

Our algorithm starts with a random assignment. For each search step, it takes all the variables that appeared in unsatisfied clauses, and tentatively flip each of them to get a collection of genotypic neighbors. For each of these neighbors, it calculates the degree of novelty of their corresponding phenotype, and in addition, fitness, which is the number of clauses they satisfy. In the next step, using techniques from multi-objective optimization, it selects only the neighbors that are in the Pareto front (we consider fitness and degree of novelty as our two objectives). Finally, it randomly chooses one from this selected group of neighbors as the base for the next iteration. If it does not find a satisfying assignment after $MAXITER$ steps, it restarts the search process.

3.2. Experimental Results

We tested our algorithm on 3-SAT instances with different problem sizes—50, 100, 200, and 300. For each category, we randomly generated 100 instances, and filtered out those that are unsatisfiable. We performed 10 different trials on each instance, and gave each run a maximum of 3000 seconds to finish.

Let c be the parameter so that the length of phenotype library $|L|$ is given by $c \times m$, where m is the problem size. We found that the optimal value for c is around 0.5. However, the optimal setting for the parameter $MAXITER$ is less clear-cut. We have not been able to discover a significant relationship between $MAXITER$ and the problem size. For the purpose of the experiment, we assume a linear relationship and let $MAXITER$ be $r \times m$ where r is fixed at 30.

For all categories, the novelty search algorithm outperforms WalkSAT (shown in Figure 3). Given a fixed amount of time, novelty search is able to solve more instances than WalkSAT. However, there is no significant difference in scaling behavior between the two.

More specifically, we looked at the data from the $m = 200$ category. We compared the two algorithms on individual instances and plotted the differences between means, and between standard deviations. As shown in Figure 1 and 2, on 25% of all instances, novelty search has smaller average running time, and on 35% of all instances, novelty search has a smaller standard deviation of running time than WalkSAT.

We have also tried looking into the detailed data from the $m = 300$ category. However, since a critical number of trials were forced to stop before they were able to find satisfying assignments, the summary statistics were skewed and we had to exclude them from further analysis.

4. Conclusions and Future Research

In this paper, we have shown the comparison between WalkSAT and our novelty search algorithm and found that novelty search outperforms WalkSAT at solving random 3-SAT instances with a range of problem sizes. While the WalkSAT family of algorithms have been extensively studied and tested, and many of its variants have significantly improved their efficiency using different heuristics [13], our algorithm is only at its earliest stage of development. Therefore, these preliminary results of the performance of novelty search seem quite promising. In future work, we hope to further optimize our our algorithm and compare it against the state-of-the-art SAT solvers. In addition to random instances, we would also like to apply our algorithm to real-world industrial instances and there is hope that novelty search can effectively exploit the structures of those instances.

References

[1] B. Selman, H. J. Levesque, D. G. Mitchell *et al.*, "A new method for solving hard satisfiability problems." in *AAAI*, vol. 92, 1992, pp. 440–446.

[2] H. H. Hoos, "On the run-time behaviour of stochastic local search algorithms for sat." in *AAAI/IAAI*, 1999.

[3] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.

[4] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 1971, pp. 151–158.

[5] S. Kirkpatrick, B. Selman *et al.*, "Critical behavior in the satisfiability of random boolean expressions," *Science-AAAS-Weekly Paper Edition-including Guide to Scientific Information*, vol. 264, no. 5163, pp. 1297–1300, 1994.

[6] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[7] C. P. Gomes, B. Selman, H. Kautz *et al.*, "Boosting combinatorial search through randomization," *AAAI/IAAI*, vol. 98, pp. 431–437, 1998.

[8] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman, "Dynamic restart policies," *AAAI/IAAI*, vol. 97, pp. 674–681, 2002.

[9] R. M. Stallman and G. J. Sussman, "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis," *Artificial intelligence*, vol. 9, no. 2, pp. 135–196, 1977.

[10] B. Selman, H. Kautz, B. Cohen *et al.*, "Local search strategies for satisfiability testing," *Cliques, coloring, and satisfiability: Second DIMACS implementation challenge*, vol. 26, pp. 521–532, 1993.

[11] B. Mazure, L. Saïs, and É. Grégoire, "Tabu search for sat," in *AAAI/IAAI*, 1997, pp. 281–285.

[12] D. McAllester, B. Selman, and H. Kautz, "Evidence for invariants in local search," in *AAAI/IAAI*, 1997, pp. 321–326.

[13] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Satisfiability solvers," *Foundations of Artificial Intelligence*, vol. 3, pp. 89–134, 2008.