GenSAT: A Navigational Approach

Yury Smirnov and Manuela M. Veloso

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891
{smir, mmv}@cs.cmu.edu

Abstract. GenSAT is a family of local hill-climbing procedures for solving propositional satisfiability problems. We restate it as a navigational search process performed on an N-dimensional cube by a fictitious agent with limited lookahead. Several members of the GenSAT family have been introduced whose efficiency varies from the best in average for randomly generated problems to a complete failure on the realistic, specially constrained problems, hence raising the interesting question of understanding the essence of their different performance. In this paper, we show how we use our navigational approach to investigate this issue. We introduce new algorithms that sharply focus on specific combinations of properties of efficient GenSAT variants, and which help to identify the relevance of the algorithm features to the efficiency of local search. In particular, we argue for the reasons of higher effectiveness of HSAT compared to the original GSAT. We also derive fast approximating procedures based on variable weights that can provide good switching points for a mixed search policy. Our conclusions are validated by empirical evidence obtained from the application of several GenSAT variants to random 3SAT problem instances and to simple navigational problems.

1 Introduction

Recently an alphabetical mix of variants of GSAT [6, 11] has attracted a lot of attention from Artificial Intelligence (AI) researchers: TSAT, CSAT, DSAT, HSAT [3, 5], WSAT [12], WGSAT, UGSAT [2] just to name few. All these local hill-climbing procedures are members of the GenSAT family. Propositional satisfiability (SAT) is the fundamental problem of the class of NP-hard problems, which is believed not to admit solutions that are always polynomial on the size of the problems. Many practical AI problems have been directly encoded or reduced to SAT. GenSAT (see Table 1) is a family of hill-climbing procedures that are capable of finding satisfiable assignments for some large-scale problems that cannot be attacked by conventional resolution-based methods.

GSAT [6, 11] is an instance of GenSAT in which *initial* (see Table 1) generates a random truth assignment, *hill-climb* returns all those variables whose flips[1] give the greatest increase in the number of satisfied clauses and *pick* chooses one of these variables at random [3]. Previous work on the behavior of GSAT and

[1] Flip is a change of the current value of a variable to the opposite value.

```
procedure GenSAT (Σ)
  for i:=1 to Max_Tries
    T:= initial(Σ)
    for j:=1 to Max_Flips
      if T satisfies Σ then return T
      else poss-flips := hill-climb(Σ, T)
          ; compute best local neighbors of T
        V := pick(poss-flips) ; pick a variable
        T := T with V's truth assignment inverted
    end
  end
return "no satisfying assignment found"
```

Table 1. The GenSAT Procedure.

similar hill-climbing procedures [3] identified two distinct search phases and suggested possible improvements for GenSAT variants. HSAT is a specific variant of GenSAT, which uses a queue to control the selection of variables to flip[2]. Several research efforts has attempted to analyze the dominance of HSAT compared with the original GSAT for randomly generated problem instances. We have developed a navigational search framework that mimics the behavior of GenSAT. This navigational approach allows us to re-analyze the reasons of higher effectiveness of HSAT and other hill-climbing procedures by relating it to the number of equally good choices. This navigational approach also suggests strong approximating SAT procedures that can be applied efficiently to practical problems. An approximation approach can be applied to both "easy" and "hard" practical problems, in the former case it will likely to produce a satisfiable assignment, whereas in the latter case it will quickly find an approximate solution. For a standard testbed of randomly generated 3SAT problems, the transition phase between "easy" and "hard" problem instances corresponds to the ratio value of 4.3 between the number of clauses L to the number of variables N [10, 1]. Figure 1 demonstrates the probability of generating a satisfying assignment for random 3SAT problems depending on the L/N-ratio.

An approximate solution can be utilized in problems with time-critical or dynamically changing domains. Interestingly, we found that it also provides a good starting point for a different search policy, i.e. serves as a switching point between distinct search policies within the same procedure. Such an approach can be utilized beneficially in multi-processor/multi-agent problem settings.

Our experiments with randomly generated 3SAT problem instances and realistic navigational problems confirmed the results of our analysis.

[2] See Section3 for the definition of HSAT.

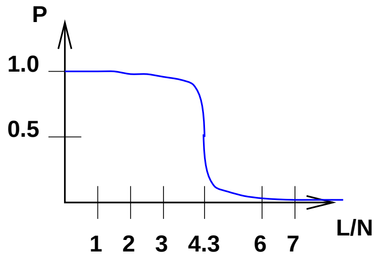

Fig. 1. The transition phase for random 3SAT problems.

2 GenSAT as an Agent-Centered Search

State spaces for boolean satisfiability problems can be represented as N-dimensional cubes, where N is the number of variables. We view GSAT and similar hill-climbing procedures as performing search on these high-dimensional cubes by moving a fictitious agent with limited lookahead. For efficiency reasons, the majority of GSAT-like procedures limit the lookahead of the agent to the neighbors of its current state, i.e., to those vertices of the cube that are one step far from the current vertex. An edge of the cube that links two neighboring vertices within the same face of the cube, corresponds to the flip of a variable. Thus, we reduced the behavior of GSAT to agent-centered search on a high-dimensional cube. Recall, in agent-centered search the search space is explored incrementally by an agent with limited lookahead. Throughout the paper we refer to this navigational version of GenSAT as to NavGSAT.

The worst-case complexity of both informed and uninformed agent-centered search is of the order of the number of vertices, i.e. $O(2^N)$. Moreover, unlike classical AI search where A* is an optimal informed algorithm for an arbitrary admissible heuristic, there are no optimal algorithms for agent-centered search problems[13]. Furthermore, even a consistent, admissible heuristic can become misleading, and an efficient informed agent-centered search algorithm can demonstrate worse performance than the uninformed (zero heuristic) version of the same algorithm [8].

From the algorithmic point of view, the behavior of LRTA* [9], one of the most efficient agent-centered search methods, is close to NavGSAT's behavior. Both methods look for the most promising vertex among neighbors of the current vertex. In addition to selecting a neighbor with the best heuristic value, LRTA* also updates the heuristic value of the current vertex (see Table 2). The efficiency of LRTA* depends on how closely the heuristic function represents the real distance [13]. The vast majority of GSAT-like procedures use the number of unsatisfied (or satisfied) clauses as the guiding heuristic. In general, this heuristic is neither consistent, nor admissible. However, for the most intricate random instances of SAT problems with $L = O(N)$, this heuristic is an $O(N)$ approximation of the real distance. Therefore, ϵ-search [7], a modification of LRTA* that uses approximations of admissible heuristics, applies to SAT problems.

procedure LRTA*(V, E)
Initially, $F(v) := h(v)$ for all $v \in V$.
LRTA* starts at vertex v_{start}:

1. $v :=$ the current vertex.
2. If $v \in Goal$, then STOP successfully.
3. $e := argmin_e F(neighbor(v, e))$.
4. $F(v) := max(F(v), 1 + F(neighbor(v, e)))$.
5. Traverse edge e, update $v := neighbor(v, e)$.
6. Go to 2.

Table 2. Learning Real-Time Algorithm (LRTA*). LRTA* also looks for the most promising vertex among neighbors of the current vertex.

Lemma 1. *After repeated problem-solving trials of a soluble propositional satisfiability problem with N variables and $O(N)$ clauses, the length of the solution of ϵ-search converges to $O(N^2)$.*

Proof: After repeated problem-solving trials the length of a solution of ϵ-search converges to the length of the optimal path multiplied by $(1 + \epsilon)$ [7]. On one hand, the length of the optimal path for a soluble propositional satisfiability problem is $O(N)$. On the other hand, for problems with $L = O(N)$ approximating factor ϵ is also $O(N)$. These two facts imply $O(N^2)$ complexity of the final solution after an unknown number of repeated trials. ■

Even though the length of a solution of ϵ-search converges to $O(N^2)$ for soluble problem instances, several initial trials can have exponential length. Thus, this approach can be applied only in special circumstances: One is provided possibly exponential memory and possibly exponential time for pre-processing to rebalance the heuristic values, then the complexity of solving of the pre-processed problem is $O(N^2)$. Since this scenario is not always what AI researchers keep in mind when applying GenSAT, we do not consider ϵ-search as a general navigational equivalent of GenSAT. However, in Section 3 we show that one (first) run of ϵ-search coincides completely with the run of HSAT for the majority of soluble SAT problem instances.

Thus, the question of the efficiency of GSAT and similar procedures is reduced to the domain-heuristics relations that guide agent-centered search on an N-dimensional cube. Recent works on changing the usual static heuristic – the number of unsatisfied (satisfied) clauses – to the dynamic weighted sums [2] produced another promising sub-family of GenSAT procedures. Our experiments showed that the "quality" of the usual heuristic varies greatly in different regions of the N-dimensional cube, and as the ratio of L to N grows, this heuristic becomes misleading in some regions of the problem's domain. These experiments identified the need to introduce novel heuristics and better analysis of the existing ones.

3 New Corners or Branching Factor?

We conducted a series of experiments with the ϵ-search version of LRTA* and the number of unsatisfied clauses as the heuristic values for each vertex (corner) of the N-dimensional cube. We found that the combination of a highly connected N-dimensional cube and such prior knowledge forces an agent to avoid vertices with updated (increased in step 4) heuristic values. Exactly the same effect has been achieved by HSAT - a variant of GenSAT. In HSAT flipped variables form a queue, and this queue is used in *pick* to break ties in favor of variables flipped earlier until the satisfying assignment is found or the amount of flips has reached the pre-set limit of Max_Flips. Thus, ϵ-search is a navigational analogue of HSAT for soluble problem instances.

Previous research identified two phases of GenSAT procedures: steady hill-climbing and plateau phases [3]. During the plateau phase these procedures perform series of sideway flips keeping the number of satisfied clauses on the same level. The reduction of the number of such flips, i.e. cutting down the length of the plateau, has been identified as the main concern of such procedures. Due to high connectivity of the problem domain and the abundance of equally good choices during the plateau phase, neither HSAT nor ϵ-search re-visit already explored vertices (corners) of the cube for large-scale problems. This property of HSAT has been stated as the reason of its performance advantage for randomly generated problems in comparison with GSAT [5].

To re-evaluate the importance of visiting new corners of the N-dimensional cube, we introduced another hill-climbing procedure, that differs from GSAT only in keeping track of all visited vertices and Never Re-visiting them again, NRGSAT. On all randomly generated 3SAT problems, NRGSAT's performance in terms of flips was identical to GSAT's one. Practically, NRGSAT ran much slower, because it needs to maintain a list of visited vertices and check it before every flip. Based on this experiment, we were able to conclude that exploring new corners of the cube is not that important. This increased our interest in studying further reasons for the performance advantage of HSAT over GSAT.

We focused our attention on *poss-flips* – the number of equally good flips between which GSAT randomly picks [4], or, alternatively, the branching factor of GSAT search during the plateau phase. We noticed that on earlier stages of the plateau phase both GSAT and NRGSAT tend to increase *poss-flips*, whereas HSAT randomly oscillates *poss-flips* around a certain (lower) level. To confirm the importance of *poss-flips*, we introduced *variable weights*[3] as a second heuristic to break ties during the plateau phase of NavGSAT. NavGSAT monitors the number of flips performed for each variable and among all equally good flips in terms of the number of unsatisfied clauses, NavGSAT picks a variable that was flipped the least number of times. In case of second-order ties, they can be broken either randomly, fair – NavRGSAT – or deterministically, unfair, according to a

[3] Weight of each variable is the number of times this variable has been flipped from the beginning of the search procedure. Each flip of a variable increases its weight by one.

fixed order – NavFGSAT.

Problem	Algorithm	Mean	Median	St.Dev.
100	GSAT	12,869	5326	9515
vars,	HSAT	2631	1273	1175
430	NavFGSAT	3558	2021	1183
clauses	NavRGSAT	3077	1743	1219
1000	GSAT	4569	2847	1863
vars,	HSAT	1602	1387	334
3000	NavFGSAT	1475	1219	619
clauses	NavRGSAT	1649	1362	675
1000	GSAT	7562	4026	3515
vars,	HSAT	3750	2573	1042
3650	NavFGSAT	3928	2908	1183
clauses	NavRGSAT	4103	3061	1376

Table 3. Comparison of number of flips for GSAT, HSAT, NavRGSAT and NavFGSAT.

Both NavRGSAT and NavFGSAT allow to flip back the just flipped variable. Moreover, the latter procedure often forces to do so due to the fixed order of variables. However, the performance of both NavRGSAT and NavFGSAT is very close to HSAT's performance. Table 3 presents median, mean and standard deviation of GSAT, HSAT, NavRGSAT and NavFGSAT for randomly generated 3SAT problems with 100 and 1000 variables and different ratios L to N. We investigated problems of this big size, because they represents the threshold between satisfiability problems that accept solutions by conventional resolution methods, for example Davis-Putnam procedure, and ones that can be solved by GenSAT hill-climbing procedures.

In the beginning of the plateau phase both NavGSAT methods behave similarly to HSAT: Variables flipped earlier are considered last when NavGSAT is looking for the next variable to flip. As more variables gain weight, NavGSAT methods' behavior deviates from HSAT. Both methods can be perceived as an approximation of HSAT.

We identified that a larger number of *poss-flips* is the main reason why GSAT loses to HSAT and NavGSAT on earlier stages of the plateau phase. As the number of unsatisfied clauses degrades, there are less choices for equally good flips for GSAT, and the increase of *poss-flips* is less visible. During earlier sideway flips GSAT picks equally good variables randomly, this type of selection leads to the vertices of the cube with bigger *poss-flips*, where GSAT tends to be "cornered" for a while. Figure 2 presents average amounts of *poss-flips* with the 95%-confidence intervals. The *poss-flips* were summed up for each out of four hill-climbing procedures for every step in the beginning of the plateau phases

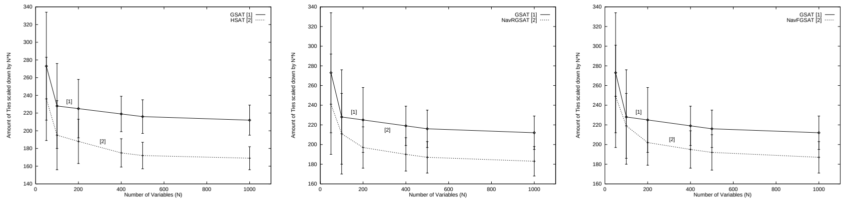

Fig. 2. Comparison of *Poss-Flips* for GSAT, HSAT, NavRGSAT and NavFGSAT.

(from $0.25N$ to N) for a range of problem sizes. Since the number of variables and the interval of measuring grow linearly on N, we present sums of *poss-flips* scaled down by N^2. As it follows from Figure 2, the original GSAT consistently outnumbers all other three procedures during that phase, although its confidence intervals overlap with NavRGSAT and NavFGSAT's confidence intervals.

Figure 3 presents the dynamics of *poss-flips* during a typical run of GSAT. It is easy to see that on early plateaux *poss-flips* tend to grow with some random noise, for example, in Figure 3 second, third and fifth plateaux produced obvious growth of *poss-flips* until drops corresponding to the improvement of the heuristic values and, thus, the end of the plateau. During the first and fourth plateaux, the growth is not that steady though still visible. Even though flips back are prohibited for NRGSAT, it maintains the same property, because of the high connectivity of the problem domain and the abundance of equally good choices.

Figure 4 represents the average percentage of ties for a 3SAT problem with 100 variables and 430 clauses over 100 runs for GSAT and HSAT, and for GSAT and NavRGSAT. The average number of *poss-flips* for GSAT dominates the analogous characteristic for HSAT by a noticeable amount. This type of dominance is similar in the comparison of GSAT with NavRGSAT in the beginning of the plateau phase. In the second part of the plateau phase the number of *poss-flips* for HSAT or NavRGSAT approaches the number of *poss-flips* for GSAT. Lower graph represents second-order ties for NavRGSAT that form a subset of *poss-flips*.

Our experiments confirmed the result obtained in [3] that the whole picture scales up linearly in the number of variables and the number of *poss-flips*. The plateau phase begins after about $0.2N - 0.25N$ steps. By that moment at most a quarter of the variable set has been flipped, and thus NavFGSAT mimics HSAT up to a certain degree. After $2N$ or $3N$ flips, both versions of NavGSAT diverge significantly from HSAT. After these many steps both NavRGSAT and NavFGSAT still maintain random oscillation of *poss-flips*, whereas GSAT tends to promote higher levels of *poss-flips*. Unfortunately, for problems with larger ratio of the number of clauses to the number of variables NavFGSAT is often trapped in an infinite loop. NavRGSAT also may behave inefficiently for such problems: From time to time the policy of NavRGSAT forces it to flip the same variable with a low weight several times in a row to gain the same weight as other variables from the set of *poss-flips*.

Thus, NavGSAT showed that the number of *poss-flips* plays an important

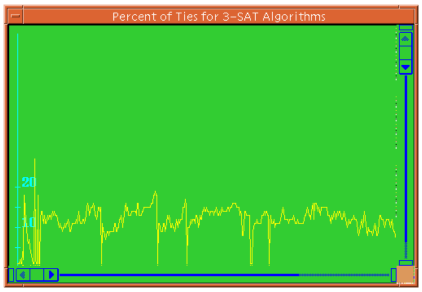

Fig. 3. Dynamics of *Poss-Flips* for GSAT.

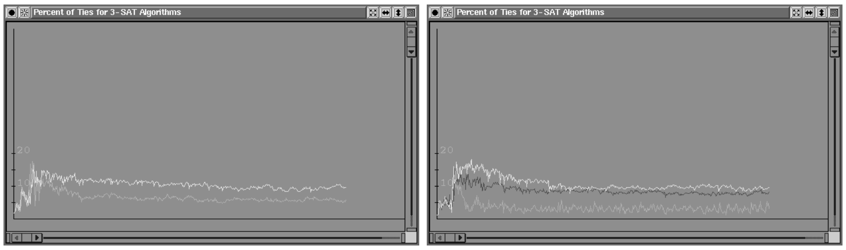

Fig. 4. Percentage of *Poss-Flips* for GSAT with HSAT and GSAT with NavRGSAT.

role in improving the efficiency of GenSAT procedures. HSAT capitalizes on this property and therefore constitutes one of the most efficient hill-climbing procedures for random problem instances. However, many real-world satisfiability problems are highly structured and, if applied, HSAT may easily fail due to its queuing policy. NavGSAT suggests another sub-family of GenSAT hill-climbing procedures that does not tend to increase the number of *poss-flips*. Weights of variables and their combinations can be used as a second tie-breaking heuristic to maintain lower level of *poss-flips* and find exact or deliver approximate solutions for those problems for which HSAT fails to solve.

For randomly generated 3SAT problems HSAT proved to be one of the most efficient hill-climbing procedures. There has been reports on HSAT's failures in solving non-random propositional satisfiability problems [5]. We view the non-flexibility of HSAT's queue heuristic as a possible obstacle in solving over-constrained problems. This does not happen in solving random 3SAT problems with low L/N-ratio.

4 Approximate Satisfaction

While running experiments with GSAT, HSAT and other hill-climbing procedures, we noticed that GSAT experiences biggest loss in the performance in the beginning of the plateau phase where the amount of *poss-flips* can be as high as 20-25%. On the other hand, HSAT, NavFGSAT and NavRGSAT behave equally good during the hill-climbing phase and the beginning of the plateau phase. We thus concluded that any of the latter three procedures can be applied to provide fast approximate solutions. For some problems, versions of NavGSAT are not as efficient as HSAT. Nonetheless, we introduced NavFGSAT and NavRGSAT to show that HSAT's queuing policy is not the unique way of improving the efficiency of solving propositional satisfiability problems.

Approximate solutions can be utilized in time-critical problems where the quality of the solution discounts the time spent for solving the problem. NavGSAT can be also applied to problems with dynamically changing domains, when the domain changes can influence the decision making process. Finally, approximate solution provide an excellent starting point for a different search policy. For example, WGSAT and UGSAT [2] utilized a promising idea of the instant heuristic update based on the weight of unsatisfied clauses. An approximate solution provided by HSAT or NavGSAT constitutes an excellent starting point for WGSAT, UGSAT or another effective search procedure of a satisfiable solution, for example, ϵ-search (with heuristic updates). Among others we outline the following benefits of employing HSAT or NavGSAT to deliver a good starting point for another search method:

- Perfect initial assignment with a low number of unsatisfied clauses.
- Absence of hill-climbing phase that, for example, eliminates noise in tracking clause weights.
- Efficient search in both steps of policy-switching approach.
- Convenient point in time to fork search in multi-agent/multi-processor problem scenarios.

Although HSAT, itself, is an efficient hill-climbing procedure for randomly generated problems with a low clause-variable ratio, we expect that HSAT might experience difficulties in more constrained problems. NavGSAT provides another heuristic that guides efficiently in the initial phases. On the other hand, the hill-climbing phase may either produce noise in clause weight bookkeeping or redundant list of vertices with updated heuristics that slows down the performance of ϵ-search. Search with policy switching can benefit significantly from employing efficient procedures in all of its phases.

5 Navigational Problems

To confirm the results of our navigational approach to GSAT, we applied all the discussed above hill-climbing procedures to the following simple navigational problem:

> **Navigational Problem (NavP):** An agent is given a task to find the shortest path that reaches a goal vertex from a starting vertex in an "obstacle-free" rectangular grid.

NavP is a simplistic planning problem. It can be represented as a propositional satisfiability problem with $N = |S| * D$ variables, where S is the set of vertices in the rectangular grid and D is the shortest distance between starting vertex X and goal vertex G. In a correct solution, a variable x_s^d is assigned *True* ($x_s^d = 1$), if s is dth vertex on the shortest path from X to G, and *False* otherwise. There can be only one variable with the *True* value among variables representing grid vertices that are d-far from starting vertex X. This requirement implies $L_1 = \Theta(|S|^2 * D)$ pigeonhole-like constraints:

$$\bigwedge_{d=1}^{D-1} \bigwedge_{s_1 \neq s_2} (\neg V_{s_1}^d \vee \neg V_{s_2}^d)$$

Already these constraints make the domain look "over-constrained," since the ratio of L_1 to N is not asymptotically bounded. Another group of constraints has to force *True*-valued variables to form a continuous path. There can be different ways of presenting such constraints, we chose the easiest and the most natural presentation that does not produce extra variables:

$$\bigwedge_{d=2}^{D-1} \bigvee_{s \in S} (V_s^d \wedge (V_{s_1}^{d-1} \vee V_{s_2}^{d-1} \vee V_{s_3}^{d-1} \vee V_{s_4}^{d-1}))$$

Vertices $s_1, s_2, s_3, s_4 \in S$ are the neighbors of vertex $s \in S$ in the rectangular grid. To reduce the number of variables and clauses, the initial and the goal states are represented by stand-alone single clauses:

$$(V_{s_5}^1 \vee V_{s_6}^1 \vee V_{s_7}^1 \vee V_{s_8}^1) \quad \textit{and} \quad (V_{s_9}^{D-1} \vee V_{s_{10}}^{D-1} \vee V_{s_{11}}^{D-1} \vee V_{s_{12}}^{D-1}).$$

Vertices $s_5, s_6, s_7, s_8 \in S$ are the neighbors of the starting vertex, $s_9, s_{10}, s_{11}, s_{12} \in S$ are the neighbors of the goal vertex.

Second group of constraints is not presented in the classical CNF form. It is possible to reduce it to 3SAT, but such a reduction will introduce a lot of new variables and clauses and will significantly slow down the performance without facilitating search for a satisfiable assignment. From the point of view of hill-climbing procedures that track clause weights, this would mean only a different initial weight assignment and a linear change in bookkeeping clause weights. Therefore, we decided to stay with the original non-3SAT model and considered each complex conjunction $\bigvee_{s \in S} (V_s^d \wedge (V_{s_1}^{d-1} \vee V_{s_2}^{d-1} \vee V_{s_3}^{d-1} \vee V_{s_4}^{d-1}))$ as a single clause. Together with the starting and goal vertex constraints, the second group contains $L_2 = \Theta(D)$ constraints that force *True*-valued variables to form a continuous path.

It is fairly easy to come up with an initial solution, so that all but one constraint are satisfied. Figure 5 shows one of such solutions that alternates between the goal vertex and one of its neighbors, and the final path that satisfies all the constraints. The original GSAT has complexity that is exponential on D. It performs poorly for such domains, because at every step it has more equally good chances than any other algorithm. HSAT was able to solve "toy" problems with less than 200 variables until its search was under the influence of initial states. For larger problems, after an initial search HSAT used to switch to a

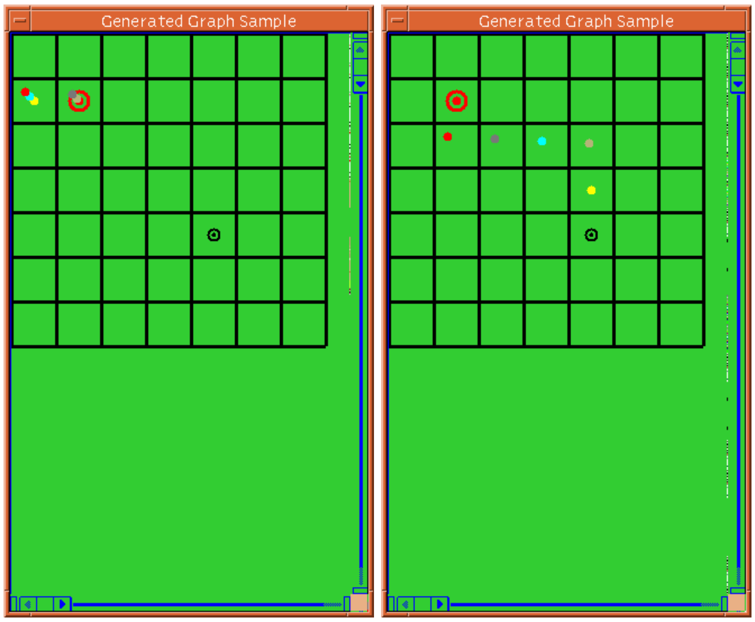

Fig. 5. Initial and final solutions for NavP.

systematic search that avoided changing recently changed vertices. Since HSAT re-started search from both starting and goal vertices on a regular basis, all the variable corresponding to their neighboring vertices has frequently changed their values. Therefore, paths from the opposite direction attempted to avoid changing these variables again. This was one of the domain where the queuing policy of HSAT played against it.

A slightly modified versions of NavFGSAT and NavRGSAT were capable of solving larger problems using Top-Down Depth-First-Search (TDDFS). TDDFS traverses repeatedly the search tree (the set of vertices reachable in D steps) from the root down, each time attempting to visit the least visited vertex from the current vertex or, if possible, unvisited vertex. The only modification of this behavior was that NavGSAT methods alternated roots between the starting vertex and the goal vertex while performing such search.

6 Conclusions

We showed that GenSAT hill-climbing procedures for solving propositional satisfiability problems can be interpreted as navigational, agent-centered search on a high-dimensional cube, NavGSAT. This type of search heavily depends on how well heuristic values represent the actual distance towards the set of goal states. HSAT, one of the most efficient GSAT-like procedures, maintains low level of *poss-flips*. We identified this property as the main benefit of HSAT in comparison with the original GSAT. However, the non-flexibility of HSAT's queuing policy can be an obstacle in solving more constrained problems. We introduced two versions of NavGSAT that also maintain low level of *poss-flips* and can be applied as approximating procedures for time-critical or dynamically changing problems, or serve as a starting phase in search procedures with switching search policies.

Acknowledgements

This research is sponsored in part by the National Science Foundation under grant number IRI-9502548. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations or the U.S. government.

References

1. Crawford, J., Auton, L.: Experimental results on the crossover point in satisfiability problems. In *Proceedings of 11th National Conference on Artificial Intelligence (AAAI)* (1993) 21–27
2. Frank, J.: Weighting for Godot: Learning Heuristics for GSAT. In *Proceedings of 13th National Conference on Artificial Intelligence (AAAI)* (1996) 338–343
3. Gent, I., Walsh, T.: Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of 11th National Conference on Artificial Intelligence (AAAI)* (1993) 28–33
4. Gent, I., Walsh, T.: An empirical analysis of search in GSAT. *Journal of Artificial Intelligence Research* **1** (1993) 47–59
5. Gent, I., Walsh, T.: Unsatisfied variables in local search. In J. Hallam, editor,*Hybrid Problems, Hybrid Solutions.* IOS press. (1995)
6. Gu, J.: Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin* **3(1)** (1992) 8–12
7. Ishida, T., Shimbo, M.: Improving the learning efficiencies of realtime search.. In *Proceedings of 13th National Conference on Artificial Intelligence (AAAI)* (1996) 338–343
8. Koenig, S., Smirnov, Y.: Graph learning with a nearest neighbor approach. In *Proceedings of the Conference on Computational Learning Theory (COLT)* (1996) 19–28
9. Korf, R.: Real-time heuristic search. *Artificial Intelligence* **42(2-3)** (1990) 189–211
10. Mitchell, D., Selman, B., Levesque, H.: Hard and easy distributions of SAT problems. In *Proceedings of 10th National Conference on Artificial Intelligence (AAAI)* (1992) 459–465
11. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In *Proceedings of 10th National Conference on Artificial Intelligence (AAAI)* (1992) 440–446
12. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In *Proceedings of 12th National Conference on Artificial Intelligence (AAAI)* (1994) 337–343
13. Smirnov, Y., Koenig, S., Veloso, M., Simmons, R.: Efficient goal-directed exploration. In *Proceedings of 13th National Conference on Artificial Intelligence (AAAI)* (1996) 292–297

This article was processed using the LaTeX macro package with LLNCS style