Diversification and Determinism in Local Search for Satisfiability*

Chu Min Li[1] and Wen Qi Huang[2]

[1]LaRIA, Université de Picardie Jules Verne,
33 Rue St. Leu, 80039, Amiens Cedex 1, France
chu-min.li@u-picardie.fr
[2]Huazhong university of science and technology, Wuhan, China
wqhuang@mail.hust.edu.cn

Abstract. The choice of the variable to flip in the Walksat family procedures is always random in that it is selected from a randomly chosen unsatisfied clause c. This choice in Novelty or R-Novelty heuristics also contains some determinism in that the variable to flip is always limited to the two best variables in c. In this paper, we first propose a diversification parameter for Novelty (or R-Novelty) heuristic to break the determinism in Novelty and show its performance compared with the random walk parameter in Novelty+. Then we exploit promising decreasing paths in a deterministic fashion in local search using a gradient-based approach. In other words, when promising decreasing paths exist, the variable to flip is no longer selected from a randomly chosen unsatisfied clause but in a deterministic fashion to surely decrease the number of unsatisfied clauses. Experimental results show that the proposed diversification and the determinism allow to significantly improve Novelty (and Walksat).

1 Introduction

Consider a propositional formula $\mathcal{F}$ in Conjunctive Normal Form (CNF) on a set of boolean variables $\{x_1, x_2, \ldots, x_n\}$, the satisfiability problem (SAT) consists in testing whether all clauses in $\mathcal{F}$ can be satisfied by some consistent assignment of truth values to variables.

SAT is the first known [1] and one of the most well-studied NP-complete problems. It has many applications like graph coloring, circuit designing or planning, since such problems can be encoded into CNF formulas in a natural way and solved by a SAT solver.

Given a CNF formula $\mathcal{F}$ and an assignment, local search procedures repeatedly repair locally this assignment, i.e. flipping the value of one variable, to find an assignment satisfying all clauses of $\mathcal{F}$. Since the introduction of GSAT [14] in which the best variable is picked to be flipped to decrease the number of unsatisfied clauses, there has been

* This work is partially supported by Programme de Recherches Avancées de Cooprations Franco-Chinoises (PRA SI02-04) and project Tolérant (Bestfit) under the research program HTSC of the Picardie region in France.

F. Bacchus and T. Walsh (Eds.): SAT 2005, LNCS 3569, pp. 158–172, 2005.

considerable research on local search methods to find satisfying assignments for CNF formulae, see, e.g. [3, 9, 12, 13, 7, 10, 11, 4].

Perhaps the most significant early improvement was to incorporate a "random walk" component where variables were flipped from some unsatisfied clause [12], leading to the development of the well-known Walksat procedure [13] in which the variable to flip is always picked from a randomly selected unsatisfied clause. Another contemporary idea was to break ties in favor of least recently flipped variables [3]. This improvement to GSAT resulted in HSAT. McAllester, Selman and Kautz introduced Novelty and R-Novelty heuristics into the Walksat family by combining two concerns when picking a variable to flip from within a unsatisfied clause: favoring the best variable to maximize the number of satisfied clauses and avoiding flipping the most recently flipped variable in the clause to prevent the local search from repeating earlier flips [10].

Novelty and R-Novelty are among the best local search methods. However, Hoos [5] showed that they are essentially incomplete in the sense that in some situations, they can get stuck in a local basin of attraction and fail to get out. Hoos developed slightly modified procedures Novelty+ and R-Novelty+ by forcing a random walk with a fixed probability in which the variable to flip is randomly picked from a random unsatisfied clause. Novelty+ and R-Novelty+ are probabilistically approximately complete (PAC), meaning that by running them long enough, the probability of missing an existing satisfying assignment can be made arbitrarily small [5].

We note that on one hand, the variable to flip is always randomly picked in the Walksat family procedures from a unsatisfied clause in the sense the unsatisfied clause is randomly selected, and on the other hand, the choice of the variable to flip in Novelty and R-Novelty also involves some determinism: the picked variable is necessarily one of the two best variables in the unsatisfied clause.

In this paper, we propose new local search procedures in the Walksat family in two ways:

(i) We weaken the determinism in Novelty in a way that all variables in the randomly selected unsatisfied clause c can be picked to be flipped instead of limited to the two best variables in c. Concretely, we introduce diversification moves in which THE least recently flipped variable in c is picked to be flipped, resulting a new heuristic called Novelty++. Novelty++ can be considered as a reinforcement of Novelty+ in the sense that while Novelty+ makes a random walk in which all variables in c can be picked with equal probability, Novelty++ deterministically picks THE least recently flipped variable in c in a diversification step.

(ii) We weaken the randomness of the Walksat family procedures by combining GSAT with Walksat. In some precisely defined situations, the variable to flip is picked in a deterministic way as in GSAT instead of from a randomly selected unsatisfied clause. We call the new procedure G^2WSAT for Gradient-based Greedy Walksat. In the remaining situations, G^2WSAT uses Novelty++ or other Walksat family heuristics to pick the variable to flip.

This paper is organized as follows. Section 2 presents Novelty++ and compares its performance with Novelty and Novelty+. Section 3 presents G^2WSAT and compares the performance of G^2WSAT using Novelty++ with SDF [11] and UnitWalk [4], the two other effective local search procedures, as well as Novelty and Walksat. Section 4 concludes.

2 Extending Novelty with Diversification

Originally introduced in [13], Walksat differs from its predecessor GSAT essentially in that the variable to be flipped is no longer (deterministically with probability 1-p and randomly with probability p) picked among all variables but from a randomly selected unsatisfied clause. It starts with a randomly generated truth assignment. Then it repeatedly changes (flips) the assignment of a variable picked according to a heuristic, until either a satisfying assignment is found or a given maximum number of flips, *Maxsteps*, is reached. This process is repeated up to a maximum number of *Maxtries* times.

Algorithm 1: Walksat

Input: SAT-formula $\mathcal{F}$, $Maxtries$, $Maxsteps$, $Heuristic$
Output: A satisfying truth assignment A of $\mathcal{F}$, if found
begin
  for *try=1* ***to*** $Maxtries$ **do**
    $A\leftarrow$randomly generated truth assignment;
    for *flip=1* ***to*** $Maxsteps$ **do**
      if A *satisfies* $\mathcal{F}$ **then** return A;
      $c\leftarrow$ randomly selected clause unsatisfied under A;
      $v\leftarrow$ pick a variable from c according to $Heuristic$;
      $A\leftarrow$ A with v flipped;
  return "Solution not found";
end;

Let x be a variable, break(x) be the number of clauses in $\mathcal{F}$ which are satisfied by the current assignment A but would be unsatisfied if x is flipped, make(x) be the number of clauses in $\mathcal{F}$ that currently are unsatisfied but would be satisfied if x is flipped. Let score(x) be the difference of make(x) and break(x) (score(x)=make(x) - break(x)). Walksat provides several heuristics to pick a variable to flip from a randomly chosen unsatisfied clause c, among which:

Walksat(p): If there are variables x in c such that break(x)=0, randomly pick one of them, otherwise with probability p, randomly pick a variable from c and with probability 1-p, randomly pick one of variables x such that break(x) is the smallest.

Novelty(p): Sort the variables x in c by score(x), breaking ties in favor of the least recently flipped variable. Consider the best and second best variable under this sort. If the best variable is not the most recently flipped one in c, then pick it. Otherwise, with probability p, pick the second best one, and with probability 1-p, pick the best variable.

R-Novelty(p): This is the same as Novelty, except in the case where the best variable is the most recently flipped one in c. In this case, pick the variable to flip among the best and second best variables according to p and the difference of score of these two variables. For more details, see [10].

We now concentrate on Novelty heuristic. Obviously, it always picks one of the two best variables according to their score that would result in the smallest (or the second

smallest) total number of unsatisfied clauses. The intuition of parameter p (called noise) is that if the best variable is the most recently flipped one in c, flipping it risks to cancel a useful earlier move. To avoid this, the second best variable is picked with probability p [10].

Hoos studied the run time behavior of Novelty and found that while Novelty is very effective, it may sometimes get stuck in a loop. As an example, Hoos gave a formula $\mathcal{F}$ consisting of the 6 following clauses:

$$c_1 : \bar{x_1} \vee x_2$$
$$c_2 : \bar{x_2} \vee x_1$$
$$c_3 : \bar{x_1} \vee \bar{x}_2 \vee \bar{y}$$
$$c_4 : x_1 \vee x_2$$
$$c_5 : \bar{z_1} \vee y$$
$$c_6 : \bar{z_2} \vee y$$

This formula has a unique solution x_1=x_2=1, y=z_1=z_2=0. Hoos showed that if the initial assignment is x_1=x_2=y=z_1=z_2=1, Novelty never reaches the unique solution [5], because y is never flipped.

The restart mechanism in Walksat allows to remedy the situation. However in practice this mechanism critically relies on the use of good $Maxsteps$ setting which is difficult to obtain a priori. Hoos then extended Novelty by forcing a random walk with probability wp (walk probability) in which a variable in c is randomly selected. The extended Novelty is called Novelty+.

Novelty+(p, wp): With probability wp, randomly pick a variable from c (random walk), with probability 1-wp, do as Novelty.

Let us look at Hoos example once again. The reason that Novelty gets stuck in a loop is due to the fact that y is never flipped. The purpose of the random walk in Novelty+ is to make the heuristic pick y when c_3 is selected. However the random walk does so only with probability 1/3. This observation leads us to extend Novelty in the following way:

Novelty++(p, dp): With probability dp (diversification probability), pick the least recently flipped variable in c (diversification), with probability 1-dp, do as Novelty.

Obviously, the difference between Novelty+ and Novelty++ is that the random walk in Novelty+ is replaced by the diversification in Novelty++. In practice, Novelty++ is stronger than Novelty+. For instance, Novelty++ directly picks y in c_3 in the above example in a diversification step while Novelty+ may still pick x_1 or x_2 in a random walk as Novelty does.

When Novelty gets stuck in a local basin of attraction, probably there is a clause c that is unsatisfied again and again. The diversification in Novelty++ allows to flip all variables in c by turns when the search proceeds, since after the least recently variable in c is flipped, a different variable in c becomes the new least recently flipped.

So Novelty++ presumably improves (further than Novelty+) the coverage rate of Novelty as defined in [11] to measure how systematically the search explores the entire space.

Table 1 compares the performance of Novelty(p), Novelty+(p, wp) and Novelty++(p, dp) (respectively N(p), N+(p, wp) and N++(p, dp) in the table) for random 3-SAT problems and several classes of structured problems, where p is the noise parameter which is fixed to be 0.20, 0.35 and 0.50. wp and dp respectively are the random walk probability in Novelty+ and the diversification probability in Novelty++ which are fixed to be 0.01, 0.02 and 0.05.

A total of 21 local search procedures are evaluated in table 1. All these procedures share the same implementation and the same data structure from Satz [8], and simplify the input formula by satisfying the eventual unit clauses in the formula before the local search. Note that Novelty(p)≡Novelty+(p, 0)≡Novelty++(p, 0).

We generate 2000 random 500 variable and 2125 clause 3-SAT formulas and eliminate the 912 unsatisfiable ones using Satz. For larger hard random 3-SAT problems, we generate 1000 random 600 variable and 2550 clause formulas and 300 random 1000 variable and 4250 clause formulas. However no solver is available to eliminate unsatisfiable formulas of these sizes in reasonable time. So the 1000 hard random 600 variable formulas, as well as the 300 hard random 1000 variable formulas, probably contain about a half of unsatisfiable problems also used in the experimentation. $Maxsteps$ is fixed to be 10^5 for 500 variable problems, 2×10^5 for 600 variable problems and 5×10^5 for 1000 variable problems. 10^5 is the default cutoff value of the original Walksat family procedures, while 2×10^5 and 5×10^5, as well as 10^6 and 10^7 used below, are cutoff values somewhat arbitrarily fixed here.

The behavior of the local procedures for other cutoff values deserves future study.

We run Novelty, Novelty+, Novelty++ with $Maxtries = 1$ (one run) for each formula in each class. We say a formula is *solved* by a procedure if the procedure finds a solution satisfying all clauses of the formula. The number of solved formulas in a class represents the success rate of the procedure for this class. This execution is repeated 100 times to get the final averaged success rate given in the table 1.

Structured problems Flat200-479, QG, AIS, Logistics, Blockworld, all available in SATLIB[1], are also used to evaluate the performance of Novelty++ compared with Novelty and Novelty+. In order to make the comparison clearer, we eliminate unsatisfiable formulas in the QG class, and smaller formulas in the AIS, Logistics and Blockworld classes in which larger a formula, harder it is. So in our experimentation, local search procedures are run to solve the 100 satisfiable formulas in Flat200-479 class, the 10 satisfiable formulas in QG (satQG in the table) and the largest (and the hardest) formula remaining in the AIS, Logistics and Blockworld classes. $Maxsteps$ is fixed to be 10^6 to all these problems except bw_large.d for which 10^7 flips are used. As for random problems, we run each procedure with $Maxtries = 1$ for each formula in a class to get a success rate for this class and repeat the execution 100 times to get final averaged success rate given in table 1.

[1] http://www.satlib.org

Table 1. Average success rate and successful run length (the number of flips to find a solution in a successful run) of Novelty, Novelty+ and Novelty++ for random 3-SAT and structured problems

	500vars	600vars	1000vars	Flat200	satQG	ais12	logistics.d	bw_large.d
	succ rate	succ rate	succ rate	succ rate	succ rate	succ rate	succ rate	succ rate
	#flip	#flip	#flip	#flip	#flip	#flip	#flip	#flip
N(.2)	0.0358	0.0137	0.0020	0.0772	0.610	0	0.94	0.81
	43919	89461	263457	314589	157354	0	314949	3819298
N+(.2, .01)	0.0620	0.0257	0.0073	0.1329	0.626	0.09	0.94	0.74
	43407	87354	244555	342308	177019	600588	272877	4092116
N++(.2, .01)	*0.0752*	*0.0331*	*0.0096*	*0.1665*	*0.621*	*0.32*	*0.97*	*0.70*
	41308	*83556*	*233827*	*326908*	*174334*	*517261*	*255227*	*3769416*
N+(.2, .02)	0.0717	0.0307	0.0091	0.1554	0.610	0.15	0.98	0.68
	42755	84166	230063	331649	166960	394956	252352	4803834
N++(.2, .02)	*0.0922*	*0.0409*	*0.0121*	*0.2023*	*0.642*	*0.47*	*0.99*	0.54
	39599	*81692*	*216712*	*314755*	*168948*	*472868*	*257194*	*4281634*
N+(.2, .05)	0.0944	0.0407	0.0138	0.1911	0.628	0.36	0.97	0.47
	39612	80012	226484	307785	176049	530276	245133	4755044
N++(.2, .05)	*0.1299*	*0.0592*	*0.0225*	*0.2937*	*0.662*	*0.78*	*0.98*	*0.13*
	37599	*75300*	*218320*	*314195*	*155574*	*370399*	*187548*	*4551418*
N(.35)	0.1889	0.0942	0.0405	0.3778	0.684	0	1	0.01
	39923	81522	233992	336305	101838	0	150960	523016
N+(.35, .01)	0.2236	0.1137	0.0537	0.4887	0.663	0.08	1	0.01
	38523	78222	225226	329866	82270	374812	139387	3935403
N++(.35, .01)	*0.2525*	*0.1315*	*0.0676*	*0.5601*	*0.655*	*0.16*	*1*	*0.01*
	36874	*74005*	*210373*	*299901*	*78833*	*513290*	*146827*	*9034913*
N+(.35, .02)	0.2411	0.1250	0.0626	0.5224	0.660	0.08	1	0.01
	37778	74336	214503	302060	84524	284610	154852	7636423
N++(.35, .02)	*0.2850*	*0.1503*	*0.0805*	*0.6283*	*0.655*	*0.28*	*1*	*0.01*
	35694	*71890*	*197205*	*287165*	*84537*	*472346*	*171744*	*4258715*
N+(.35, .05)	0.2807	0.1498	0.0808	0.5997	0.650	0.24	1	0.01
	36191	72502	197306	290165	80302	494390	169413	7666983
N++(.35, .05)	*0.3727*	*0.2036*	*0.1234*	*0.7947*	*0.634*	*0.49*	*1*	*0*
	34111	*68382*	*184949*	*252688*	*55898*	*512219*	*212222*	*0*
N(.5)	0.5185	0.3118	0.2375	0.8585	0.613	0	0.71	0
	34169	67335	188880	225958	67213	0	434699	0
N+(.5, .01)	0.5360	0.3262	0.2561	0.8877	0.612	0.09	0.57	0
	33618	65780	179796	208462	69846	597206	487620	0
N++(.5, .01)	*0.5617*	*0.3449*	*0.2798*	*0.9154*	*0.604*	*0.1*	*0.50*	*0*
	32575	*63487*	*171378*	*189963*	*67538*	*505106*	*438835*	*0*
N+(.5, .02)	0.5482	0.3354	0.2673	0.9011	0.602	0.07	0.43	0
	33223	65321	177141	199674	59886	481570	453581	0
N++(.5, .02)	*0.5870*	*0.3634*	*0.3048*	*0.9223*	*0.603*	*0.21*	*0.27*	*0*
	31879	*61308*	*162701*	*174457*	*69033*	*599057*	*382697*	*0*
N+(.5, .05)	0.5708	0.3520	0.2949	0.9142	0.601	0.11	0.37	0
	32511	62915	167434	188466	73372	459436	529073	0
N++(.5, .05)	*0.5919*	*0.3641*	*0.3163*	*0.9397*	*0.593*	*0.33*	*0.14*	*0*
	30953	*59633*	*153879*	*175887*	*73184*	*473227*	*437366*	*0*

Table 1 shows that Novelty++ is consistently better than Novelty and Novelty+ in case noise is important (random 3-SAT, Flat200) and in case stagnation behavior occurs (ais12). In other words, when random walk is needed, diversification systematically does better. Novelty++ generally has a success rate 2 or 3 times larger than Novelty+ for ais12 for the same value of wp and dp. It also solves significantly more random 3-SAT and Flat200 formulas, especially when noise parameter is low (0.2 and 0.35).

Note that the success rate in table 1 can be computed in an equivalent way as follows. We consider a multi-set based on each class where every formula occurs 100 times. We run a local search procedure with $Maxtries = 1$ for every formula in the multi-set. The number of formulas solved divided by the number of elements in the multi-set is the success rate. For example, the Flat200 class has 100 formulas. The corresponding multi-set has 100*100=10000 formulas. Novelty++(0.35, 0.05) solves 7947 formulas in the multi-set. Its success rate for the class Flat200 is 0.7947. It solves 1950 (over 10000) formulas more than Novelty+(0.35, 0.05) in the multi-set.

We recall that about a half of formulas in the random 600 and 1000 variable 3-SAT classes probably might be unsatisfiable. The success rate for these two classes probably might be multiplied by 2. Consequently, the success rate difference between Novelty++ and Novelty+ for these two classes might also be multiplied by 2.

It appears in Table 1 that QG problems are not sensitive to noise nor to random walk, since the success rates for different noise parameters don't have significant difference. The only cases Novelty+ is better than Novelty++ are the logistics.d and bw_large.d problems for which Novelty(0.2) is the best. In other words, Neither random walk nor diversification is necessary for these two problems. Moreover, noise should be low to solve them.

All evaluated procedures roughly have the same time complexity per flip. Table 1 shows that when Novelty++ has better success rate, it also generally needs fewer flips to find a solution.

3 Exploiting Promising Decreasing Paths in Local Search

The behavior of GSAT before finding a solution might roughly be characterized as follows:

1. Repeatedly decrease the number of unsatisfied clauses (move down along a decreasing path) while possible;
2. Escape from a local minimum;
3. Go to 1.

One major difference between Walksat family and GSAT is that the behavior of Walksat can no longer be characterized as above, since the variable to flip is always picked from a randomly selected unsatisfied clause and may increase the number of unsatisfied clauses in non local minima. The difficulty in GSAT is how to recognize a promising decreasing path from previous moves, since moving along a decreasing path risks to simply repeat (or cancel) earlier flips, which might explain the performance of Walksat over GSAT.

However we believe that the purpose of a local search procedure always is to repeatedly decrease the number of unsatisfied clauses. Even when the procedure flips a variable such that the number of unsatisfied clauses is increased, it hopes that decreasing variables appear and can lead to a solution satisfying all clauses.

This observation suggests us that the decreasing variables resulted from a move could be better and more promising than variables in a randomly picked unsatisfied clause c, and that in this case, a local search procedure should pick one of these decreasing variables instead of a variable in c.

We now formally define promising decreasing variables and promising decreasing paths.

A variable is said decreasing if flipping it would decrease the number of unsatisfied clauses. Let x and y be variables, $x \neq y$, y is not decreasing. If it becomes decreasing after x is flipped, then we say that y is a *promising decreasing variable* after x is flipped. There may be 0, 1, or several promising decreasing variables after x is flipped. All promising decreasing variables are collected in a set.

Let y be a promising decreasing variable after some variable is flipped. If y is always decreasing after one or more other moves, it is always promising and remains in the promising decreasing variable set. Otherwise it should be removed from the set.

A *promising decreasing path* is a sequence of moves in which every move flips a promising decreasing variable.

Note that if a variable x is flipped such that the number of clauses is increased, re-flipping x would decrease the number of unsatisfied clauses, i.e., x is decreasing. However x is not a promising decreasing variable, since re-flipping x would simply cancel a previous move. The major originality of our approach is that such x is never considered when exploiting promising decreasing paths.

We want a local search procedure which, whenever there are promising decreasing variables, does as GSAT and deterministically picks the best of them to minimize the total number of unsatisfied clauses, breaking ties in favor of the least recently flipped variable as in HSAT [3]. In other cases, the procedure does as Walksat and uses a heuristic such as Novelty++ to pick the variable to flip.

For this purpose, we need a method to efficiently find all promising decreasing variables after a flip and remove old promising decreasing variables which are no longer decreasing.

A variable x is decreasing iff score(x)=make(x)-break(x) > 0. The following gradient-based approach originally introduced in [7] allows us to compute the score of every variable after a flip.

Assume that the formula $\mathcal{F}$ contains m clauses on n variables. Let c_i $(1 \leq i \leq m)$ be a clause in $\mathcal{F}$. c_i=$x_{i_1} \vee ... \vee x_{i_k} \vee \bar{x}_{i_{k+1}} \vee ... \vee \bar{x}_{i_{k+r}}$. Note that we put all positive literals in c_i before the negative ones.

We consider now all variables in c_i as integer variables taking values 0 or 1. We define:

$$\mathcal{E}_i(x_{i_1}, ..., x_{i_k}, x_{i_{k+1}}, ..., x_{i_{k+r}}) = (1 - x_{i_1})...(1 - x_{i_k})x_{i_{k+1}}...x_{i_{k+r}}$$

Obviously $\mathcal{E}_i$ has value 0 iff one of x_{i_j} $(1 \leq j \leq k)$ is assigned 1 or one of x_{i_s} $(k + 1 \leq s \leq r)$ is assigned 0. In other words, $\mathcal{E}_i$=0 iff c_i is satisfied. otherwise $\mathcal{E}_i$=1.

We now define:

$$\mathcal{E}(x_1, ..., x_n) = \sum_{i=1}^{m} \mathcal{E}_i \tag{1}$$

Given an assignment A (a point in the search space), the value of $\mathcal{E}$ is the number of unsatisfied clauses in $\mathcal{F}$. If A satisfies all clauses in $\mathcal{F}$, then $\mathcal{E} = 0$.

Since each variable appears at most once in a clause, $\mathcal{E}$ is a linear function for any variable x_f and can be written as $\mathcal{E}(x_f)$. Let v_f be the current value of x_f. $\mathcal{E}(v_f)$ stands for $\mathcal{E}$ simplified after the substitution of x_f by v_f. Taylor's equation gives us

$$\mathcal{E}(x_f) = \mathcal{E}(v_f) + (x_f - v_f)\frac{\partial \mathcal{E}(x_f)}{\partial x_f} \tag{2}$$

So $\frac{\partial \mathcal{E}(x_f)}{\partial x_f} \neq 0$ indicates the variation of $\mathcal{E}$ when x_f changes. If $\frac{\partial \mathcal{E}(x_f)}{\partial x_f} > 0$, then $\mathcal{E}(x_f)$ increases (decreases) when x_f increases (decreases). If $\frac{\partial \mathcal{E}(x_f)}{\partial x_f} < 0$, then $\mathcal{E}(x_f)$ decreases (increases) when x_f increases (decreases). So we can get all decreasing variables at a given point A by computing $\frac{\partial \mathcal{E}(A)}{\partial x_f}$ for every variable x_f.

For example, if at point A, x_1=0, x_2=1, x_3=0, x_4=1, $\frac{\partial \mathcal{E}(A)}{\partial x_1}$=2, $\frac{\partial \mathcal{E}(A)}{\partial x_2}$=2, $\frac{\partial \mathcal{E}(A)}{\partial x_3}$=-2, $\frac{\partial \mathcal{E}(A)}{\partial x_4}$=-2, then x_1 and x_4 are increasing variables ($\mathcal{E}$ will be increased by 2 if x_1 or x_4 is flipped), and x_2 and x_3 are decreasing variables ($\mathcal{E}$ will be decreased by 2 if x_2 or x_3 is flipped). Note that score(x_1)=-$\frac{\partial \mathcal{E}(A)}{\partial x_1}$=-2, score($x_2$)=$\frac{\partial \mathcal{E}(A)}{\partial x_2}$=2, score($x_3$)=-$\frac{\partial \mathcal{E}(A)}{\partial x_3}$=2, score($x_4$)=$\frac{\partial \mathcal{E}(A)}{\partial x_4}$=-2. In other words, |score(x_f)|=|$\frac{\partial \mathcal{E}(A)}{\partial x_f}$| for all x_f, but the sign may be different.

We now rewrite Taylor's equation 2 as

$$\mathcal{E}(x_1, ..., x_f, ..., x_n) = \mathcal{E}(x_1, ..., v_f, ..., x_n) + (x_f - v_f)\frac{\partial \mathcal{E}(x_1, ..., x_f, ..., x_n)}{\partial x_f} \tag{3}$$

Then for any variable $x_g \neq x_f$, we have:

$$\frac{\partial \mathcal{E}(x_1, ..., x_f, ..., x_n)}{\partial x_g} = \frac{\partial \mathcal{E}(x_1, ..., v_f, ..., x_n)}{\partial x_g} + (x_f - v_f)\sum_{i=1}^{m} \frac{\partial^2 \mathcal{E}_i}{\partial x_f \partial x_g} \tag{4}$$

We denote the assignment after t flips by A^t in local search. The value of x_j is v_j^t ($1 \leq j \leq n$). After a new flip, A^t becomes A^{t+1} in which $v_j^t = v_j^{t+1}$ except for one variable x_f=$v_f^{t+1} = 1 - v_f^t$. Assuming that we know the value of $\frac{\partial \mathcal{E}(x_j)}{\partial x_j}$ for every x_j at point A^t, equation 4 suggests us a reasonable and efficient way to compute $\frac{\partial \mathcal{E}(x_j)}{\partial x_j}$ at point A^{t+1}.

In fact, we note that $v_f = v_f^t$ at point A^t. At point A^{t+1}, $\frac{\partial \mathcal{E}(x_1,...,x_f,...,x_n)}{\partial x_g}$ is $\frac{\partial \mathcal{E}(A^{t+1})}{\partial x_g}$, but $\frac{\partial \mathcal{E}(x_1,...,v_f^t,...,x_n)}{\partial x_g}$ is equal to $\frac{\partial \mathcal{E}(A^t)}{\partial x_g}$, since A^{t+1} differs from A^t only

in the value of x_f (recall that $\frac{\partial \mathcal{E}(x_1,\ldots,v_f^t,\ldots,x_n)}{\partial x_g}$ is $\frac{\partial \mathcal{E}(x_1,\ldots,x_f,\ldots,x_n)}{\partial x_g}$ with x_f replaced by v_f^t so that all variables in it have value by A^t).

Since $x_f - v_f^t = v_f^{t+1} - v_f^t = 1 - v_f^t - v_f^t$ at point A^{t+1}, equation 4 becomes:

$$\frac{\partial \mathcal{E}(A^{t+1})}{\partial x_g} = \frac{\partial \mathcal{E}(A^t)}{\partial x_g} + (1 - 2v_f^t) \sum_{i=1}^{m} \frac{\partial^2 \mathcal{E}_i}{\partial x_f \partial x_g} \tag{5}$$

Note that $\sum_{i=1}^{m} \frac{\partial^2 \mathcal{E}_i}{\partial x_f \partial x_g} = 0$ for all x_g not occurring in any clause containing x_f. To see this, let $c_i = x_1 \vee \bar{x}_2 \vee x_4$, $\mathcal{E}_i = (1-x_1)x_2(1-x_4)$. $\frac{\partial^2 \mathcal{E}_i}{\partial x_2 \partial x_g} = 0$ for any $g \notin \{1, 4\}$ (including $g = 2$). The following properties hold:

$$\frac{\partial \mathcal{E}(A^{t+1})}{\partial x_g} = \begin{cases} \frac{\partial \mathcal{E}(A^t)}{\partial x_g}, \text{ if } x_g \text{ not occurring in any clause with } x_f \\ \frac{\partial \mathcal{E}(A^t)}{\partial x_g} + (1 - 2v_f^t) \sum\limits_{x_f \text{ and } x_g \text{ occur in clause } c_i} \frac{\partial^2 \mathcal{E}_i}{\partial x_f \partial x_g} \\ \frac{\partial \mathcal{E}(A^t)}{\partial x_f}, \text{ if } x_g = x_f \end{cases} \tag{6}$$

Equation 6 shows that when x_f is flipped, $\frac{\partial \mathcal{E}(x_1,\ldots,x_f,\ldots,x_n)}{\partial x_g}$ should be re-computed only for those x_g occurring in some clause containing x_f. These variables and corresponding clauses can be stored by a preprocessing in a list associated with x_f to speed up the calculus.

In summary, the local search procedure, which we call G^2WSAT for Gradient-based Greedy Walksat, uses equation 6 to maintain a set of promising decreasing variables and flips the best promising decreasing variable if any. Otherwise, it uses a heuristic such as Novelty++ to pick a variable to flip.

G^2WSAT is defined in algorithm 2.

Table 2 compares the success rate of G^2WSAT using Novelty++ ($G^2(p, dp)$ in the table) with Novelty++ (N++(p, dp) in the table) on the same problems as in table 1. Novelty++(p, 0) is just Novelty(p). Recall the only difference between G^2WSAT and Novelty++ here is that G^2WSAT flips the best promising decreasing variable if any. Otherwise it is the same as Novelty++. The success rate is also computed in the same manner. The $Maxsteps$ (cutoff value for the number of flips) is also the same as in table 1 (10^5 for random 500 variable 3-SAT, 2×10^5 for 600 variables, 5×10^5 for 1000 variables; 10^7 for bw_large.d and 10^6 for other structured problems).

Due to the lack of space, we don't give the successful run lengths (#flips) of G^2WSAT and Novelty++ in table 2. G^2WSAT generally needs fewer flips than Novelty++ to find a solution. Table 3 gives some typical examples of the successful lengths of G^2WSAT which can be compared with those of Novelty++ given in table 1.

Table 2 shows that G^2WSAT is almost always better than Novelty++ except for Flat200 problems. In particular, the best success rate is always obtained by G^2WSAT, even for Flat200 problems. It is remarkable that $G^2WSAT(0.2, dp)$ improves Novelty++(0.2, dp) and raises its success rate to 100%.

Algorithm 2: G^2WSat

Input: SAT-formula $\mathcal{F}$, $Maxtries$,$Maxsteps$, $Heuristic$
Output: A satisfying truth assignment A of $\mathcal{F}$, if found
begin
  for *try=1 to* $Maxtries$ **do**
    $A \leftarrow$ randomly generated truth assignment;
    Compute $\frac{\partial \mathcal{E}(x_1,\ldots,x_n)}{\partial x_j}$ for all x_j at A;
    Store all decreasing variables in stack $DecVar$;
    for *flip=1 to* $Maxsteps$ **do**
      if A *satisfies* $\mathcal{F}$ **then** return A;
      if $DecVar$ *is not empty* **then** $x \leftarrow x$ in $DecVar$ such that $|\frac{\partial \mathcal{E}(x_1,\ldots,x_n)}{\partial x}|$ is the largest, breaking ties in favor of the least recently flipped variable;
      else $c \leftarrow$ randomly selected clause unsatisfied under A;
        $x \leftarrow$ pick a variable from c according to $Heuristic$;
      $A \leftarrow A$ with x flipped;
      update $\frac{\partial \mathcal{E}(x_1,\ldots,x_n)}{\partial x_j}$ for all x_j using equation 6;
      delete all variables which are no longer decreasing from $DecVar$;
      push all new decreasing variables into $DecVar$ which are different from x and were not decreasing before x is flipped;
  return "Solution not found";
end;

As in table 1, the success rate difference between G^2WSAT and Novelty++ for random 600 and 1000 variable 3-SAT problems might probably be multiplied by 2. It appears in table 2 that the success rate difference for 3-SAT increases with the noise and diversification parameters. For example, for 500 variable 3-SAT problems, the largest success rate difference is 0.0337 when the noise parameter is 0.2, it is respectively 0.0404 and 0.0625 when the noise parameter is 0.35 and 0.5. On the other hand, when the noise parameter is 0.5, the success rate difference between G^2WSAT and Novelty++ respectively is 0.0150, 0.0177, 0.0342 and 0.0625 when the diversification parameter is 0, 0.01, 0.02 and 0.05. The same phenomenon can be observed for the 600 and 1000 variable problems.

The difference of 0.0625 here for random 500 variable 3-SAT means that G^2WSAT(0.5, 0.05) solves 6807 more formulas than Novelty++(0.5, 0.05) in our experimentation. Refer to table 1, G^2WSAT(0.5, 0.05) solves 9098 more random 500 variable 3-SAT formulas than Novelty+(0.5, 0.05), representing a success rate difference of 0.0836. Considering the hardness to improve the Walksat family procedures which are already highly effective, we believe that G^2WSAT combining with Novelty++ is a significant improvement. As a comparison, the success rate difference between Novelty(0.5) and Walksat(0.5) for these formulas is 0.0704, see table 3 below.

In table 3, we compare G^2WSAT combined with Novelty++(p, dp) (G^2(p, dp) in the table) with Walksat, Novelty, UnitWalk, SDF on the same problems as in table 1 or in table 2. In addition to the success rates computed in the same manner as in table 1, we also report the average number of flips to find a solution (i.e. average length of a successful run) and the total real run time in seconds on an Athlon2000+

Table 2. Average success rate of Novelty++ and G^2WSAT using Novelty++ for random 3-SAT and structured problems

	500vars	600vars	1000vars	Flat200	satQG	ais12	logistics.d	bw_large.d
N++(.2, 0)	0.0358	0.0137	0.0020	0.0772	0.610	0	0.94	0.81
G^2(.2, 0)	*0.0477*	*0.0188*	*0.0034*	*0.0693*	*0.672*	*0*	*1*	*0.93*
N++(.2, .01)	0.0752	0.0331	0.0096	0.1665	0.621	0.32	0.97	0.70
G^2(.2, .01)	*0.0992*	*0.0416*	*0.0123*	*0.1607*	*0.679*	*0.52*	*1*	*0.89*
N++(.2, .02)	0.0922	0.0409	0.0121	0.2023	0.642	0.47	0.99	0.54
G^2(.2, .02)	*0.1178*	*0.0513*	*0.0160*	*0.1955*	*0.685*	*0.68*	*1*	*0.77*
N++(.2, .05)	0.1299	0.0592	0.0225	0.2937	0.662	0.78	0.98	0.13
G^2(.2, .05)	*0.1636*	*0.0754*	*0.0285*	*0.2701*	*0.725*	*0.88*	*1*	0.26
N++(.35, 0)	0.1889	0.0942	0.0405	0.3778	0.684	0	1	0.01
G^2(.35, 0)	*0.2104*	*0.1056*	*0.0438*	*0.3207*	*0.732*	*0*	*1*	*0.17*
N++(.35, .01)	0.2525	0.1315	0.0676	0.5601	0.655	0.16	1	0.01
G^2(.35, .01)	*0.2884*	*0.1490*	*0.0754*	*0.5083*	*0.730*	*0.25*	*1*	*0.03*
N++(.35, .02)	0.2850	0.1503	0.0805	0.6283	0.655	0.28	1	0.01
G^2(.35, .02)	*0.3229*	*0.1710*	*0.0904*	*0.5778*	*0.747*	*0.40*	*1*	*0.01*
N++(.35, .05)	0.3727	0.2036	0.1234	0.7947	0.634	0.49	1	0
G^2(.35, .05)	*0.4131*	*0.2278*	*0.1318*	*0.7226*	*0.737*	*0.59*	*1*	*0*
N++(.5, 0)	0.5185	0.3118	0.2375	0.8585	0.613	0	0.71	0
G^2(.5, 0)	*0.5335*	*0.3218*	*0.2372*	*0.8312*	*0.734*	*0*	*0.84*	*0*
N++(.5, .01)	0.5617	0.3449	0.2798	0.9154	0.604	0.1	0.50	0
G^2(.5, .01)	*0.5894*	*0.3568*	*0.2832*	*0.9051*	*0.710*	*0.13*	*0.65*	*0*
N++(.5, .02)	0.5870	0.3634	0.3048	0.9223	0.603	0.21	0.27	0
G^2(.5, .02)	*0.6212*	*0.3788*	*0.3127*	*0.9283*	*0.713*	*0.25*	*0.53*	*0*
N++(.5, .05)	0.5919	0.3641	0.3163	0.9397	0.593	0.33	0.27	0
G^2(.5, .05)	*0.6544*	*0.4018*	*0.3521*	*0.9511*	*0.699*	*0.38*	*0.34*	*0*

under Linux of each procedure to run 100 times for each class. Note that G^2WSAT uses equation 6 to incrementally update the score of each variable, which is quite different from the original Walksat procedures. To show that the gradient-based approach improves the time performance, we use original Walksat_v37 downloaded from http://www.cs.washington.edu/homes/kautz in table 3.

UnitWalk is downloaded from http://logic.pdmi.ras.ru/~arist/UnitWalk. SDF is downloaded http://www.cs.ualberta.ca/~dale/software.html.

We run UnitWalk using the following command line

UnitWalk -f $\mathcal{F}$ -p $Maxsteps$ -r 100 -sr -T 20000

where a time cutoff of 2×10^4 seconds is set to solve formula $\mathcal{F}$, and run SDF using the following command line

sdfflt -mf $Maxsteps$ -mr 1 -rep 100 -ne -f $\mathcal{F}$

to solve formula $\mathcal{F}$.

The $Maxsteps$ value is the same as in table 1 or in table 2 for different problems.

The best success rate, #flips, total run time for each class are bold-faced in table 3. Generally, a procedure with a better success rate is faster and has shorter successful runs. However, SDF uses a float-point implementation in which search steps are more expensive. So-called substitution operations are needed in UnitWalk in addition

Table 3. Experimental results for UnitWalk, SDF, G^2WSAT, Novelty and Walksat

	500vars #flips time	600vars #flips time	1000vars #flips time	Flat200 #flips time	satQG #flips time	ais12 #flips time	logistics.d #flips time	bw_large.d #flips time
SDF	0.3674 36218 56165s	0.1707 70702 115162s	0.0437 215026 210688s	**0.9859** **134555** 7057s	0.42 93207 ?	**1** **145781** 267s	1 65080 2442s	? ? ?
UnitWalk	0.4409 41429 25132s	0.2731 76443 53733s	0.2399 205646 55543s	0.8960 325342 24646s	0.5280 134808 93620s	1 232861 10374s	1 **3791** 53s	0.02 5362045 19899s
$G^2(.2,0)$	0.0477 40950 12263s	0.0188 86223 23166s	0.0034 236392 20951s	0.0693 312826 5532s	0.672 167887 **5452s**	0 0 221s	1 133848 28s	**0.93** **3419339** **2733s**
$G^2(.2,.05)$	0.1636 35207 11519s	0.0754 70840 22970s	0.0285 198330 21147s	0.2692 293976 4813s	0.725 162965 5642s	0.88 278722 **80s**	**1** 98065 **26s**	0.26 4543846 8867s
$G^2(.5,0)$	0.5335 33214 8771s	0.3218 66156 19963s	0.2372 183047 19151s	0.8312 240477 2297s	**0.734** 87954 6392s	0 0 227s	0.84 438108 172s	0 0 14329s
$G^2(.5,.05)$	**0.6544** **29238** **7666s**	**0.4018** **55478** **18835s**	**0.3521** **145030** **17701s**	0.9511 154994 **1264s**	0.699 **84898** 7184s	0.38 521013 186s	0.34 565058 318s	0 0 15047s
Novelty(.2)	0.036 43200 14631s	0.0140 89277 29099s	0.0023 278784 26590s	0.0734 315588 7763s	0.098 262181 27657s	0 0 409s	0.95 296197 57s	0.79 3672728 3819s
Novelty(.5)	0.5178 34225 10786	0.3123 66702 25613s	0.2371 186607 24614s	0.8544 226115 2940s	0.189 399360 36628s	0 0 427s	0.64 467981 210s	0 0 17557s
Walksat(.2)	0.1129 41111 13582s	0.0525 82424 26269s	0.0187 234638 21509s	0.1093 337845 7115s	0.064 176761 27397s	0.26 492378 343s	0.64 617464 104s	0.51 4574077 3976s
Walksat(.5)	0.4474 40169 12074s	0.2844 77768 25334s	0.2341 202990 20614s	0.8118 296044 4155s	0.04 447541 38714s	0.05 554170 391s	0.60 476361 168s	0 0 15368s

to flips. Some irregularities may also happen in some problem classes. For example, $G^2WSAT(0.2,0)$ solves more easily qg7-13 problems in QG class for which unsuccessful runs are very time-consuming, which explains the better time performance of $G^2WSAT(0.2,0)$ with a lower success rate.

Table 3 shows that G^2WSAT generally has better performance than Novelty using the same parameters except for Flat200 problems, for which $G^2WSAT(p, dp)$ is not better than Novelty(p) when dp=0, but $G^2WSAT(p, dp)$ is significantly better than Novelty(p) when $dp > 0$.

When noise p is 0.2, G^2WSAT(0.2, 0) generally is not better than Walksat(0.2). Again the diversification allows to remedy the situation and makes G^2WSAT(0.2, 0.05) substantially better than Walksat(0.2). When noise p is 0.5, G^2WSAT(0.5, 0) generally is substantially better than Walksat except for ais12 for which diversification

is necessary and makes $G^2WSAT(0.5, 0.05)$ significantly better than Walksat(0.5), and except for logistics.d and bw_large.d for which noise should be low.

We observe that G^2WSAT always gives the best result among all Walksat family procedures in table 3, using appropriate noise and diversification parameters. Furthermore, it seems that G^2WSAT spends less time per flip than the original Novelty and Walksat used here, thanks to the gradient-based approach. For example, $G^2WSAT(0.5, 0)$ spends 227 seconds to make the 100×10^6 flips to solve ais12 100 times without success, while Novelty(0.2) and Novelty(0.5) respectively spend 409 and 427 seconds to do the same thing[2].

$G^2WSAT(0.5, 0.05)$ has a success rate significantly better than UnitWalk for random 3-SAT, Flat200 and QG problems, while $G^2WSAT(0.2, 0.05)$ is better for QG and bw_large.d problems. UnitWalk performs well for ais12 formula with a success rate 1 while the best rate of G^2WSAT for this formula is 0.88. Nevertheless, G^2WSAT is substantially faster than UnitWalk for all problems in table 3. $G^2WSAT(0.2, 0.05)$ also has the success rate 1 for ais12 formula using 3×10^6 flips in a run and is still much faster than UnitWalk using 10^6 flips in a run.

The same observation can be made when comparing G^2WSAT and SDF. We fail to run SDF for some QG problems. When solving bw_large.d formula, we stopped SDF after 30000 seconds.

Remark. QG problems contains several unit clauses. The local search procedures compared in tables 1 and 2 all simplify the input formula by satisfying unit clauses before the local search. It seems that the Walksat and Novelty procedures used in table 3 don't contain this simplification, which might explain the success rate difference of Novelty for QG problems. On other formulas which don't contain any unit clause, our implementation of Novelty reproduces the same success rate of the original Novelty.

4 Conclusion

We have proposed two extensions of the Walksat family local search procedures. The first extension is the diversification in Novelty so that in each search step, with probability dp the least recently flipped variable in a randomly selected unsatisfied clause c is picked to be flipped, while in the remaining cases, normal Novelty heuristic is used to pick the variable to flip in c. The new heuristic is called Novelty++. The diversification allows to weaken the determinism in Novelty which always picks one of the two best variables in c. It is also stronger than the random walk in Novelty+, since it deterministically picks the least recently flipped variable in c.

The second extension is the deterministic exploitation of so-called promising decreasing variables using a gradient-based approach. The new procedure is called G^2WSAT. We have combined G^2WSAT with Novelty++ such that whenever there

[2] After our experimentation is done, we are told that the new version (version 45) of Walksat has been speeded up (by 30-50% for large problems) using an improvement to the Walksat variable flip algorithm due to Alex Fukunaga (The search behavior (the assignments explored) is totally unaffected) [2].

are promising decreasing variables, the best (according to its score) of them is picked to be flipped, otherwise Novelty++ is used to pick the variable to flip.

Experimental results on a large number of random 3-SAT and structured problems show the performance of Novelty++ compared with Novelty and Novelty+ under different parameter settings, and the performance of G^2WSAT combined with Novelty++ compared with state-of-the-art local search procedures such as Novelty, Novelty+, Walksat, UnitWalk and SDF.

Similar to other heuristics in the Walksat family, Novelty++ is sensitive to noise and diversification parameters. An adaptive noise mechanism as presented in [6] might be used to automatically adjust the parameters when the search proceeds, in order to further improve the performance of Novelty++.

References

1. S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of 3rd Annual ACM Symp. Theory of Computing*, pages 151–158, 1971.
2. A. Fukunaga. Efficient implementation of sat local search. In *Proceedings of SAT-2004 (Seventh International Conf. on Theory and Applications of Satisfiability Testing*, Vancouver, British Columbia, 2004.
3. I. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of AAAI-93*, 1993.
4. E. A. Hirsch and A. Kojevnikov. Unitwalk: A new sat solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):91–111, 2005.
5. H. Hoos. On the run-time behavior of stochastic local search algorithms for sat. In *Proceedings of AAAI-99*, pages 661–666, 1999.
6. H. Hoos. An adaptive noise mechanism for walksat. In *Proceedings of AAAI-02*, pages 655–660. AAAI Press / The MIT Press, 2002.
7. W. Q. Huang and Jin Ren Chao. Solar: a learning from human algorithm for solving sat. *Science in China (Series E)*, 27(2):179–186, 1997.
8. C.M. Li and Anbulagan. Look-Ahead Versus Look-Back for Satisfiability Problems. In *Proceedings of CP-97, Third International Conference on Principles and Practice of Constraint Programming*, pages 342–356. Springer-Verlag, LNCS 1330, Shloss Hagenberg, Austria, 1997.
9. B. Mazure, L. Saïs, and É. Grégoire. Tabu Search for SAT. In *Proceedings of AAAI'97*, 1997.
10. D.A. McAllester, B. Selman, and H. Kautz. Evidence for invariant in local search. In *Proceedings of AAAI-97*, pages 321–326, 1997.
11. D. Schuurmans and F. Southey. Local search characteristics of incomplete sat procedures. *Artificial Intelligence*, 132(2):121–150, 2001.
12. B. Selman and H. Kautz. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proceedings of IJCAI-93*, 1993.
13. B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of AAAI-94, 12th National Conference on Artificial Intelligence*, pages 337–343. AAAI Press, Seattle, USA, 1994.
14. B. Selman, D. Mitchell, and H. Levesque. A new Method for Solving Hard Satisfiability Problems. In *Proceedings of AAAI'92*, pages 440–446, 1992.