UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo — Trento (Italy), Via Sommarive 14
http://dit.unitn.it/

REACTIVE SEARCH FOR MAX-SAT: DIVERSIFICATION-BIAS PROPERTIES WITH PROHIBITIONS AND PENALTIES

Roberto Battiti and Paolo Campigotto

July 2007

Technical Report # DIT-07-058

Reactive search for MAX-SAT: diversification-bias properties with prohibitions and penalties

Roberto Battiti and Paolo Campigotto
DISI - Dipartimento di Ingegneria e Scienza dell' Informazione,
Università di Trento

July 2007

Abstract

Many incomplete approaches for SAT and MAX-SAT have been proposed in the last years. The objective of this investigation is not so much horse-racing (beating the competition on selected benchmarks) but understanding the qualitative differences between the various approaches by analyzing simplified versions thereof. In particular, we focus on *reactive search* schemes where task-dependent and local properties in the configuration space are used for the dynamic on-line tuning of local search parameters.

We consider the choice between prohibition-based and penalty-based reactive approaches, and the choice between considering all variables or only the variables appearing in unsatisfied clauses. On simplified versions we consider the trade-off between diversification and bias after starting from a local minimizer, the so called D-B plots. We then consider long runs of the complete algorithms on selected MAX-SAT instances, by measuring both the number of iterations (flips) and the CPU times required by the single iterations with efficient data structures.

The results confirm the effectiveness of reactive approaches, in particular when combined with non-oblivious objective functions. Furthermore a complex non-linear behavior of penalty-based schemes is observed.

1 Introduction

Most of the stochastic local search (SLS) algorithms for SAT and MAX-SAT are characterized by a set of parameters whose tuning is crucial in term of CPU time requirements and solution quality, see for example the review in [10]. However, the appropriate tuning depends on both the problem and the current task being solved, implying costly human intervention. Furthermore, the appropriate configuration can vary widely in different regions of the configuration space around a given tentative current solution, leading to dynamic adaptive schemes.

Reactive search strategies for the on-line dynamic tuning of these free parameters to the current task being solved and to the local characteristics can be used to obtain more robust and efficient techniques [3].

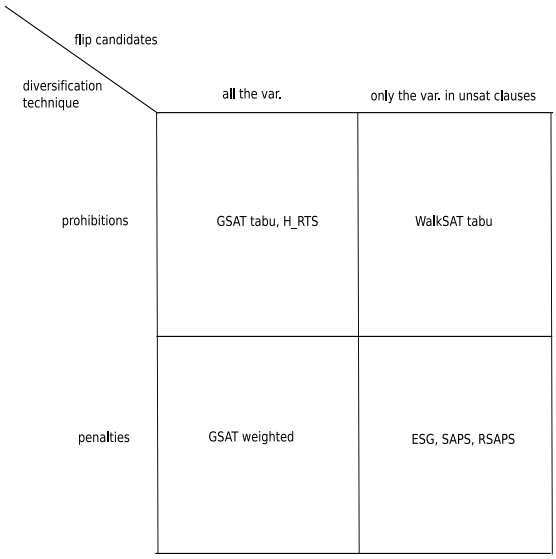

Figure 1: A classification of different SLS schemes for SAT considered in this paper along the two dimensions: *all* variables versus *unsatisfied* variables, prohibition-based versus penalty-based.

The scope of this paper does not allow a detailed review, see for example [5] for a recent survey of propositional satisfiability and the related constraint programming problem, and [10] (in particular Chap. 7-8) for a survey of stochastic local search approaches for SAT and MAX-SAT. Let us focus onto reactive schemes and let us classify them according to the target acted upon during the on-line adaptation. In detail, the reaction can be on the generation of a set of *constraints on the variables* through the *prohibition* of recently-applied moves (tabu search), or on the *modification of the cost function* guiding the local search. For brevity, the two paradigms will be denoted as *prohibition-based* and *penalty-based*, respectively. The first method aims at pushing the configuration out from the attraction basin around a local minimizer but temporarily prohibiting some moves which would lead the trajectory back to the starting point. The second method, also termed "dynamic local search," modifies the objective function guiding the search so that a local minimum is raised to encourage the exploration of different areas, see Fig. 2.

A second macroscopic difference is given by the selection of the variables considered during each local search step. In the basic schemes (like GSAT), all variables are potential candidates for the next flip, in more recent proposals (like walksat), only the *variables appearing in unsatisfied clauses* are considered

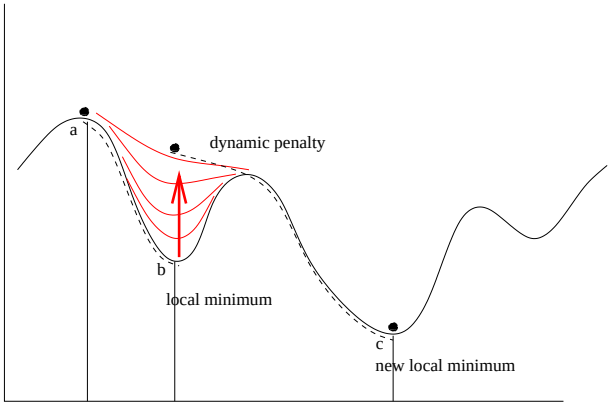

Figure 2: Transformation of the objective function to gently push the solution out of a given local minimum, derived from [3].

for a possible flip. With a slight terminology abuse, but in the interest of brevity, we will denote as *unsatisfied variables* the set of variables which appear in unsatisfied clauses given the current truth assignment.

Given the space constraints of this paper we report selected results within an ongoing investigation with the following aims:

- to compare and identify qualitative differences between prohibition and penalty-based schemes.
- to better understand the role of the *non-oblivious* search considered in [2, 1] in differentiating between solutions belonging to a given plateau. Looking at the internal structure of the solution and aiming at a redundant satisfaction of the clauses (more than a single matched literal) can hint at proper directions to follow on a plateau to facilitate the eventual discovery of an improving move.
- to compare schemes acting on all variables with schemes acting on unsatisfied variables. There is a "rule of thumb" that considering all variables results in less iterations but longer CPU-times per iteration, see for example [10], so that the potential reduction in the number of search steps gained by the complete neighborhood examination before selecting the next move can be wasted by the longer CPU times required per iteration.
- to distinguish very clearly between results as a function of the number of iterations and as a function of CPU times. The former are of high interest because they refer to how the information derived during the search is used effectively to minimize the number of steps, the latter are of course of interest for the final user, not interested in theoretical results but in minimizing solution times.

In the next section, we describe the algorithms considered in this work: GSAT and GSAT/tabu, WalkSAT/SKC, WalkSAT/TABU, AdaptNovelty$^+$ and SAPS.

In subsection 2.3, we describe the Hamming-based Reactive Tabu Search algorithm. In section 4, we analyze the diversification-bias results. Finally, in section 5 we present a selection of the experiments on long runs of the considered candidates algorithms, while in section 6 we concentrate on CPU times.

2 Stochastic local search and reactive approaches for MAX-SAT

The simplest cost function guiding stochastic local search (SLS) algorithms for MAX-SAT is the number of unsatisfied clauses for a given variables truth assignment. This function will be denoted as f in the following. Escaping local minima of f in an intelligent manner can be considered as the underlying motivation of most reactive search approaches.

In many cases, local minima are actually large *plateaus* where the algorithm cannot determine the "right" direction to continue its search and a random selection among equivalent neighbors is performed. The ensuing random walk among the set of moves preserving the f value doesn't encourage a fast traversal of plateau regions: the lack of a direction guiding the search makes the algorithm wander in an aimless fashion over these broad flat regions.

A remedy consists of transforming f into a modified g cost function, therefore *tilting* the plateau surface, and generating *hints* of a proper direction of movement. This cost function modification may look at the *internal structure* of the current solution, not simply at the number of satisfied clauses. In [1] one exploits *non-oblivious* cost functions, which measure the *degree* of satisfaction of each clause by counting the number of matched literals. Aiming at a redundant satisfaction eliminates the embarrassment in selecting among seemingly similar situations and may eventually permit to flip a variable to satisfy a new clause, without loosing any already satisfied clause.

Another approach to escape from local minima or plateaus of f is given by *dynamic local search*, that relies on a reactively weighted version of the oblivious function. The work in [21] uses weights to encourage the satisfaction of "more difficult" clauses. Clause weighting is motivated in [21] in order to "fill-in" local minima.

The dynamic modification of the cost function is not the only opportunity for the application of reactive techniques into SLS algorithms for SAT. In [1], the reactive search paradigm is applied to SAT by dynamically adjusting the prohibition parameter. In [11], an adaptive mechanism for walkSAT is developed to dynamically adjust the value of the noise parameter in the walkSAT algorithm, which determines if a random walk step rather than a greedy step must be executed. In Adaptive Novelty$^+$ [28] a similar mechanism for noise adaptation is applied to the Novelty$^+$ algorithm. Adaptive Novelty$^+$ is considered one of the best-performing SLS algorithms for SAT [10].

Finally, in [6] a technique to learn an evaluation function for local search algorithms from features of the search space points visited during search is presented. The learned evaluation function is used to bias future search trajectories toward better optima on the same problem. The results of the application of this approach to SAT instances are reported in [6].

In the next subsection, the GSAT and GSAT/tabu algorithms are described. An overview of the WalkSAT/SKC, WalkSAT/TABU and AdaptNovelty$^+$ algorithms follows. To conclude the section, the Hamming Reactive Tabu Search and the Scaling and Probabilistic Smoothing algorithms are depicted.

2.1 GSAT and GSAT/tabu algorithms

Even if its performance is not competitive with the more recent SLS algorithms, the GSAT algorithm [24] was one of the first SLS algorithms for SAT. It also creates a touchstone for the other algorithms. GSAT is a local search greedy strategy, that at each search step tries to improve the solution by flipping the variable that, when flipped, leads to the maximum decrease in the number of unsatisfied clauses. I.e., for each variable v it calculates:

- the number $loose_v$ of clauses currently satisfied that would become unsatisfied if the variable v is flipped;
- the number $gain_v$ of clauses currently unsatisfied that would become satisfied if the variable v is flipped;

and greedily selects uniformly at random one variable v minimizing the quantity $loose_v - gain_v$. Let f the score function for the GSAT algorithm, which simply counts the number of unsatisfied clauses. Then, the quantity $loose_v - gain_v$ is called Δf. minimizing Δf, GSAT randomly picks one of them. In order to escape from local minima, it adopts a periodic restart mechanism that reinitializes the search step after a specific number of flips has been executed. The pseudocode for the GSAT algorithm is in fig. 3. command usage:

At the time of this writing, GSAT/Tabu [27] is one of the best-performing variants of GSAT. It simply enriches the GSAT algorithm via a tabu search criterion, that avoids the current flipped variable to be flipped back for the next T search steps. The T parameter is the prohibition parameter. Its setting is a crucial point for the algorithm performance: a low T value does not allow for easily escaping from local minima with great attraction basins, while a high T value avoids search intensification in a promising region because too many variables are tabu.

2.2 WalkSAT/SKC, WalkSAT/TABU and AdaptNovelty$^+$

The ancestor of WalkSat family algorithms is the WalkSat/Skc algorithm [22]. It was created to allow a fast escaping from local minima, without necessarily requiring a random re-inizialization of the search, like in the case of GSAT algorithm. To do this, WalkSat algorithms randomly alternate between greedy

```
1.  procedure GSAT
2.  ⌈ input: a set of clauses S
3.  | output: a satisfying truth assignment for S , if found
4.  | let MAX_TRIES the maximum number of allowed trials
5.  | let MAX_FLIPS the maximum number of allowed flips for each trial
6.  |
7.  | for i = 1 to MAX_TRIES do
8.  | ⌈ select a random truth assignment T
9.  | | for j = 1 to MAX_FLIPS do
10. | | ⌈ if T satisfies S then
11. | | |     return T
12. | | | otherwise
13. | | | ⌈ select at random one variable with the minimum Δf value
14. | ⌊ ⌊ ⌊ flip the selected variable
15. ⌊ return no assignment found
```

Figure 3: The pseudocode for the GSAT algorithm.

minimizing moves and random noisy moves. The moves of both kinds are randomly selected from the variables appearing in unsatisfied clauses. At each iteration, WalkSAT/Skc first randomly chooses an unsatisfied clause, and then selects a variable to flip within the clause. The variable is selected applying the heuristic in fig. 4.

```
1.  let wp the noise setting parameter
2.  for all variables in the selected clause do
3.  ⌈ calculate the number loose_v of clauses currently satisfied that would
4.  ⌊ become unsatisfied if the variable v is flipped
5.
6.  if a variable v with loose_v = 0 exists then
7.      flip it /* zero damage step */
8.  otherwise
9.  ⌈ with probability 1 − wp select randomly one variable v with the
10. | minimum value for loose_v and flip it /* greedy step */
11. | with probability wp select randomly a variable from the clause
12. ⌊ and flip it /* random walk step */
```

Figure 4: The selection of the variable to flip in the WalkSAT algorithm.

Note the difference with GSAT algorithm, that at each iteration globally determines the best move (i.e., it do not perform any clause selection to determine the variable to flip).

The AdaptiveWalkSat [11] algorithm dynamically adjusts the value w_p during the search. For our experiments, we used the WalkSAT family AdaptNovelty$^+$

algorithm [28], that is one of the most performing and robust SLS algorithm for SAT currently known. It is the "reactive" version of Novelty$^+$. On its turn, Novelty$^+$ is the enhanced version of "Novelty" algorithm. It exploits the concept of variable "age" and, differently from WalkSat/SKC, it uses the same scoring function of GSAT. The age of variable v is simply the number of search steps (i.e., variable flips) that have been performed since v has been flipped last. Whenever v is flipped, its age is reset to 0 and increased by 1 with every subsequent search steps. Analogously to GSAT description (see sec. 2.1), let $loose_v$ the number of newly unsatisfied clauses and $gain_v$ the number of newly satisfied clauses if the variable v is flipped and let $\Delta f = loose_v - gain_v$, where f is the score function. The search step performed by Novelty$^+$ is described by the self-explanatory pseudocode in fig. 5.

```
1.  let w_p the walk probability parameter
2.  let p the noise setting parameter
3.  let age_v the age of variable v
4.
5.  choose an unsatisfied clause c;
6.  with probability 1 − wp do
7.      if the variable in c with the the minimum Δf does not have minimal
8.          age then
9.              flip it
10.     otherwise
11.         with probability 1 − p do
12.             flip it
13.         with probability p do
14.             flip the second best variable
15. with probability wp do /* random walk step */
16.     select a random variable in c
17.     flip it
```

Figure 5: The pseudocode for the Novelty$^+$ algorithm.

The AdaptNovelty$^+$ algorithm has been designed to dynamically adjust the noise parameter p based on search progress. In fact, the optimal setting for p is instances-dependent: even small deviations from the optimal value can lead to substantially decreased performance [11]. At the beginning of the search, AdaptNovelty$^+$ sets p to 0. This setting (and, in general, low values for p) allows for a greedy search, resulting in rapid improvement for the evaluation function value. Whenever the algorithm gets stuck in a local minimum, p is increased, such that the search is diversified. The noise parameter keeps increasing until the search process overcomes the stagnation situation. From now on, p gradually decreases leading to an increase in search intensification. To detect search stagnation, the AdaptNovelty$^+$ algorithm exploits the search history. In particular, if no improvements in the score function (i.e., in the number of un-

satisfied clauses) is observed within the last search steps, the search stagnation is declared.

The WalkSAT/TABU algorithm [16] adopts the same score function and the same two stage variables selection mechanism of the WalkSAT/SKC algorithm described above. However, different from all the others WalkSAT family algorithms, it has not the noise parameter. Furthermore, as the name itself suggests, a tabu search method is exploited to constrain to choose the least recently flipped variables.

2.3 H-RTS: Reactive tabu search guided by non-oblivious functions

The *non-oblivious* functions (NOB) introduced in [25] to obtain better approximation ratios represent a finer-grained approach with respect to the standard function f (called *oblivious*), taking into account also the "degree" of satisfaction of the clauses. Given an assignment X, let S_i denote the set of clauses in the given task in which exactly i literals are true and let $w(S_i)$ denote the cardinality of S_i. Let *"n"* the number of variables of the input SAT instance. In addition, a d-neighborhood of a given truth assignment is defined as the set of all assignment where the value of at most d variables is changed. The performance ratio for any oblivious local search algorithm with a d-neighborhood for MAX–2–SAT is 2/3 for any $d = o(n)$, while non-oblivious local search with an 1-neighborhood achieves a performance ratio 3/4, see [25]. The performance ratio is improved even if the search is restricted to a much smaller neighborhood. The oblivious function for MAX–k–SAT is of the form:

$$f_{NOB}(X) = \sum_{i=1}^{k} c_i w(S_i)$$

and requiring a best performance ratio for local search determines $\Delta_i = c_{i+1} - c_i$ as:

$$\Delta_i = \frac{1}{(k-i+1)\begin{pmatrix} k \\ i-1 \end{pmatrix}} \left[\sum_{j=0}^{k-i} \begin{pmatrix} k \\ j \end{pmatrix} \right]$$

Because the positive factors c_i that multiply $w(S_i)$ in the function f_{NOB} are strictly increasing with i, the approximations obtained through f_{NOB} tend to be characterized by a "redundant" satisfaction of many clauses. LS-NOB achieves a performance ratio $1 - \frac{1}{2^k}$ for MAX–k–SAT.

Beyond possessing a better worst-case behavior, in [1] also the average performance of the NOB functions is empirically demonstrated to be better with respect to the standard OB functions. I.e., the NOB functions lead to local optima of better average quality with respect to the standard OB functions, at least for the benchmark of random 3-SAT instances used for the experiments in [1]. The good average performance of non-oblivious functions is a crucial point for their usage in the framework of heuristics.

In [1], first an integration of the standard and the NOB functions in a simple local search scheme is studied. The developed scheme adopts a combined two-phase local search strategy: the search is initially guided by a NOB function and, once a local minimum is found, an OB function is used followed by a plateau search phase to further investigate the search space around its first local minimum. At the base of this approach, the fact that local optima of the standard cost function are not necessarily local optima of the different cost function. Based on the promising results obtained by this simple hybrid schema, the work in [1] proposes an integrated heuristic (Hamming-based Reactive Tabu Search, H-RTS for short) that integrates Tabu Search with the periodic activation of non-oblivious functions and the "reactive" triggering of a strong diversification phase (see the pseudocode in Fig. 7). The Hamming distance among the search space points is used as a diversification trigger.

The initial truth assignment for H-RTS is generated in a random way, and NOB local search is applied until the first local optimum of f_{NOB} is encountered. LS-NOB obtains local minima of better average quality than LS-OB, but then the guiding function becomes the standard oblivious one. This choice is motivated by the success of the NOB & OB combination and by the poor diversification properties of NOB alone, see [1].

The search proceeds by repeating phases of local search followed by phases of tabu search (TS) (lines 8–17 in Fig. 7), until 10 n iterations are accumulated. The variable t, initialized to zero, contains the current iteration and increases after a local move is applied, while t_r contains the iteration when the last random assignment was generated. During each combined phase, first the local optimum of f is reached, then $2(T+1)$ moves of Tabu Search are executed. The design principle underlying this choice is that prohibitions are necessary for diversifying the search only after local search (LS) reaches a local optimum. Finally, an "incremental ratio" test is executed to see whether in the last $T+1$ iterations the trajectory tends to move away or come closer to the starting point. A possible reactive modification of T_f is executed depending on the tests results, see the procedure $REACT$ in fig. 6 for details. The fractional prohibition T_f (the prohibition T is obtained as $T_f\ n$) is therefore changed during the run to obtain a proper balance of diversification and bias.

The random restart executed after 10 n moves guarantees that the search trajectory is not confined in a localized portion of the search space.

Currently, there is no detailed comparison among the H-RTS approach and the best-performing SLS algorithms for SAT and MAX-SAT proposed in the last years. The last sections of this paper aim at covering this gap.

2.4 Dynamic local search algorithms

The Dynamic Local Search Algorithms (DLS) rely on penalizing solution components in order to allow the algorithm to escape from local minima In the case of SAT, the solution components are the clauses constituting the input CNF formula. A positive weight is associate to each clause, and these weights are dynamically modified during algorithm execution. In particular, whenever

procedure REACT($T_f, X^{(t)}, X^{(t-2(T+1))}$)
/* REACT: feedback scheme to adjust the prohibition T_f */
{ *Returns the updated prohibition* T, T_f *is the reference to the current fractional prohibition*}

1 $deriv \leftarrow \frac{H(X^{(t)}, X^{(t-2(T+1))}) - (T+1)}{T+1}$
2 **if** $deriv \leq 0$ **then** $T_f \leftarrow T_f + \frac{1}{100}$
3 **else if** $deriv > \frac{1}{2}$ **then** $T_f \leftarrow T_f - \frac{1}{100}$

4 **if** $T_f > \frac{1}{4}$ **then** $T_f \leftarrow \frac{1}{4}$
5 **else if** $T_f < \frac{1}{40}$ **then** $T_f \leftarrow \frac{1}{40}$

7 **return** $\max\{\lfloor T_f n \rfloor, 4\}$

Figure 6: The *REACT* function of the H-RTS algorithm. The pseudocode is taken from [1].

the algorithm gets stuck in a local minima, the penalty weights of the solution components are increased, leading to the an increase of the evaluation function value for the solution. Since the neighbours of the solution might not have been affected at the same extent, improving steps are now possible. Therefore, the DLS algorithms use a modified version of the "standard" score function f that counts the number of unsatisfied clauses. The score function wf for the DLS algorithms typically calculates the total weight of the unsatisfied clauses under a given truth assignment. Among the big family of DLS algorithms, for our experiment we chose the Scaling and Probabilistic Smoothing (SAPS) algorithm [29], as it is one of the most recent and performing algorithms. SAPS in an improved version of the Exponentiated Subgradient algorithm [20]. The pseudocode for the SAPS algorithm is at the end of this subsection. SAPS starts its investigation from a random truth values variable assignment. Initially, all clause weights are equal to one. For each subsequent search step, it selects the variable to be flipped uniformly at random from the set of the variables appearing in the currently unsatisfied clauses and causing the biggest reduction in the total weight of *unsatisfied* clauses. When a local minimum is reached, i.e. a truth assignment where flipping any variable appearing in an *unsatisfied* clause causes no decrease in the total weight of the *unsatisfied* clauses, two cases arise:

1. with probability η the search is continued by flipping a variable chosen uniformly at random among all the variables in currently unsatisfied clauses;

2. with probability $1 - \eta$, the search process is terminated and the SAPS weight update procedure is started.

During the SAPS weight update procedure, first the weights of currently unsatisfied clauses are scaled via multiplication by a factor α. Then, with probability p_{smooth} , the weights of all clauses are smoothed towards the average clauses

```
procedure HAMMING-REACTIVE-TABU-SEARCH
1.  repeat
2.      t_r ← t
3.      X ← random truth assignment
4.      T ← ⌊T_f n⌋
5.      repeat                                   { NOB local search }
6.          X ← BEST-MOVE (LS, f_NOB)
7.      until largest Δf_NOB = 0
8.      repeat
9.          repeat                               { local search }
10.             X ← BEST-MOVE (LS, f_OB)
11.         until largest Δf_OB = 0
12.         X_I ← X
13.         for 2(T + 1) iterations              { reactive tabu search }
14.             X ← BEST-MOVE (TS, f_OB)
15.         X_F ← X
16.         T ← REACT(T_f, X_F, X_I)
17.     until (t − t_r) > 10 n
18. until solution is acceptable or maximum number of iterations reached
```

Figure 7: The H-RTS algorithm.

weight via the formula:

$$w_i = w_i * \rho + \bar{w} * (1 - \rho) \tag{1}$$

where w_i is the current weight of the i-th clause, $\bar{w}$ is the average weight of all clauses after scaling and ρ is a fixed parameter ranging in $(0, 1)$. With probability $1 - p_{smooth}$, the smoothing phase is skipped. RSAPS is a reactive variant of SAPS, adjusting the the smoothing probability p_{smooth} during the search. However, RSAPS cannot self-tune all its parameters: in particular, the parameter ρ still needs to be set by hand. The self-explanatory pseudocode for the SAPS algorithm is in fig. 8.

```
1.  procedure updateWeights ()
2.      input: clause weights vector W
3.      let α the scaling factor
4.      let ρ the smoothing factor
5.      let P_smoothing the smoothing probability
6.
7.      /* scaling stage */
8.      for each unsatisfied clause i do
9.          W[i] = W[i] * α
10.
11.     /* probabilistic smoothing stage */
12.     with probability P_smoothing do
13.         calculate mean weight w̄ among all clauses
14.         for each clause i do
15.             W[i] = W[i] * ρ + (1 − ρ) * w̄
16. procedure local_minimum_for_wf
17.     output: true if the current state is a local minum, false otherwise
18.
19.     if for each variable v in unsatisfied clauses there is no reduction
20.         Δwf_v in the total weight of unsatisfied clauses when flipped then
21.                 return true
22.     otherwise
23.                 return false
24. procedure SAPS
25.     input: a set of clauses S
26.     output: a satisfying truth assignment for S , if found
27.     let MAX_TRIES the maximum number of allowed trials
28.     let MAX_FLIPS the maximum number of allowed flips for each trial
29.     let w_p the random walk probability
30.     let W the clause weights vector
31.
32.     for each clause i do
33.         W[i] = 1
34.     for i = 1 to MAX_TRIES do
35.         select a random truth assignment T
36.         for j = 1 to MAX_FLIPS do
37.             if T satisfies S then
38.                 return T
39.             otherwise
40.                 for all variables v in unsatisfied clauses do
41.                     calculate the reduction Δwf_v in the total weight of unsatisfied
42.                     clauses if the variable v is flipped
43.                 if ( local_minimum_for_wf () ) then
44.                     with probability w_p do
45.                     select randomly a variable in unsatisfied clauses
46.                     flip the selected variable
47.                     with probability 1 − w_p do
48.                     updateWeights ()
49.                 otherwise
50.                     select the variable b with the highest ΔW_b value
51.                     flip the selected variable
52.     return no assignment found
```

Figure 8: The pseudocode for the SAPS algorithm.

3 Benchmark problems

Our tests are dedicated to MAX-3-SAT instances. The benchmark suite used has been created via the generator of random instances was obtained from B. Selman.

The SAT instances corresponding to some ratios of clauses to variables are satisfiable with a very large probability. This result is clearly explained by the analysis of [18]. For this reason, and in order to focus onto the most relevant data for MAX-3-SAT, only the non-trivial cases of 300 variables and 1500 clauses and of 500 variables and 5000 clauses are considered in the following experimental part.

In detail, if $n : m$ identify variables and clauses, 50 instances for the 500:5000 and 300:1500 cases has been randomly randomly generated. The different algorithms are run 10 times for each instance, and therefore also different random initial assignments. The total number of tests is therefore 500 and, unless specified in a different way, the mean results are averages of the 500 runs.

4 Diversification-bias analysis

The purpose of this exploration is to understand how the two basic reactive schemes acting on prohibitions and penalties lead the trajectory away from a local minimum. Are the two possibilities different ways to generate a similar dynamical system (a trajectory with statistically similar properties) or is there any qualitative difference? We consider *skeletal* version of the techniques, so that many complex ingredients are not added and the basic mechanisms act in isolation and we follow the diversification-bias empirical analysis ("D-B plots") proposed in [1]. Thus, the metric used to measure the quality of the visited points (or, simply, the bias of the algorithm) is the cost function value over those points, while the diversification is measured via the Hamming distance. Clearly, adopting these metrics, the best algorithms are the one that realize the best compromise between diversification and bias: they greatly diversify their search while visiting points with low cost function values. When a local search algorithm is started, new points are visited at each iteration until the first local optimum is encountered, because the number of satisfied clauses increases by at least one. During this phase additional diversification schemes are not necessary and potentially dangerous, because they could lead the trajectory astray, away from the local optimum. The compromise between bias and diversification becomes critical after the first local optimum is encountered. In fact, if the local optimum is strict, the application of a move will worsen the cost function value, and an additional move could be selected to bring the trajectory back to the starting local optimum. Even if the local optimum is not strict (plateau regions are typical for SAT) there is no guarantee that a simple local search algorithm will not produce a localized trajectory, for example such that its maximum Hamming distance from the first encountered local minimum is bounded by a value much less than the number of variables of the input SAT instance (i.e.,

the maximum possible hamming distance value).

The mean bias and diversification after starting from a local minimum depend on the value of the internal parameters of the different algorithms. In order to isolate the effect of these parameters, a series of tests is executed where all other experimental conditions are unchanged and only a single parameter is changed.

All runs of the algorithms considered proceed as follows: as soon as the first local optimum for the "standard" f function is encountered, it is stored and the algorithm is then run for additional $4 * n$ iterations. The final D-B values are averaged over 500 tests are reported. In order to produce a fair comparison among the algorithms, for each run 200000 search steps are executed, without random restarts that could corrupt the experiment statistics. In fact, note that we are interested in the behaviour of the SLS algorithms after the first local minimum is met, skipping the particular cases happening when the first local minimum is discovered some steps before a scheduled random re-initialization is executed. However, we experimentally verify that no algorithms of the WalkSAT family may be involved in this experiment, even if some of them are considered extremely performing (see section 2). In fact, the algorithms of the WalkSAT family (with walk probability parameter equal to 0.5) cannot discover a local minimum for the instances of the adopted benchmark, even if 10000000 instead of 100000 iterations are executed for each run. We argue this is due to the fact that, with respect to other SLS SAT solvers, WalkSAT family algorithms heavily rely on "noisy" moves [23].

Therefore, in order to allow WalkSAT-based-algorithms "to take part to the game", for each test we identify the first local minimum via GSAT algorithm. Then, depending on the different test, we run one among the following alternatives: GSAT/tabu, GSAT (weighted version), WalkSAT/tabu and WalkSAT (weighted version), starting from the the discovered local minimum. Constraining all the algorithms to start from the same (local minimum) point allows to "make the competition more fair". Clearly, the quality of the first local minimum may affect the actions performed by the algorithm in the subsequent steps. Furthermore, the random initial assignment constituting the starting point for the algorithm heavily biases the first local minimum that will be discovered. Therefore, a random initial assignment may influence the results of the search, and, as a consequence, the results of the experiment, even if we stress that the final values of our experiments are an average over 50 input instances. In addition, if the algorithms begin their search from the same local minimum point, they perform exactly the same number of iterations during the experiment.

The tests presented in this work are dedicated to selected MAX-3-SAT instances defined in [18] (see section 3). The different algorithms are run for the 500:5000 and 300:1500 instances randomly generated, for a total of 500 tests in both cases. The average results are presented.

First, we evaluate a method based on fixed prohibitions, the GSAT/tabu algorithm. Then we study the performance of a "weighted" approach, which is a simplified version of the weighted GSAT algorithm. Initially all clause weights are equal to one and, once the first local optimum is encountered, the weights

of the currently unsatisfied clauses are increased by a fixed quantity Δw.

Fig. 9 shows the results obtained by running the considered algorithm for $4 * n$ steps after the first local minimum discovered by the GSAT algorithms for the same SAT instances. The labels for the curve in the figure represent the different Δw values considered. The value 0 for the "GSATweighted" curve represents the case of the original GSAT algorithm [24].

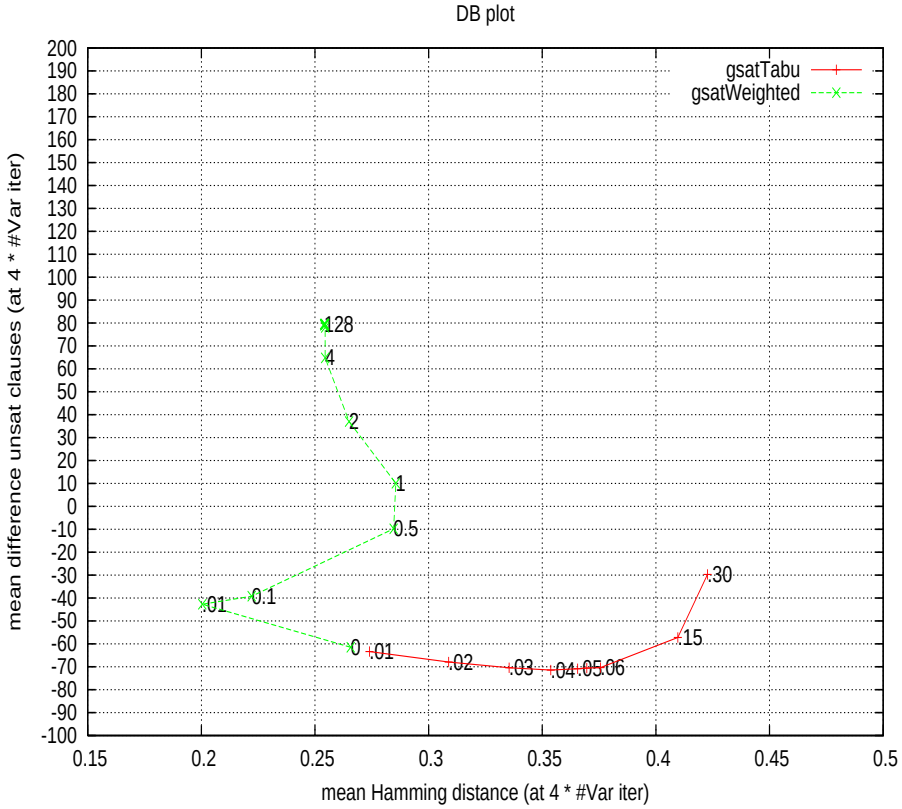

Figure 9: Diversification-bias plots of prohibition and penalty-based strategies. In the first case, the curve is labeled with the Δw values for the weights update (points for $\Delta w = 8, 16, 32, 64, 128$ are at a very similar position), in the second case with the fractional prohibition T_f.

The bias is plotted as *difference* w.r.t. the starting f value at the local minimum, the Hamming distance is divided by the number of variables n. A remarkably different behavior can be observed between the prohibition and the penalty-based approaches. With prohibitions, the diversification effect is as expected: a larger prohibition is related to a larger diversification (larger Hamming distances reached). Furthermore, for a wide range of T_f values (approximately

between 0.01 and 0.1) the D-B values are Pareto-optimal: both a larger diversification *and* a better bias are obtained. For $T_f = 0.05$, after the first local optimum, on average 70 additional clauses are satisfied and Hamming distance equal to 0.37 n is reached. On the contrary, for the penalty-based scheme the behavior is quite complex and non-linear. At the beginning, when Δw increases from zero to 0.5 the diversification increases *but* the bias worsens (from $\Delta f = -45$ to $\Delta f = -10$). Then the bias keeps worsening in a drastic manner and the diversification is actually *decreased*. A limiting situation of $\Delta f = 80$ and Hamming distance 0.26 n is reached for Δw values bigger than 8.

Note that, even if in Fig. 9 the curve for the "weighted" approach is called "GSAT-weighted", we stress that the algorithm used for the experiment is a specific version of the GSAT weighted algorithm, that performs weights updating only at the first local minimum discovered during the search, rather than at each random restart.

Fig. 10 shows the results for the same experiment executed in Fig. 9, except that the algorithms are kept running only for $n/4$ steps from the first local minimum encountered. Note that the behavior of the weighted and the tabu-based approaches is coherent with the results for the $4 * n$ case.

In the experiment depicted in Fig. 11, we consider the same approaches as for Fig. 9. However, in this case the algorithms acts only over the *unsatisfied* variables. We consider a version of the WalkSAT/SKC algorithm (called "WalkSATtabu_f" in Fig. 11), which is driven by the same score function f of the GSAT algorithm. We develop also a weighted variant of it, called "WalkSAT_wf". The probability of performing random walk steps for the algorithm called "WalkSAT_wf" has been fixed to 0.5.

The results are again surprising. The prohibition-based version confirms the relationship between larger prohibition and larger diversification, although in this case a larger diversification is paid by a rapidly worsening bias. In general, if one compares these results with the ones of the previous Fig. 9, the Δf values are inferior (at best $\Delta f = -11$ is obtained).

For the penalty-based case, again a strongly non-linear effect is observed. For Δw values growing from zero to 0.5 the diversification increases and the bias worsens up to $\Delta f = 11$, then the diversification dramatically worsens (e.g. it goes from 0.46 n to 0.28 n for $\Delta w = 2$) to reach a limiting value of 0.18 n for large weight increases.

Fig. 12 shows the results for the same experiment executed in Fig. 11, except that the algorithms are kept running only for $n/4$ steps from the first local minimum encountered. Again, the behavior of the weighted and the tabu-based approaches is coherent with the results for the $4 * n$ case.

In Fig. 13 and Fig.14, you can find the behavior of the tabu and weighted approach and of the tabu and weighted approach based on the modified versions of WalkSAT algorithm, respectively, taking into account the first local minimum after $10 * n$ steps of local search, where the greedily selected variable to flip is always accepted, even if the f value remains equal or worsens. For all the considered algorithms, their behavior confirms the results of the analysis for the experiment considering the first local minimum.

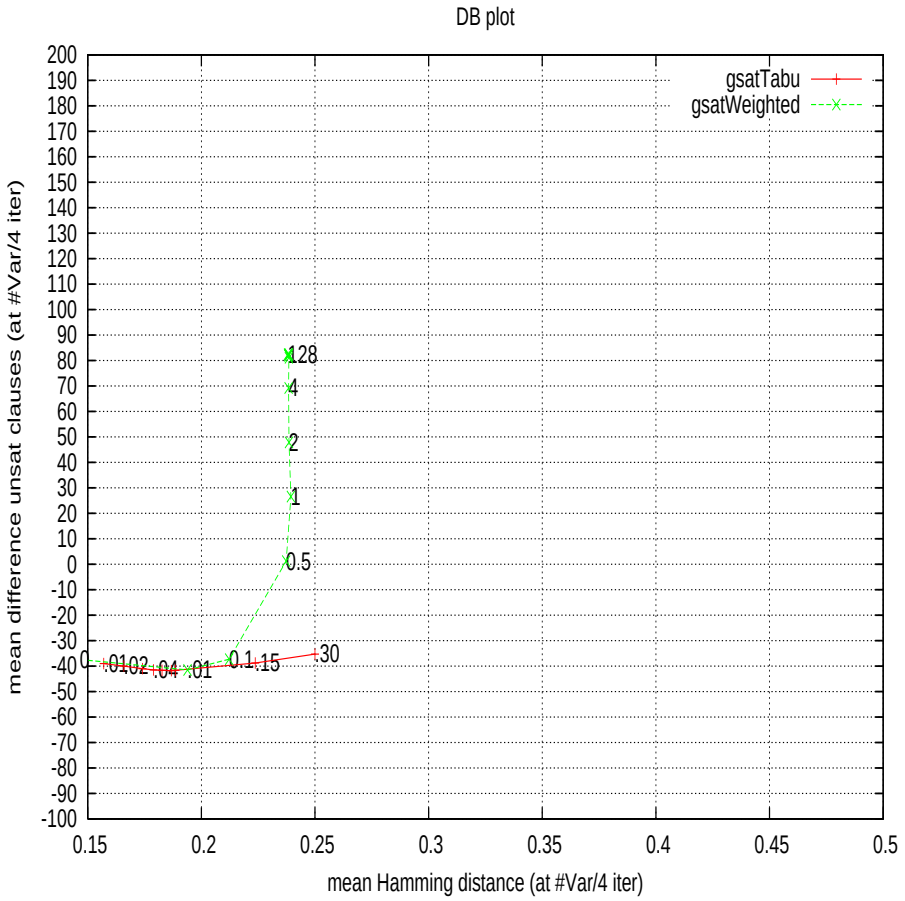

Figure 10: Diversification-bias plots measured at distance $n/4$ of prohibition and penalty-based strategies. In the first case, the curve is labeled with the Δw values for the weights update (points for $\Delta w = 8, 16, 32, 64, 128$ are at a very similar position), in the second case with the fractional prohibition T_f.

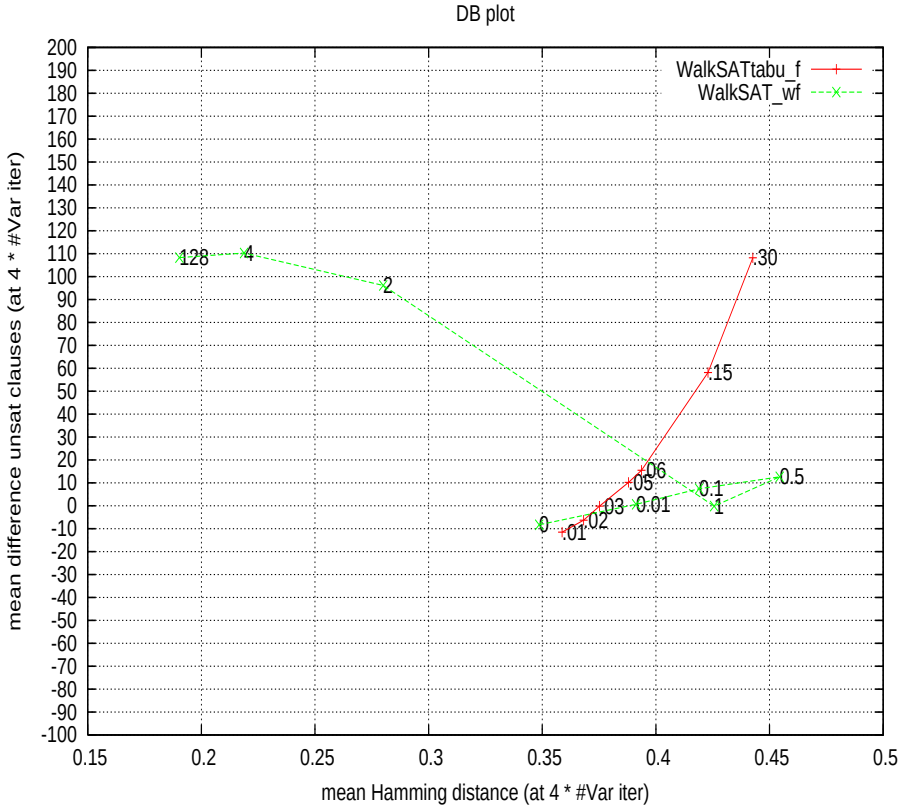

Figure 11: The diversification-bias performance of the modified versions of weighted WalkSAT and WalkSAT-tabu algorithms using the score function f of the GSAT algorithm.

These D-B results on simplified schemes suggest that prohibition-based schemes acting on all variables are characterized by a higher level of robustness and an overall better compromise between diversification and bias. In particular, it may be the case that penalties can be more dangerous than prohibitions because of the possible interference between the original function f and the modified function g. For example, pushing up a given local minimum by the weighting mechanism can hide other unexplored local minima. The superiority of more complex weighting schemes based either on weight decay (forgetting) or redistributions can be related to curing these complex interference effects.

In the next section we will consider the original complete schemes, including the reactive and dynamic versions and analyze the average f values obtained for long runs.

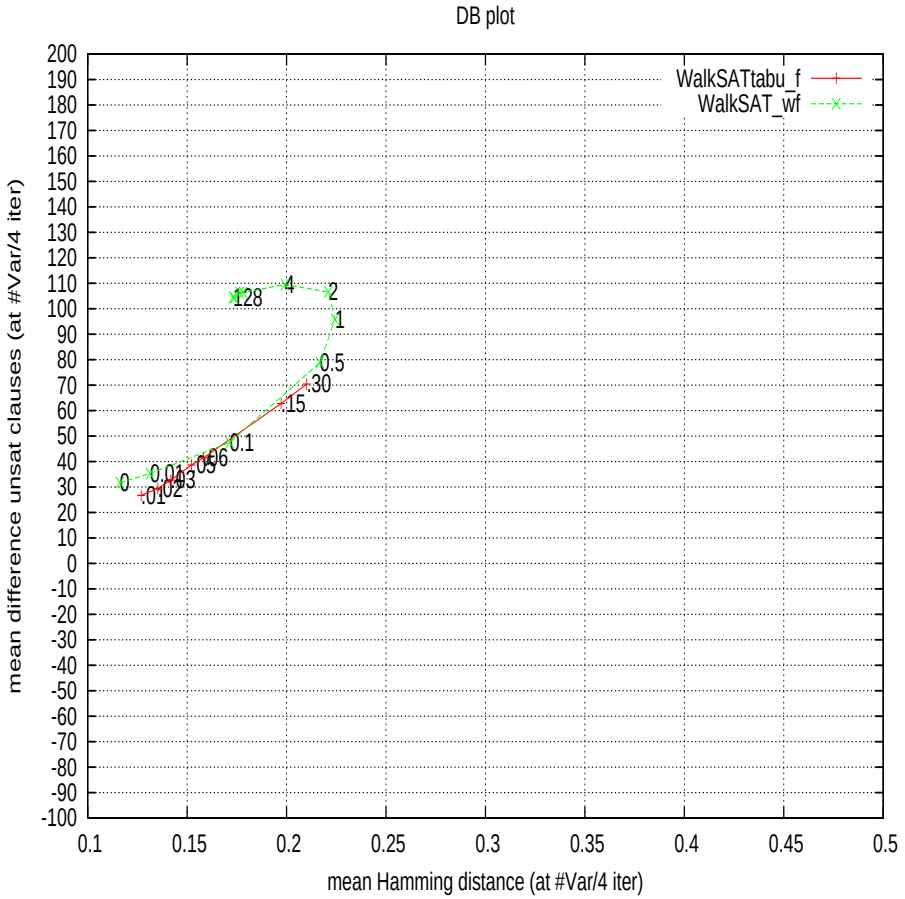

Figure 12: The diversification-bias performance measured at distance $n/4$ of the modified versions of weighted WalkSAT and WalkSAT-tabu algorithms using the score function f of the GSAT algorithm.

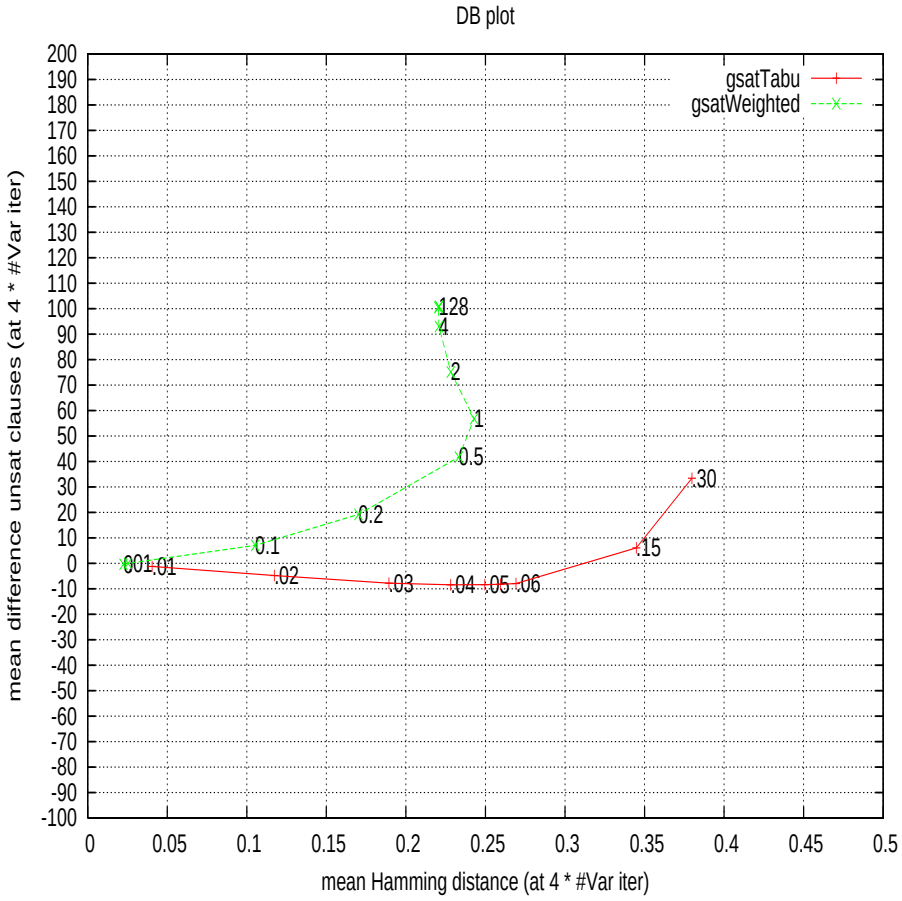

Figure 13: Diversification-bias (DB) plane taking into account the first local minimum after $10 * n$ greedy search steps.

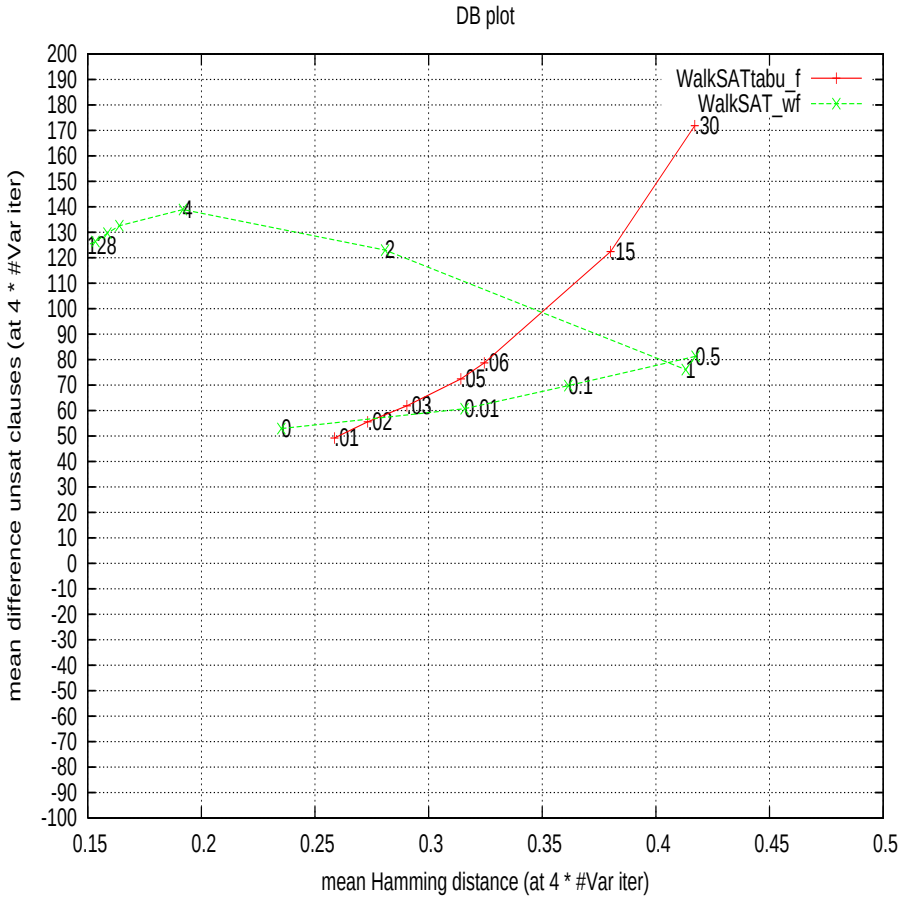

Figure 14: Diversification-bias (DB) plane taking into account the first local minimum after $10 * n$ greedy search steps.

5 Experiments on long runs

For brevity we report here only the average results (10 runs with different random seeds for each of the 50 instances) as a function of the number of iterations (flips). The user of SLS algorithms is typically interested in the number of iterations required by each algorithm to reach the desired results, or, at least, a good quality approximation. As predicted by the previous Diversification-Bias analysis, the curves in Fig. 15 confirm a clear superiority of the prohibition-based techniques with respect to the penalty-based approaches. The error bars are not shown on the plots to avoid cluttering. Among all the possible values for the tabu parameter of the WalkSAT/TABU algorithm, we plot the case where the fractional prohibition T_f is 0.01, as with this setting we obtain the best performance over the considered benchmark. The same for the GSAT/TABU algorithm, whose curve is drawn for the optimal T_f value 0.05 over our benchmark set. With this optimal setting, the GSAT/TABU algorithm reaches eventually a performance equivalent to that of H_RTS, even if its performance is inferior in the initial phase. This result clearly indicates that parameters setting is crucial for the algorithms performance: not only H_RTS reaches the results comparable to the ones with a fixed and optimal T_f, but it actually improves on these because of the dynamic on-line adaptation. This observation is emphasized by the curves for SAPS and RSAPS. They confirm the effectiveness of the reactive approach, that obtains better results while, at same time, allowing to avoid the manual tuning of the optimal parameters setting. The SAPS parameters have been set to the default values, without attempting any extensive optimization. Preliminary tests obtained changing the values did not lead to significant improvements. In detail, the following setting of SAPS parameters is considered:

- the scaling parameter α is 1.3;
- the algorithm smoothing parameter ρ is 0.8;
- the smooth probability p_{smooth} is 0.05;
- the walk probability η is 0.01.

For an explanation of these parameters, see subsection 2.4.

Finally, the curve for H_RTS shows the effectiveness of the NOB search to rapidly discover good local optima.

Fig. 16 shows the behavior of the same algorithms in a scenario closer to the satisfiability threshold (the clauses/variables ratio of the 300:1500 tasks is 5). The results of the previous analysis are confirmed, except, in this case, the competitive performance of AdaptNovelty$^+$ which eventually duplicates H_RTS performance although with a much lower start.

Let us note that many of the considered techniques have been proposed for SAT and one may argue that a direct comparison with H_RTS is not fair. On the other hand, the underlying logic of the methods is always based on maximizing the number of satisfied clauses, which is an argument in favor a

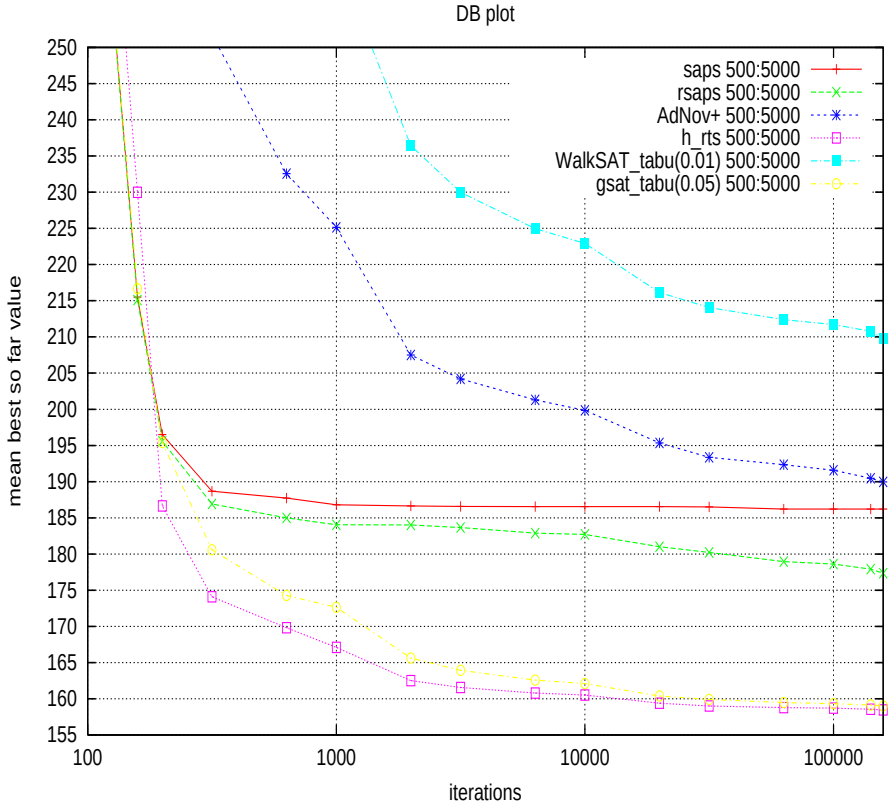

Figure 15: The mean best so far bias value reached by the SAPS, RSAPS, AdaptNovelty$^+$ and H_RTS algorithms.

direct comparison, in particular for adaptive techniques. In any case, this issue will be explored further in the future.

Fig. 17 shows the performance of a simplified version of the H_RTS algorithm, where the prohibition parameter is fixed to the number of flipping candidates causing a null variation for the score function f when the local minimum for the OB search is encountered (line 11 in fig. 7)). I.e., we eliminate the "reaction" (line 16 in fig. 7) to update the prohibition value. Note that this simplified approach obtains performances competitive with the more sophisticated reactive approach.

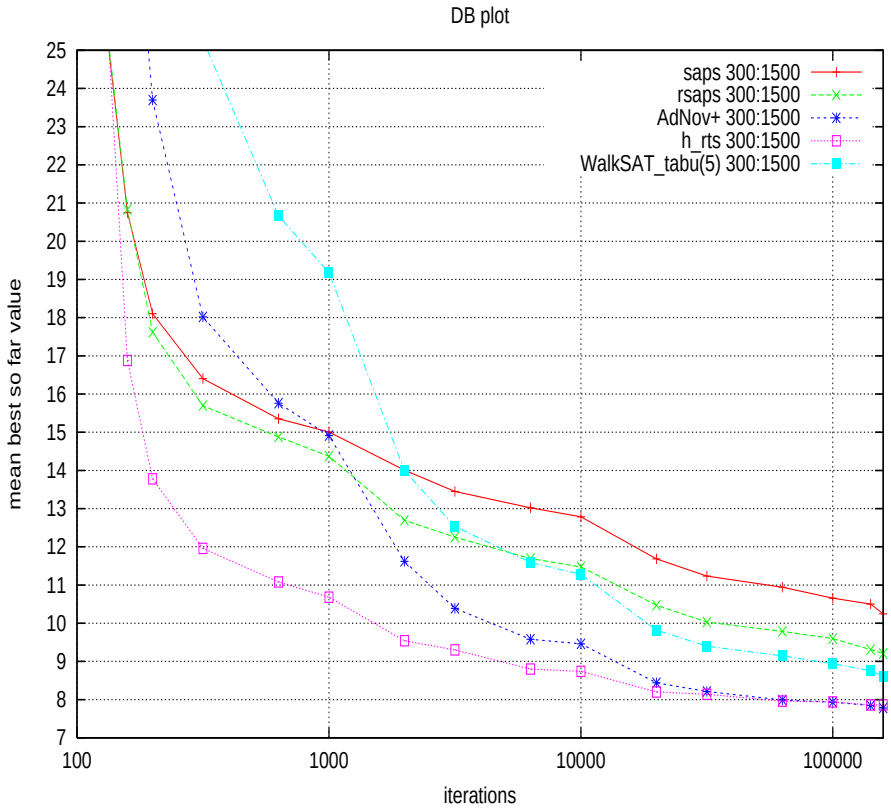

Figure 16: The mean best so far bias value reached by the SAPS, RSAPS, AdaptNovelty$^+$ and H_RTS algorithms on 300:1500 instances.

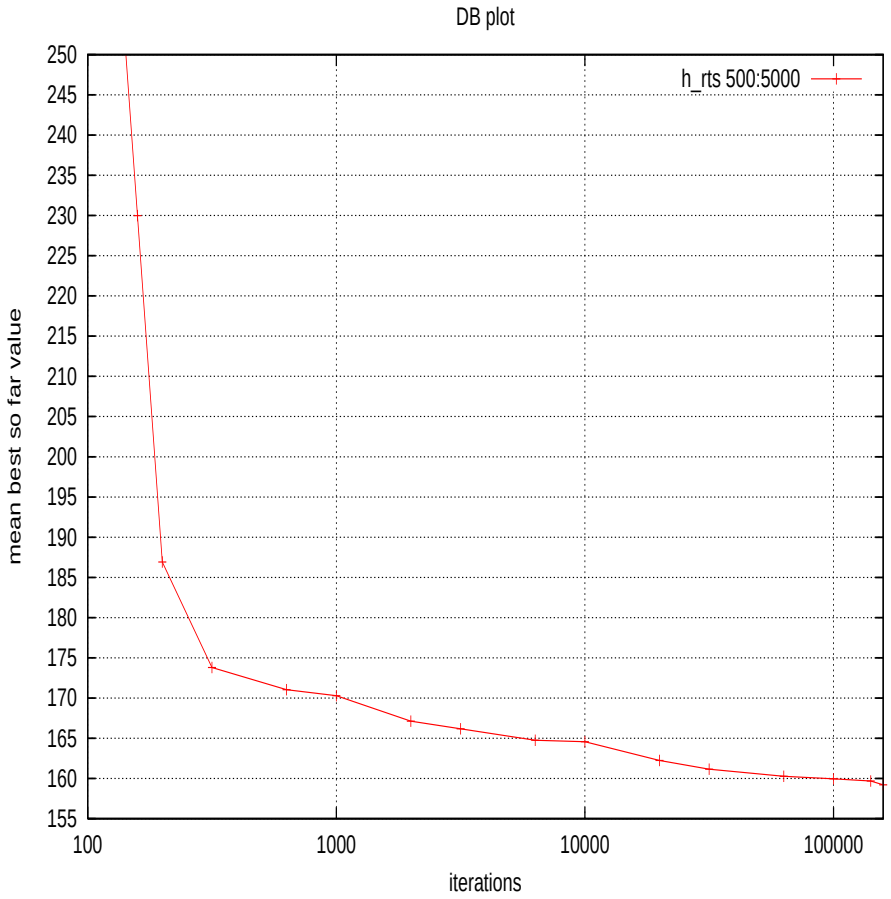

Figure 17: The mean best so far bias value reached by a simplified version of the H_RTS algorithms

.

6 Experiments on CPU times

We finally analyze the CPU time instead of iterations. The comparative CPU time results obtained are fully confirmed, when the competing algorithms are considered in the efficient UBCSAT framework implementation (see section 8). In particular, the experimental results show approximately a factor of two between the time per iteration of GSAT and H_RTS. On our machines (a 2Ghz Intel Xeon processor, with 6GB RAM), the CPU time per iteration ranges approximately from 2.5 microsec per iteration in the case of GSAT to 5 microsec for H_RTS to 7 microsec for SAPS. Table 1 show the CPU times per 100000 iterations (including the set up time) for the considered implementation of the GSAT algorithm, for the SAPS algorithm implemented in the UBCSAT framework and for the H_RTS algorithm. The version denoted by "our" of the GSAT and H_RTS algorithms refers to the original code in [1]. To speed up the execution of the H_RTS algorithm, we implemented an *"ad-hoc"* data structure (Fig. 18), that allows for a fast retrieval of the desired information. In particular, the core of the data structure is an array of lists (the "Delta vector"). The array index takes integer values over the interval $[-m, m]$, where m is the number of clauses of the SAT instance. Each array cell contains a list storing all the variables causing the variation in the number of *satisfied* clauses (Δf) indicated by the cell index value. The "max_index" stores the biggest current value for Δf. Three supporting vectors are exploited for the data structure fast update: the "offset vector", the "position vector" and the"counter vector". The first two allow for a constant time retrieval of a variable: the i-th variable is stored in position *offset vector[i]* of the *position vector [i]* list. The "counter vector" stores the length of the lists. The data structure is updated at each iteration. If the variable v is flipped, only the variable appearing in a clause containing v or its negation may change their location into the data structure. Thanks to the adoption of the supporting vectors, all the operation required to move a variable into the data structure are executed in constant time. The adoption of this data structure allows to save about 20% of the CPU time per iteration (see table 1). Note that the CPU time reported for the optimized version of the H_RTS algorithm refers to an implementation that computes the NOB functions at each iterations. Experimental tests show that a substantial portion of the CPU time per iteration is spent by H_RTS for the NOB functions computation, that allows to derive also the OB function value. If the NOB functions are calculated only when it is strictly necessary (i.e., during the NOB search phase. See lines 5-7 in fig. 7), the amount of time required by H_RTS per iteration is sensibly reduced.

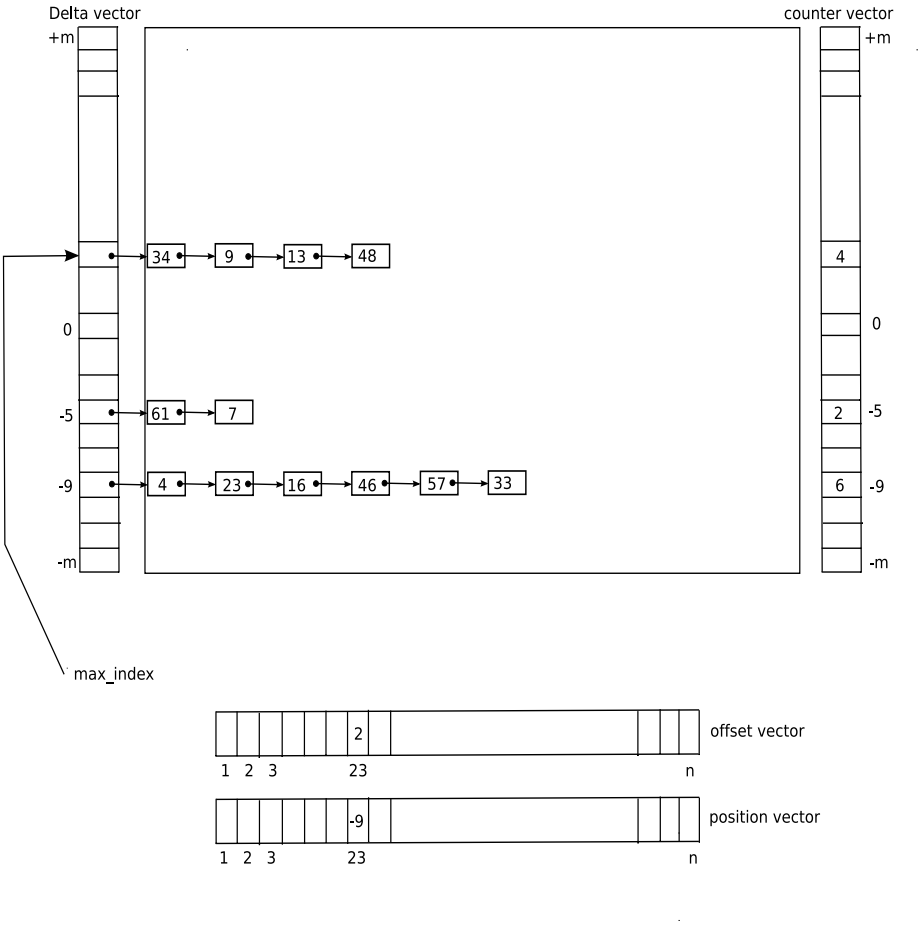

Figure 18: The data structure used by the improved version of the H_RTS algorithm.

Algorithm	version	total CPU time (sec) per 1000000 iterations
GSAT	our	8.5
GSAT	our (optimized)	6.8
GSAT	UBCSAT framework	2.5
SAPS	UBCSAT framework	7
H_RTS	our	12
H_RTS	our (optimized)	10

Table 1: CPU times for 1000000 iterations

7 Conclusion

We presented some selected results of an ongoing comprehensive evaluation of alternatives design strategies for SAT and MAX-SAT algorithms. In particular, we focus onto Diversification-Bias results and onto a comparison on long runs of the Hamming-based Reactive Tabu Search versus competitive approaches proposed in the last years. The D-B plots indicate the complexity and strong non-linearity of diversification-bias plots of penalty-based approaches and the more predictable effects of prohibition-based schemes. Some of these findings agree with [12] which focused onto dynamic local search. The experimental results on long runs confirm that reactive approaches are indeed more effective than static (non-learning) versions, in some cases beating the results which can be obtained by an optimal *off-line* tuning phase.

We are aware that the number of questions raised by this preliminary investigation is bigger that the number of answers provided. We plan to further investigate the raised issues in the next months, in particular by considering more recent proposals like the "adaptive clause weight redistribution" of [13], following the divide-and-distribute-fixed-weight approach in [14], the schemes based aggressive search for local minima proposed in [15] and different schemes like IRoTS [26] or GLS [17], which in any case do not deliver an improving performance. Furthermore we plan to study the relationship between landscapes characteristics [4] and reactive approaches. Finally, in the never-ending empirical evaluation phase, we plan to consider challenging problems defined in the last years, such as the "q-hidden" instances benchmark defined in [8], the hard satisfiable instances generator presented in [9] and the satisfiable spin glass formulas [7] motivated by a spin glass model [19].

8 Acknowledgment

We acknowledge here the colleagues who made the software corresponding to their algorithms available for experimentation. In particular, the software for the GSAT and WalkSAT families algorithms employed and for the SAPS algorithm has been made available by Dave Tompkins and Holger Hoos, at the Department of Computer Science of the University of British Columbia. It can be freely downloaded at http://www.satlib.org/ubcsat.

References

[1] R. Battiti and M. Protasi. Reactive search, a history-sensitive heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2(ARTICLE 2), 1997. http://www.jea.acm.org/.

[2] R. Battiti and M. Protasi. Solving MAX-SAT with non-oblivious functions and history-based heuristics. In D. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, number 35 in DIMACS:

Series in Discrete Mathematics and Theoretical Computer Science, pages 649–667. American Mathematical Society, Association for Computing Machinery, 1997.

[3] Roberto Battiti, Mauro Brunato, and Franco Mascia. *Reactive Search and Intelligent Optimization*. DIT - University of Trento, Via Sommarive 14, 38100, Trento - Italy, April 2007. Available at http://www.reactive-search.org/thebook/.

[4] M. Belaidouni and J.K. Hao. Landscapes of the Maximal Constraint Satisfaction Problem. *Lecture Notes in Computer Science*, 1829:244–255, 2000.

[5] L. Bordeaux, Y. Hamadi, and L. Zhang. Propositional Satisfiability and Constraint Programming: A comparative survey. *ACM Computing Surveys (CSUR)*, 38(4), 2006.

[6] J. A. Boyan and A. W. Moore. Learning evaluation functions for global optimization and boolean satisfability. In AAAI Press, editor, *In Proc. of 15th National Conf. on Artificial Intelligence (AAAI)*, pages 3–10, 1998.

[7] B. Selman H. Jia, C. Moore. From spin glasses to hard satisfiable formulas. In *H.H. Hoos, D.G. Mitchell (Eds.), Theory and Applications of Satisfiability Testing, 7th International Conference (Vancouver, May 10-13, 2004)*, pages 199–210. Springer, 2005.

[8] C. Moore H. Jia and D. Strain. Generating hard satisfiable formulas by hiding solutions deceptively. In *Proceedings of the 20h National Conference on Artificial Intelligence*, pages 384–389. AAAI Press, 2005.

[9] Harri Haanpää, Matti Järvisalo, Petteri Kaski, and Ilkka Niemelä. Hard satisfiable clause sets for benchmarking equivalence reasoning techniques. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):27–46, 2006.

[10] H. H. Hoos and T. Stuetzle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005.

[11] H.H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of the national conference on artificial intelligence*, volume 18, pages 655–660. AAAI Press; MIT Press, 1999.

[12] Holger Hoos and Dave Tompkins. Dynamic local search for sat - clause weights, search landscapes and effective model finding. Unpublished presentation at: Learning and Intelligent OptimizatioN LION 2007 12-18 February 2007, Andalo (Trento), Italy.

[13] A. Ishtaiwi, J. R. Thornton, Sattar A. Anbulagan, and D. N. Pham. Adaptive clause weight redistribution. In *Proceedings of the 12th International Conference on the Principles and Practice of Constraint Programming, CP-2006, Nantes, France*, pages 229–243, 2006.

[14] Abdelraouf Ishtaiwi, John Thornton, Abdul Sattar, and Duc Pham. Neighbourhood clause weight redistribution in local search for sat. In *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 772–776. Springer Berlin / Heidelberg, 2005.

[15] C.M. LI and W.Q. HUANG. Diversification and determinism in local search for satisfiability. In *Proceedings of 8th SAT*, Lecture notes in computer science, pages 158–172. Springer, 2005.

[16] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings of the national conference on artificial intelligence*, number 14, pages 321–326. John Wiley and sons LTD, USA, 1997.

[17] P. Mills and E. Tsang. Guided Local Search for Solving SAT and Weighted MAX-SAT Problems. *Journal of Automated Reasoning*, 24(1):205–223, 2000.

[18] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 459–465, San Jose, Ca, July 1992.

[19] M. E. J. Newman and Cristopher Moore. Glassy dynamics and aging in an exactly solvable spin model. *Physical Review E*, 60:50–68, 1999.

[20] D. Schuurmans, F. Southey, and R.C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proceedings of the international joint conference on artificial intelligence*, volume 17, pages 334–341. Lawrence Erlbaum associates LTD, USA, 2001.

[21] B. Selman and H.A. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proceedings of the eleventh national Conference on Artificial Intelligence (AAAI-93)*, Washington, D. C., 1993.

[22] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the national conference on artificial intelligence*, volume 12. John Wiley and sons LTD, USA, 1994.

[23] B. Selman, H.A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In M. Trick and D. S. Johson, editors, *Proceedings of the Second DIMACS Algorithm Implementation Challenge on Cliques, Coloring and Satisfiability*, number 26 in DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pages 521–531, 1996.

[24] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, Ca, July 1992.

[25] S.Khanna, R.Motwani, M.Sudan, and U.Vazirani. On syntactic versus computational views of approximability. In *Proc. 35th Ann. IEEE Symp. on Foundations of Computer Science*, pages 819–836, 1994.

[26] K. Smyth, H.H. Hoos, and T. Stutzle. Iterated Robust Tabu Search for MAX-SAT. *Proc. of the 16th Canadian Conference on Artificial Intelligence (AI 2003).*

[27] Olaf Steinmann, Antje Strohmaier, and Thomas Stutzle. Tabu search vs. random walk. In *KI - Kunstliche Intelligenz*, pages 337–348, 1997.

[28] Dave A. D. Tompkins and Holger H. Hoos. Novelty$^+$ and adaptive novelty$^+$. SAT 2004 Competition Booklet. (solver description).

[29] F. Hutter D.A.D. Tompkins and H.H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *Proc. Principles and Practice of Constraint Programming - CP 2002 : 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13*, volume 2470 of *LNCS*, pages 233–248. Springer Verlag, 2002.