

MFSAT: A SAT Solver Using Multi-Flip Local Search

Amol Dattatraya Mali & Yevgeny Lipen (*)

Elect. Engg. & Computer Science,

University of Wisconsin, Milwaukee, WI 53211, USA

Phone: 1-414-229-6762, Fax: 1-414-229-2769, mali@uwm.edu

(*) Strong Financial Corporation, 200 Woodland Prime,

Menomonee Falls, WI 53051, Ylipen@strong.com

Abstract

Local search-based methods of SAT solving have received a significant attention in the last decade. All local search-based methods choose the next truth assignment by flipping the value of only one variable. Simultaneously flipping values of multiple variables on an iteration can allow a local search-based solver to take long leaps in the search space. We report on SAT solver MFSAT, which has nine local search-based procedures. One of them is similar to conventional procedures which flip the value of one variable on an iteration. Other eight procedures are capable of simultaneously flipping values of multiple variables on an iteration to generate the successor truth assignment. We call these procedures “multi-flip” procedures. We report on the empirical evaluation of the nine procedures on more than 2700 benchmark SAT problems. Our results show that multi-flip procedures solve many more problems than the uni-flip procedure. Several multi-flip procedures solve problems many times faster than the uni-flip procedure.

1 Introduction

Propositional satisfiability (SAT) is a prototypical combinatorially hard problem which is as expressive as a general constraint satisfaction problem. The importance of SAT in artificial intelligence is clear from several practical applications and from several workshops, international competitions, and special issues of journals devoted to this problem. The practical applications of SAT include circuit design, diagnosis and verification, job scheduling and planning. Several combinatorial problems have been encoded as SAT and solved efficiently. These include map coloring and Steiner tree finding. Classical planning problem can be solved efficiently, when encoded and solved as a SAT problem [4]. The work on classical planning as satisfiability includes [9],[10],[11],[12], [13], and [14].

Local search-based methods have gained a significant attention in the last decade. These are iterative improvement methods. They are incomplete. These methods have solved many SAT and constraint satisfaction problems (CSPs) orders of magnitude faster than the global search-based methods. Local search methods start with an initial assignment to variables and choose assignments that maximize the increase in the number of constraints satisfied or increase the number of satisfied constraints. The methods often choose assignments that do not lead to any improvement either to do simulated annealing or escape from local maximum. The next assignment which improves the value of the score function is generally chosen by examining several candidate assignments. These candidates are obtained by assigning different values to the same variable or assigning different values to multiple variables. In both cases, any candidate assignment differs from the current assignment in the value of only one variable.

Several local search-based methods have been reported in [1], [3], [6], [7], and [8]. All of these generate candidate assignments by changing the value of only one variable. For example, given an n -variable SAT instance, n candidate assignments can be generated, each by flipping the value of one variable. Flipping the values of multiple variables on an iteration to generate the next assignment can be useful for several reasons. Multiple flips can avoid local maximum. Evaluating the effects of multiple flips allows a solver to look ahead. Multiple flips can reduce the number of iterations needed to solve a problem and/or solving time.

The nine local search-based procedures in this paper are P-1A, P-1A-2A, P-1A-2A-T, P-1A-2S, P-1A-3S, P-1A-2S-3S, P-1A-2S-3S-4S-5S, P-1A-2S-3S-4S-5S-T, and P-1A-JWOS. “A” in the names of procedures stands for all and “S” in the names of the procedures stands for some. “T” denotes the use of tabu list. Procedures whose names contain “T” use tabu list. “xA” in the name of a procedure denotes that the procedure examines “all” candidate assignments differing in the values of “x” variables from the cur-

rent assignment, on each iteration. So P-1A examines all assignments differing in the value of 1 variable. P-1A is like many conventional local search-based procedures. We also refer to P-1A as the “uni-flip” procedure. P-1A-2A examines all assignments differing in the value of 1 variable and all assignments differing in the values of two variables on each iteration. “xS” in the name of a procedure denotes that the procedure examines “some” assignments differing in the values of “x” variables on each iteration. For example, P-1A-2S examines all assignments differing in the value of 1 variable and some assignments differing in the values of two variables, on each iteration. P-1A-2S-3S-4S-5S-T examines all assignments differing in the value of one variable, some assignments differing in the values of two variables, some assignments differing in the values of three variables, some assignments differing in the values of four variables, and some assignments differing in the values of five variables, along with the tabu list restriction. Procedure P-1A-JWOS uses ideas from Jeroslow-Wang rule (JW) [2] and occurrences (OS) of variables. All nine procedures do steepest descent since they accept the best possible change from their neighborhood on each iteration, rather than any change that leads to some improvement.

Our work makes the following contributions:

- We report on eight local search-based multi-flip algorithms.
- We report on an empirical evaluation of the nine procedures on 2714 benchmark SAT instances. The empirical evaluation shows that many multi-flip procedures solved many more problems (83 %) than P-1A (44 %).
- The empirical evaluation also shows that many problems are solved faster by multi-flip procedures than P-1A. The evaluation shows that flipping the values of multiple variables on an iteration can be a very good idea.
- We discuss broader usefulness of our work. This includes development of efficient local search-based procedures using dynamic and automatic selection of neighborhood of truth assignments and multi-flip local search procedures for integer linear programming (ILP), MAX-SAT, weighted MAX-SAT and CSPs.

2 Background

Local search style methods have become a viable alternative to constructive backtrack style methods for solving constraint satisfaction and SAT problems [8]. A local search-based algorithm for SAT is reported in [1]. This algorithm generates candidate assignments by flipping the

value of one variable. Several local search algorithms have been reviewed in [7]. These include GSAT, HSAT, WSAT-G, WSAT-B, WSAT, Novelty, and Novelty+. All of these flip the value of only one variable to generate a candidate truth assignment. The random walk (RW), random walk with freebie move (RWF) and Walksat algorithms in [8] flip the value of only one variable on an iteration. The min-conflicts hill climbing algorithm [6] flips the value of one variable on an iteration. The number of satisfied clauses is a common score function in local search. Some local search-based algorithms perform steepest descent. They choose the assignment yielding maximum value of the score function from their neighborhood of assignments. Some local search-based algorithms (e.g. Random Walk, Walksat) select an assignment that yields some improvement in the value of the score function. Algorithms like Walksat some times select assignments that do not yield any improvement, even though improving assignments exist.

3 Multi-flip Procedures

In this section we first explain the notation relevant to the multi-flip procedures. Then we explain how various neighbors are found. After this we explain the algorithms of P-1A and the eight multi-flip procedures.

3.1 Notation

We represent the truth assignment to various variables assuming a fixed total order over the variables. So the assignment $x_1 = T, x_2 = F, x_3 = T$ is represented by $\langle T, F, T \rangle$, assuming that the total order over the variables is $x_1 \prec x_2 \prec x_3$. N_{1a, A_i} denotes the list of all truth assignments differing in the value of one variable, as compared with the truth assignment A_i . For example, if $A_1 = \langle T, T, T \rangle$, then N_{1a, A_1} is $[\langle F, T, T \rangle, \langle T, F, T \rangle, \langle T, T, F \rangle]$. If $A_2 = \langle F, T, F, F \rangle$, then N_{1a, A_2} is $[\langle T, T, F, F \rangle, \langle F, F, F, F \rangle, \langle F, T, T, F \rangle, \langle F, T, F, T \rangle]$. If A_i is an assignment to n variables, then length of $N_{1a, A_i} = n$. $B(N_{1a, A_i})$ is the first assignment in N_{1a, A_i} that satisfies maximum number of clauses from the given SAT instance, among all assignments in N_{1a, A_i} . In general, $B(S)$ is the first assignment from list S which satisfies maximum number of clauses among all assignments from S .

N_{2a, A_i} denotes the list of all assignments differing in the values of 2 variables when compared with the assignment A_i . For example, if $A_1 = \langle T, F, T \rangle$, then N_{2a, A_1} is $[\langle F, T, T \rangle, \langle T, T, F \rangle, \langle F, F, F \rangle]$. If $A_2 = \langle T, F, F, T \rangle$, then N_{2a, A_2} is $[\langle F, T, F, T \rangle, \langle F, F, T, T \rangle, \langle F, F, F, F \rangle, \langle T, T, T, T \rangle, \langle T, T, F, F \rangle, \langle T, F, T, F \rangle]$. In general, length of N_{2a, A_i} is $\frac{n \cdot (n-1)}{2}$ when A_i contains an assignment to n variables.

$B(N_{2a,A_i})$ is the first assignment from N_{2a,A_i} which satisfies maximum number of clauses.

N_{2,s,A_i} denotes the list of first s assignments differing in the values of two variables, when compared with the assignment A_i , where $s \leq \text{length}(N_{2a,A_i})$. So $s \leq \frac{n \cdot (n-1)}{2}$. $\frac{n \cdot (n-1)}{2}$ is $C(n, 2)$. If a value higher than $\frac{n \cdot (n-1)}{2}$ is chosen, s is considered to be equal to $\frac{n \cdot (n-1)}{2}$ by MFSAT solver. For a given A_i and s , N_{2,s,A_i} is unique. For example, if $s = 1$ and $A_1 = \langle T, T, F \rangle$, then $N_{2,s,A_1} = \langle \langle F, F, F \rangle \rangle$. If $s = 2$, then $N_{2,s,A_1} = \langle \langle F, F, F \rangle, \langle F, F, T \rangle \rangle$. Let $A_2 = \langle v_1, v_2, v_3, v_4 \rangle$, where v_1, v_2, v_3 and v_4 are values of 4 variables x_1, x_2, x_3 and x_4 respectively. Then N_{2,s,A_2} contains first s elements of the following list: $\langle \neg v_1, \neg v_2, v_3, v_4 \rangle, \langle \neg v_1, v_2, \neg v_3, v_4 \rangle, \langle \neg v_1, v_2, v_3, \neg v_4 \rangle, \langle v_1, \neg v_2, \neg v_3, v_4 \rangle, \langle v_1, \neg v_2, v_3, \neg v_4 \rangle, \langle v_1, v_2, \neg v_3, \neg v_4 \rangle$. If $v_i = T$, then $\neg v_i = F$. If $v_i = F$, then $\neg v_i = T$. The list contains $n - 1$ assignments obtained by flipping v_1 and every other value. The list contains $n - 2$ assignments obtained by flipping v_2 and every value $v_j, j > 2$. In general the list contains $n - i$ assignments obtained by flipping v_i and every value $v_k, k > i$, such that $1 \leq i \leq (n - 1)$. $B(N_{2,s,A_i})$ is the first assignment from N_{2,s,A_i} which satisfies maximum number of clauses.

N_{3,s,A_i} is the list of first s assignments differing in the values of 3 variables as compared with the assignment A_i , where $s \leq \frac{(n-1) \cdot (n-2)}{2}$. These are first s assignments from the list which has $n - 2$ truth assignments obtained by flipping v_1, v_2 and every other value. The list has $n - 3$ assignments obtained by flipping v_2, v_3 and every value $v_j, j > 3$. In general, the list has $n - i - 1$ assignments obtained by flipping v_i, v_{i+1} , and every value $v_j, j > (i + 1)$, such that $1 \leq i \leq (n - 2)$. Note that N_{3,s,A_1} here does not contain $\langle \neg v_1, v_2, v_3, \neg v_4, \neg v_5 \rangle, \langle \neg v_1, v_2, \neg v_3, v_4, \neg v_5 \rangle$ etc. If such assignments are included, the maximum length of N_{3,s,A_i} is $\frac{n \cdot (n-1) \cdot (n-2)}{6}$ which is very high for high values of n . If s higher than $\frac{(n-1) \cdot (n-2)}{2}$ is specified, MFSAT solver uses s equal to $\frac{(n-1) \cdot (n-2)}{2}$. N_{3,s,A_i} is unique for a given s and A_i .

N_{4,s,A_i} denotes the list of first s assignments from a list of truth assignments which differ in the values of 4 variables as compared with the assignment A_i . The list has $n - i - 2$ assignments obtained by flipping v_i, v_{i+1}, v_{i+2} and every value $v_j, j > (i + 2)$, such that $1 \leq i \leq (n - 3)$. So the maximum length of N_{4,s,A_i} is $\frac{(n-3) \cdot (n-2)}{2}$. If the value of s chosen is greater than $\frac{(n-3) \cdot (n-2)}{2}$, then s is assumed to be equal to $\frac{(n-3) \cdot (n-2)}{2}$ by MFSAT solver.

N_{5,s,A_i} is the list of first s assignments from a list of truth assignments differing in the values of 5 variables, as compared with the assignment A_i , where $s \leq \frac{(n-3) \cdot (n-4)}{2}$. Thus the maximum length of N_{5,s,A_i} is $\frac{(n-3) \cdot (n-4)}{2}$ and

not $\frac{n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot (n-4)}{120}$. $\frac{n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot (n-4)}{120}$ is same as $C(n, 5)$. If the chosen value of s is greater than $\frac{(n-3) \cdot (n-4)}{2}$, then $\frac{(n-3) \cdot (n-4)}{2}$ is used by MFSAT. N_{5,s,A_i} is computed in a manner similar to $N_{1,s,A_i}, N_{2,s,A_i}, N_{3,s,A_i}$, and N_{4,s,A_i} . N_{5,s,A_i} is unique for a given s and A_i . The list from which elements of N_{5,s,A_i} are chosen has $n - i - 3$ assignments obtained by flipping $v_i, v_{i+1}, v_{i+2}, v_{i+3}$ and every other value $v_j, j > (i + 3)$, such that $1 \leq i \leq (n - 4)$.

3.2 Local Search-based Algorithms

We describe the algorithms used by the 9 local search-based procedures in this subsection. Search is considered to be stuck in local maximum if the number of clauses satisfied by all alternative assignments (all neighbors) is less than or equal to the number of clauses satisfied by the current assignment.

The initial truth assignment is generated by all the 9 procedures in the following manner. To assign initial truth value to each variable, a random number k is generated and $(k \bmod 2)$ is found. If the resulting number is 0, the variable is assigned true, otherwise it is assigned false. So generation of the initial truth assignment to n variables involves generation of n random numbers.

The neighboring truth assignments are generated by each procedure in a fixed order. If multiple truth assignments satisfy maximum number of clauses which are also more than the ones satisfied by the current assignment, then the assignment generated first is chosen from these. If tabu list is used, the the earliest generated assignment which is not in the tabu list and which also satisfies maximum and more clauses than the current assignment, is selected as the next assignment.

To escape from a local maximum, each procedure generates a random positive integer p which is less than the number of variables and then flips the assignments of p variables $1, 2, 3, \dots, p$. In DIMACS format, variables are denoted by positive integers.

Algorithm for P-1A

1. Generate a random initial truth assignment for all variables. Let this be A .
2. For $\{i = 1 \text{ to } \text{max_iterations}\}$
 - { If A satisfies all clauses, then exit.
- If there is an assignment in $N_{1a,A}$ which satisfies more clauses than A , then $A \leftarrow B(N_{1a,A})$, else
- $A \leftarrow$ A randomly generated assignment for escaping from local maximum }
3. Print "No solution found".

Algorithm for P-1A-2A

It is same as that of P-1A, with the following difference. Instead of $N_{1a,A}$, we have the list obtained by concatenating $N_{1a,A}$ and $N_{2a,A}$ everywhere in the procedure for P-1A-2A.

Algorithm for P-1A-2A-T

It is same as that of P-1A-2A, with the following difference. A move (flipping of values of variables) is not made if it is in the tabu list. Tabu list serves as a simple form of short term memory managed by first in first out (FIFO) rule. If the best move is in the tabu list, next best move is used if it satisfies more clauses than A . If all moves satisfying more clauses than A are in the tabu list, a random assignment for escaping from local maximum is generated.

Algorithm for P-1A-2S

The algorithm for P-1A-2S is same as that for P-1A with the following difference. Instead of $N_{1a,A}$, we have the list obtained by concatenating $N_{1a,A}$ and $N_{2,s,A}$ everywhere in the procedure for P-1A-2S.

Algorithm for P-1A-3S

The algorithm for P-1A-3S is same as that for P-1A with the following difference. Instead of $N_{1a,A}$, we have the list obtained by concatenating $N_{1a,A}$ and $N_{3,s,A}$ everywhere in the procedure for P-1A-3S.

Algorithm for P-1A-2S-3S

The algorithm for P-1A-2S-3S is same as that for P-1A with the following difference. Instead of $N_{1a,A}$, we have the concatenation of the three lists $N_{1a,A}$, $N_{2,s,A}$ and $N_{3,s',A}$, everywhere in the procedure for P-1A-2S-3S.

Algorithm for P-1A-2S-3S-4S-5S

The algorithm for P-1A-2S-3S-4S-5S is same as that for P-1A with the following difference. Instead of $N_{1a,A}$, we have the concatenation of the five lists $N_{1a,A}$, $N_{2,s,A}$, $N_{3,s',A}$, $N_{4,s'',A}$ and $N_{5,s''',A}$, everywhere in the procedure for P-1A-2S-3S-4S-5S.

Algorithm for P-1A-2S-3S-4S-5S-T

This is same as that for P-1A-2S-3S-4S-5S with the following difference. P-1A-2S-3S-4S-5S-T uses tabu list. If the best move is in the tabu list, next best move is chosen if it leads to satisfaction of more clauses than the current assignment A . If all moves that lead to satisfaction of more clauses than A are in the tabu list, a random assignment to escape from local maximum is chosen.

Algorithm for P-1A-JWOS

This procedure uses the number of positive and negative occurrences of a variable and ideas from Jeroslow-Wang rule [2] in deciding which move to make. Consider the SAT instance $(x \vee y \vee \neg z) \wedge (z \vee \neg y)$. x, y, z all have 1 positive occurrence in the instance. x, y, z respectively have 0,1,1 negative occurrences in the instance. $D(v)$ denotes the absolute difference between the number of positive occurrences of v (p_v) and the number of negative occurrences of v (n_v). In case of the example instance, $D(x) = 1, D(y) = D(z) = 0$. $J(L)$ where L is a literal in the SAT instance is $\sum 2^{-n_i}$ found over all clauses containing L , n_i denoting the length of i th such a clause. In case of the example, $J(x) = 2^{-3} = 0.125$, $J(\neg x) = 0$, $J(y) = J(\neg z) = 2^{-3} = 0.125$, $J(z) = J(\neg y) = 2^{-2} =$

0.25. If the clause $(\neg y \vee p \vee q \vee r)$ is added to the instance, $J(\neg y) = 2^{-2} + 2^{-4} = 0.3125$.

P-1A-JWOS computes $D(v)$ values of all variables and $J(L)$ values of all literals in the given SAT instance. It ranks variables in the decreasing order of the $D(v)$ values. It ranks literals in the decreasing order of the $J(L)$ values. P-1A-JWOS uses user specified values n_6 and n_7 , $2 \leq n_6 \leq n, 3 \leq n_7 \leq n$. T_D denotes the set of first n_6 variables with high $D(v)$ values from the $D(v)$ based order. T_{JW} denotes the set of first n_7 literals with high $J(L)$ values from the $J(L)$ based order.

P-1A-JWOS

1. Generate a random initial truth assignment for all variables. Let this be A .
2. For $\{i = 1 \text{ to } \text{max_iterations}\}$
{ If A satisfies all clauses, then exit.
(a) Find an assignment in $N_{1a,A}$ which satisfies maximum number of clauses. Let this be A_1 .
(b) Find an assignment by flipping values of 2 randomly chosen variables from T_D . Let this be A_2 .
(c) Find an assignment by flipping values of 2 randomly chosen variables from T_{JW} . Let this be A_3 .
(d) If $B([A_1, A_2, A_3])$ satisfies more clauses than A , then $A \leftarrow B([A_1, A_2, A_3])$ else
 $A \leftarrow A$ randomly generated assignment to escape from local maximum }.
3. Print "No solution found".

4 Empirical Evaluation

The MFSAT solver containing the nine local search-based procedures is implemented in C++. When run, the solver first checks whether all files with .cnf extension in its directory are in DIMACS input format. If some files are not in the correct input format, it does not proceed. The solver allows a user to select the procedures to be run. The solver asks a user to choose the maximum number of iterations, length of tabu list and the number of neighboring truth assignments to be examined by various procedures. MFSAT attempts all 9 procedures in a fixed sequence on all problems in its directory, when a user chooses to run all procedures. The solver outputs the number of times each procedure got stuck in local maximum on each problem, the number of iterations used, the cpu seconds and the maximum number of clauses it could satisfy. The maximum number of clauses satisfied is the maximum of the clauses satisfied on all iterations. The empirical evaluation was conducted on a grid with Solaris 8, dual UltraSparc 3/750, 1.5 GB RAM, and 40 GB fiber channel local drive.

Results from the empirical evaluation are shown in tables 1,2,3,4,5,6,7,8, and 9. A summary based on these results is shown in Table 10. S/A in tables 1,...,9 denotes

the number of problems solved and the number of problems attempted. SOB in tables 1,...,6 denotes the number of problems solved only by the specific procedure (and not solved by any other procedure which attempted the problems). For example, table 2 shows that out of 800 problems attempted by all procedures, 53 were solved only by P-1A-2A. F in tables 1,...,5 shows the number of problems solved by a procedure fastest, when two or more procedures solved the problem. I in tables 1,...,5 denotes the number of problems solved by two or more procedures that a specific procedure solved in fewest iterations. For example, table 5 shows that 12 of the problems solved by P-1A-2A were also solved by some other procedure/procedures, such that P-1A-2A solved these in fewest iterations. FM in tables 7,8 and 9 shows the number of problems that the specific multi-flip procedure solved faster than P-1A. FU in tables 7,8 and 9 shows the number of problems that P-1A (uni-flip procedure) solved faster than the specific multi-flip procedure. For example, table 7 shows that P-1A-3S solved 152 problems faster than P-1A and that P-1A solved 217 problems faster than P-1A-3S. MF in table 10 denotes the total number of problems containing the specified number of variables that one or more multi-flip procedures solved. OMF in table 10 denotes the total number of problems containing the specified number of variables that only one or more multi-flip procedures solved. * in table 5 and - in tables 7,9 and 8 denote "Not Applicable". n_2 denotes the number of neighbors differing in the values of two variables. n_3 denotes the number of neighbors differing in the values of three variables. n_4 denotes the number of neighbors differing in the values of four variables. n_5 denotes the number of neighbors differing in the values of five variables. n_2 is relevant to P-1A-2S, P-1A-2S-3S, P-1A-2S-3S-4S-5S, and P-1A-2S-3S-4S-5S-T. n_3 is relevant to P-1A-3S, P-1A-2S-3S, P-1A-2S-3S-4S-5S, and P-1A-2S-3S-4S-5S-T. n_4 is relevant to P-1A-2S-3S-4S-5S, and P-1A-2S-3S-4S-5S-T. n_5 is relevant to P-1A-2S-3S-4S-5S, and P-1A-2S-3S-4S-5S-T. n_6 denotes the number of variables with high $D(v)$ values. n_7 denotes the number of literals with high $J()$ values. n_6 and n_7 are relevant to P-1A-JWOS.

Every problem was attempted with P-1A and at least one multi-flip procedure. Since the primary aim of this work was a comparison of multi-flip procedures and P-1A, not every multi-flip procedure was attempted on each problem. For each procedure, we found the number of problems solved by it. We also found the number of problems solved by a procedure that no other procedure did (SOB). We also found the number of solved problems on which a procedure was fastest, based on problems that were solved by 2 or more procedures. A procedure P_i was considered to be fastest on a problem X if the solving time needed by P_i was not higher than the solving time of any other procedure that solved X . So two or more procedures may be fastest on the

same problem. We found the total solving time for procedure based on the problems solved by it. We also found the number of problems solved by a procedure in fewest iterations, based on problems solved by 2 or more procedures. A procedure P_i was considered as needing fewest iterations on a problem X if the number of iterations needed by P_i was not higher than the number of iterations needed by any other procedure that solved X . We found the total number of iterations needed by each procedure based on the problems solved by it. We also found the total number of problems solved by some procedure other than P-1A (MF). This information is reported in Table 10. We also found the total number of problems solved only by some multi-flip procedure. This information (OMF) is also reported in Table 10.

The empirical results on 100 20-variable random 3-SAT satisfiable problems are shown in Table 1. All these problems have 91 clauses. These are first 100 problems from the uf20-91.tar.gz collection at <http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/>. These are hard problems from the phase transition region, with the ratio of the number of clauses and the number of variables equal to 4.55. The length of the tabu list was 5. $n_i, i \in [2, 7]$ were all equal to 10. The maximum number of iterations was 200. P-1A-2A-T solved all 100 problems. 7 out of the 8 multi-flip procedures solved more problems than P-1A. 7 multi-flip procedures needed fewer total iterations than P-1A. P-1A-JWOS needed lower total solving time than P-1A and also lower average solving time than P-1A. Two multi-flip procedures (P-1A-3S and P-1A-JWOS) were fastest on more problems than P-1A. These two procedures also solved more problems than P-1A. Every multi-flip procedure solved some problems that P-1A did not. P-1A-2A and P-1A-2A-T were best at this, both solving 20 problems that P-1A did not. MFSAT attempted remaining 900 problems from the same collection. All procedures used upto 200 iterations, tabu list length 5 and n_i values equal to 10. The results of this run are shown in Table 7. The number of problems solved by one or multi-flip procedures was 900. So each problem was solved by some multi-flip procedure. 273 out of these 900 problems were solved faster than P-1A by one or more multi-flip procedures. Only 5 out of the 900 problems were solved by P-1A faster than all 8 multi-flip procedures. Instead of SOB, F and I values, we found FM and FU values which reveal different and useful information.

Results on 800 50-variable satisfiable random 3-SAT problems from the uf50-218.tar.gz collection at SATLIB are shown in Table 2. The ratio of the number of clauses and the number of variables for problems in this collection is 4.36. These are hard problems. They belong to the phase transition region. The length of tabu list was 5. The limit on the number of iterations was 1000. All $n_i, i \in [2, 7]$ were equal to 10. 7 multi-flip procedures solved more prob-

Procedure	S/A	SOB	F	I
P-1A	80 / 100	0	37	4
P-1A-2A	98 / 100	0	23	35
P-1A-2A-T	100 / 100	0	20	29
P-1A-2S	87 / 100	0	31	7
P-1A-3S	89 / 100	0	40	5
P-1A-2S-3S	80 / 100	0	24	6
P-1A-2S-3S-4S-5S	85 / 100	0	18	6
P-1A-2S-3S-4S-5S-T	86 / 100	0	24	10
P-1A-JWOS	84 / 100	0	49	2

Table 1. Results on 20-variable random 3-SAT problems

lems than P-1A. 53 problems were solved only by P-1A-2A. P-1A-JWOS was fastest on more solved problems than all other procedures. At least 7 multi-flip procedures each solved more than 100 problems that P-1A did not. P-1A-2A solved 462 problems that P-1A did not. We also evaluated the 9 procedures on the remaining 200 problems from the uf50-218.tar.gz collection of 1000 problems.

Procedure	S/A	SOB	F	I
P-1A	195 / 800	2	78	18
P-1A-2A	647 / 800	53	69	237
P-1A-2A-T	623 / 800	43	70	243
P-1A-2S	200 / 800	2	75	19
P-1A-3S	188 / 800	1	77	22
P-1A-2S-3S	199 / 800	3	58	18
P-1A-2S-3S-4S-5S	223 / 800	2	69	23
P-1A-2S-3S-4S-5S-T	218 / 800	2	56	23
P-1A-JWOS	221 / 800	3	100	26

Table 2. Results on hard 50-variable random 3-SAT problems

The results on 60-variable problems are shown in Table 3. These are 3-SAT problems with 258 clauses from dejong60cnf.tar.gz collection at <http://www.in.tu-clausthal.de/~gottlieb/benchmarks/sat>. These are hard problems from the phase transition region with the ratio of the number of clauses and the number of variables equal to 4.3. The length of tabu list was 5, all six n_i values were equal to 10 and the limit on the number of iterations was 1100. Three multi-flip procedures solved more problems than P-1A.

The results on 75-variable problems are reported in Table 4. These are hard random 3-SAT satisfiable problems with 325 clauses from the uf75-325.tar.gz collection at SATLIB.

Procedure	S/A	SOB	F	I
P-1A	4 / 40	0	3	2
P-1A-2A	19 / 40	5	3	6
P-1A-2A-T	19 / 40	4	2	8
P-1A-2S	3 / 40	0	1	0
P-1A-3S	2 / 40	0	1	1
P-1A-2S-3S	5 / 40	0	1	0
P-1A-2S-3S-4S-5S	3 / 40	0	3	0
P-1A-2S-3S-4S-5S-T	2 / 40	0	1	0
P-1A-JWOS	4 / 40	0	2	0

Table 3. Results on hard 60-variable random 3-SAT problems

The problems lie in the phase transition region. The limit on the number of iterations was 1000 in the attempts on all problems. The length of tabu list was 10 in the case of attempts on 87 problems and 5 in case of attempts on 12 problems. The $n_i, i \in [2, 7]$ values were equal to 10 in case of attempts on 22 problems and equal to 20 in case of attempts on 77 problems. Five out of eight multi-flip procedures solved more problems than P-1A. 13 problems were solved only by P-1A-2A. Six multi-flip procedures were fastest on more problems than P-1A. P-1A-2A solved 31 problems that P-1A did not.

Procedure	S/A	SOB	F	I
P-1A	1 / 99	0	0	0
P-1A-2A	32 / 99	13	7	8
P-1A-2A-T	31 / 99	10	8	11
P-1A-2S	2 / 99	2	2	2
P-1A-3S	0 / 99	0	0	0
P-1A-2S-3S	5 / 99	1	3	0
P-1A-2S-3S-4S-5S	3 / 99	0	2	1
P-1A-2S-3S-4S-5S-T	1 / 99	1	0	0
P-1A-JWOS	1 / 99	1	1	0

Table 4. Results on hard 75-variable random 3-SAT problems

The results on 90-variable satisfiable problems from 3 colorable flat graph coloring set collection flat30-60.tar.gz at SATLIB are shown in Table 5. The problems have 300 clauses. The SAT instances are encodings of problems with 30 vertices and 60 edges. The limit on the number of iterations was 1000 and all n_i values were equal to 20. Two multi-flip procedures solved more problems than P-1A. P-1A-2S was fastest on more solved problems than P-1A.

The results on 100-variable problems are shown in Table

Procedure	S/A	SOB	F	I
P-1A	8 / 100	1	5	2
P-1A-2A	86 / 100	66	0	12
P-1A-2A-T	0 / 0	*	*	*
P-1A-2S	11 / 71	1	7	5
P-1A-3S	3 / 40	0	0	0
P-1A-2S-3S	5 / 40	0	2	1
P-1A-2S-3S-4S-5S	5 / 40	0	4	1
P-1A-2S-3S-4S-5S-T	0 / 0	*	*	*
P-1A-JWOS	3 / 40	0	3	1

Table 5. Results on 90-variable graph coloring problems

6. These are problems from the CBS_k3_n100_m403_b10 collection at SATLIB. These are hard satisfiable random 3-SAT problems with 403 clauses and 10 backbones. The hardness of SAT instances is correlated with the number of backbones. A backbone is a variable that must be assigned only 1 value. For example, x and y are backbones in the SAT instance $x \wedge \neg y \wedge (p \vee q)$. m is a backbone in the SAT instance $(m \vee n) \wedge (m \vee \neg n)$. 10 problems were attempted only with P-1A, P-1A-2A and P-1A-2A-T with 2000 as the limit on the number of iterations and 5 as the length of tabu list. 46 problems were attempted only with P-1A and P-1A-2A, with 1500 as the limit on the number of iterations. 44 problems were attempted only with P-1A and P-1A-2A, with 1700 as the limit on the number of iterations. 14 problems were attempted only with P-1A, P-1A-2S, P-1A-3S, P-1A-2S-3S, P-1A-2S-3S-4S-5S, P-1A-2S-3S-4S-5S-T, and P-1A-JWOS, with $n_i, i \in [2, 7]$ values equal to 20, tabu list's length equal to 5 and 5000 as the limit on the number of iterations. 26 problems were attempted only with P-1A, P-1A-2A, P-1A-2S, P-1A-3S, P-1A-2S-3S, P-1A-2S-3S-4S-5S, P-1A-2S-3S-4S-5S-T, and P-1A-JWOS, with 1500 as the limit on the number of iterations and $n_i, i \in [2, 7]$ values equal to 20. 45 problems were attempted only with P-1A, P-1A-2A, with 1200 as the limit on the number of iterations. 25 problems were attempted only with P-1A, P-1A-2S, P-1A-3S, P-1A-2S-3S, P-1A-2S-3S-4S-5S and P-1A-JWOS, with all n_i values equal to 20 and 1200 as the limit on the number of iterations. P-1A-2A solved maximum number of problems (34). Five multi-flip procedures solved more problems than P-1A. 33 problems were solved only by P-1A-2A. We also attempted 60 more problems from the 403 clause, 10 backbones collection with P-1A, P-1A-JWOS and P-1A-2A. P-1A solved only 1 problem and P-1A-2A solved 16 problems. All these procedures used upto 2000 iterations on these 60 problems. Values of n_6 and n_7 that P-1A-JWOS used were 20 and 10 respectively.

Procedure	S/A	SOB
P-1A	0 / 210	0
P-1A-2A	34 / 171	33
P-1A-2A-T	2 / 10	1
P-1A-2S	0 / 65	0
P-1A-3S	1 / 65	1
P-1A-2S-3S	0 / 65	0
P-1A-2S-3S-4S-5S	1 / 65	1
P-1A-2S-3S-4S-5S-T	0 / 40	0
P-1A-JWOS	1 / 65	1

Table 6. Results on hard 100-variable random 3-SAT problems

We also used 150-variable satisfiable 3 colorable flat graph coloring problems from the flat50-115.tar.gz collection at SATLIB. The problems have 545 clauses. The problems are encodings of graph coloring problems with 50 vertices and 115 edges. 10 problems were attempted only with P-1A-2A and P-1A, with the number of iterations limited to 1000. 21 problems were attempted only with P-1A, P-1A-2S, P-1A-3S, P-1A-2S-3S, P-1A-2S-3S-4S-5S, P-1A-2S-3S-4S-5S-T, and P-1A-JWOS, with all n_i values equal to 20 and the number of iterations limited to 1200. P-1A-2A was the most effective procedure on these problems. P-1A did not solve any problem. P-1A-2A solved 4 out of the 10 problems.

Procedure	S/A	FM	FU
P-1A	775 / 900	-	-
P-1A-2A	887 / 900	128	405
P-1A-2A-T	893 / 900	120	367
P-1A-2S	783 / 900	136	208
P-1A-3S	778 / 900	152	217
P-1A-2S-3S	797 / 900	115	247
P-1A-2S-3S-4S-5S	799 / 900	108	338
P-1A-2S-3S-4S-5S-T	800 / 900	107	346
P-1A-JWOS	798 / 900	158	153

Table 7. Results on additional 20-variable random 3-SAT problems

Multi-flip procedures are able to examine more truth assignments than the uni-flip procedure in the same number of iterations. So one can compare the performances of multi-flip procedures using upto k iterations with the performance of the uni-flip procedure using upto k' iterations, where $k' \gg k$. The uni-flip procedure may solve more problems if it is allowed to use more iterations than the

multi-flip procedure. To test this, we evaluated the procedures on several hard satisfiable 40-variable and 80-variable 3-SAT problems. The evaluation showed that multi-flip procedures continued to have advantages over the uni-flip procedure.

Results on 97 satisfiable 40-variable 3-SAT problems from the `dejong40cnf.tar.gz` collection are shown in Table 8. P-1A was allowed to use 10,000 iterations and the multi-flip procedures were all allowed only 1000 iterations. The length of the tabu list was 5 and $n_7 = n_2 = n_3 = n_4 = n_5 = n_6 = 10$. The number of problems that 3 or more multi-procedures solved was 63 (out of 97). The number of problems that were solved by 4 or more multi-flip procedures was 52 (out of 97). The number of problems solved by 5 or more multi-flip procedures was 42 (out of 97). This shows that given a problem, there is a good chance that some multi-flip procedure will solve it. Number of problems that two or more multi-flip procedures solved faster than P-1A was 52. Number of problems that three or more multi-flip procedures solved faster than P-1A was 42. Number of problems that four or more multi-flip procedures solved faster than P-1A was 27. This shows that given a problem, there is a good chance that some multi-flip procedure will solve it faster than P-1A. 76 out of the 97 problems were solved faster than P-1A by one or more multi-flip procedures. P-1A solved 12 out of the 97 problems faster than all multi-flip procedures. 90 out of the 97 problems were solved by one or more multi-flip procedures. Total number of problems solved by one or more multi-flip procedures excluding P-1A-2A was 83.

Procedure	S/A	FM	FU
P-1A	82 / 97	-	-
P-1A-2A	83 / 97	29	53
P-1A-2A-T	0 / 0	-	-
P-1A-2S	47 / 97	28	30
P-1A-3S	52 / 97	42	19
P-1A-2S-3S	44 / 97	29	29
P-1A-2S-3S-4S-5S	47 / 97	27	41
P-1A-2S-3S-4S-5S-T	50 / 97	21	47
P-1A-JWOS	51 / 97	43	23

Table 8. Results on hard 40-variable 3-SAT problems

The results on 80-variable hard 3-SAT problems are shown in the Table 9. These are problems from the `dejong80cnf.tar.gz` collection. The uni-flip procedure was allowed 20,000 iterations on each of these problems. The multi-flip procedures were allowed between 2000 and 2500 iterations only. On 18 problems, the multi-flip procedures were allowed 2000 iterations, with all n_i values equal to 20

and tabu list of length 10. On 57 problems, they were allowed 2500 iterations, with all n_i values equal to 10 and tabu list of length 10. This evaluation shows that allowing many more iterations to uni-flip procedure did not improve its performance. It solved only 6 out of 76 problems and the multi-flip procedures solved 22 out of the 75 problems they attempted. 8 out of the 75 problems were solved by one or more multi-flip procedures faster than P-1A. P-1A did not solve any problem faster than all multi-flip procedures. So multi-flip procedures continue to have superior performance.

Procedure	S/A	FM	FU
P-1A	6 / 76	-	-
P-1A-2A	22 / 75	5	5
P-1A-2A-T	0 / 0	-	-
P-1A-2S	2 / 75	0	2
P-1A-3S	2 / 75	1	1
P-1A-2S-3S	3 / 75	2	1
P-1A-2S-3S-4S-5S	0 / 75	0	2
P-1A-2S-3S-4S-5S-T	4 / 75	2	2
P-1A-JWOS	1 / 75	0	2

Table 9. Results on hard 80-variable 3-SAT problems

A summary based on results in tables 1,2,3,4,5,6,7,8, and, 9 is shown in Table 10. Multi-flip procedures solved 83 % (2254) of the 2714 problems they attempted. The uni-flip procedure solved only 44 % (1197) of the attempted 2714 problems. 39 % of the attempted problems (1061) were solved only by multi-flip procedures. The multi-flip procedures solved 1057 more problems than the uni-flip procedure.

5 Discussion

Previously developed local search-based methods for SAT solving flip value of only one variable to generate the next truth assignment. Flipping values of multiple variables simultaneously can yield lower solving times and/or reduce the number of iterations needed to solve problems. We reported on eight local search-based procedures which are capable of simultaneously flipping values of multiple variables on an iteration. We reported on their empirical evaluation on 2714 benchmark problems. A brief summary of the results from tables 1,...,9 is shown in Table 10. This shows that the multi-flip approach can be considerably superior to the steepest descent-based uni-flip procedure.

Procedures P-1A-2A and P-1A-2A-T examine many more flips than GSAT, random walk, min-conflict hill

# Vars	# Problems	MF	OMF
20	1000	1000	145
40	97	90	7
50	1001	923	685
60	40	25	21
75	99	47	46
80	76	22	17
90	100	88	81
100	270	55	55
150	31	4	4

Table 10. Summary based on tables 1,2,3,4,5,6,7,8, and 9

climbing and Walk-sat algorithms. This can significantly increase the solving times of P-1A-2A and P-1A-2A-T. Most previously developed local search-based algorithms including the four mentioned above do not conduct steepest descent. In the first step, Walk-sat chooses an unsatisfied clause and flips the value of a variable in it, if the flip does not make any currently satisfied clause unsatisfied. If a flip is not made in the first step, a stochastic flip is made in the second step. Some other local search-based algorithms flip values of a bounded number of variables rather than all, to decide which variable's value to flip. Multi-flip procedures may benefit from stochastic moves that Walk-sat makes. Since the multi-flip procedures are more effective than P-1A and because algorithms like Walk-sat have been much more successful, algorithms in the middle are worthy of investigation. The algorithms in the middle are those which flip values of more than 1 variable on some iterations without examining as many flips as P-1A-2A and P-1A-2A-T. P-1A-JWOS is one such procedure. P-1A-JWOS was more successful than P-1A but not as successful as P-1A-2A. The number of candidate assignments examined by P-1A-JWOS on each iteration is $(n + 2)$ which lies between the number of candidates examined by P-1A (n) and P-1A-2A ($n + \frac{n \cdot (n-1)}{2}$). Ideas from global search-based solvers can be used to intelligently select which variables' values to flip in local search. Procedures like P-1A-JWOS may be more successful than P-1A-2A if they examine some more wisely chosen neighbors than $n + 2$.

The 9 procedures differ because of the different neighborhoods they examine. A Venn diagram showing the relationships among the neighborhoods of the 9 procedures is shown in Fig. 1. The problem of an intelligent selection of neighboring truth assignments to examine, is same as the problem of an automatic selection of neighborhood. The size of the neighborhood used by the 9 procedures is fixed. This can be varied during search. A procedure can examine more neighbors when many clauses are unsatisfied and ex-

amine fewer neighbors when more clauses are satisfied. The length of the tabu list can also be varied during search. Our future work includes strategies for automatically selecting and varying neighborhood during local search. Our future work also includes an investigation of multi-flip variants of walk-sat. Our future work also includes an evaluation of the procedures through multiple runs on same problems. This is very relevant since different runs will have different initial truth assignments.

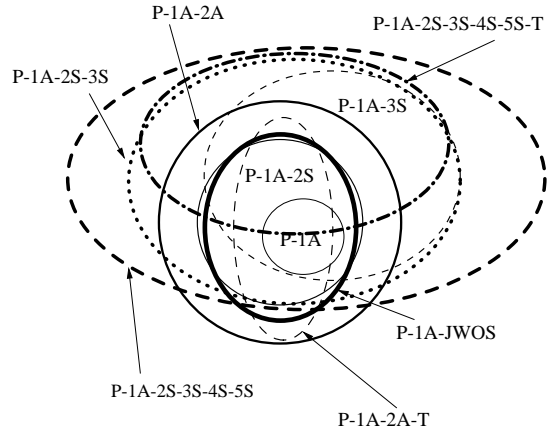


Figure 1. Relations between neighborhoods of the 9 procedures

Kautz & Walser use local search for solving integer linear programming problems (ILPs) [5]. Since ILP and SAT are special kinds of CSPs, multi-flip procedures can be developed for solving ILPs and CSPs. An algorithm which uses local search for solving MAX-SAT and weighted MAX-SAT is reported in [3]. MAX-SAT is the problem of finding truth assignment to variables that satisfies maximum number of clauses in the given instance. W-MAX-SAT is the problem of finding a truth assignment that maximizes the sum of the weights of the satisfied clauses. Given that multi-flip procedures are effective on SAT instances, their effectiveness at solving MAX-SAT and W-MAX-SAT is worthy of study. On most of the 2714 problems, we found that the maximum number of clauses satisfied by some multi-flip procedure was higher than the maximum number of clauses satisfied by P-1A. In fact on most unsolved problems, the multi-flip procedures were unsuccessful because they failed to satisfy one or two clauses. This shows that multi-flip procedures can be very effective on MAX-SAT, W-MAX-SAT, MAX-CSPs and W-MAX-CSPs.

6 Conclusion

Previously developed local search-based SAT solving algorithms flip value of only one variable to generate the next assignment. Simultaneously flipping the values of multiple variables can reduce solving times and/or reduce the number of iterations needed to solve problems and/or satisfy more clauses in case the instance is unsatisfiable and/or satisfy more clauses from satisfiable instances in limited time or limited number of iterations. We reported on eight multi-flip procedures. They all have a low order polynomial running time. We compared the effectiveness of these eight procedures with that of one uni-flip procedure on 2714 hard benchmark SAT instances. The evaluation shows that multi-flip procedures solve many more problems than the uni-flip procedure. In fact multi-flip procedures solved 83 % of the 2714 problems and the uni-flip procedure solved only 44 % of these problems. 1061 problems were solved only by the multi-flip procedures. Multi-flip procedures solved 1057 more problems than the uni-flip procedure. The evaluation also shows that several problems are solved by multi-flip procedures many times faster than the uni-flip procedure. The evaluation also shows that allowing the uni-flip procedure to run for many more iterations than the multi-flip procedures does not always improve the performance of the uni-flip procedure. This shows that making many flips in few iterations can be superior to making a single flip on many more iterations. We discussed the broader usefulness of our work. Multi-flip procedures can be highly effective on MAX-SAT, W-MAX-SAT, MAX-CSP and W-MAX-CSP. Our future work includes development of multi-flip procedures which examine flips of values of more variables than highly successful Walk-sat and which examine far fewer flips than P-1A-2A and P-1A-2A-T.

Acknowledgement: This work has been funded in part by NSF grant IIS-0119630 to Amol Dattatraya Mali. The authors thank Christine Cheng for useful comments on the earlier version of this paper. The authors thank Misha Guterman, Guy Schenk, Allyson Wu, Tristan Rice, Heng Zhou, Paul Schultz and Cindy Rubio for useful discussions about simulated annealing and multi-flip variants of walk-sat and for conducting their preliminary evaluation.

[1] Jun Gu, Local search for satisfiability (SAT) problem, *IEEE Transactions on systems, man, and cybernetics*, Vol. 23, No. 4, July/August 1993, pp. 1108-1129.

[2] John Hooker, *Logic-based methods for optimization*, John Wiley & Sons, Inc., 2000, page 294.

[3] Steve Joy, John Mitchell, and Brian Borchers, A branch and cut algorithm for MAX-SAT and weighted MAX-SAT,

DIMACS series in discrete mathematics and theoretical computer science, Volume 35, 1997, pp. 519-536.

[4] Henry Kautz and Bart Selman, Pushing the envelope: Planning, Propositional logic and Stochastic search, *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 1996, pp. 1194-1201.

[5] Henry Kautz and Joachim Walser, State-space planning by integer optimization, *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 1999, pp. 526-533.

[6] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird, Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, *Artificial Intelligence* 58, 1992, pp. 161-205.

[7] Dale Schuurmans and Finnegan Southey, Local search characteristics of incomplete SAT procedures, *Proceedings of the seventeenth national conference on artificial intelligence (AAAI)*, Austin, Texas, 2000, pp. 297-302.

[8] Wei Wei and Bart Selman, Accelerating random walks, *Proceedings of eighth international conference on principles and practices of constraint programming (CP)*, 2002.

[9] Amol Dattatraya Mali, Hierarchical task network planning as satisfiability, *Proceedings of European Conference on Planning (ECP)*, 1999, pp. 122-134.

[10] Amol Dattatraya Mali, Plan merging and plan reuse as satisfiability, *Proceedings of European Conference on Planning (ECP)*, 1999, pp. 84-96.

[11] Amol Dattatraya Mali, On the hybrid propositional encodings of planning, *Computational Intelligence journal*, Vol. 18, No. 3, 2002, pp. 386-419.

[12] Mark Iwen and Amol Dattatraya Mali, DSATZ: A directional SAT solver for planning, *Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2002, pp. 199-208.

[13] Amol Dattatraya Mali, Enhancing HTN planning as satisfiability, *Proceedings of International conference on Artificial Intelligence and Soft Computing (ASC)*, 2000, pp. 325-333.

[14] Amol Dattatraya Mali and Subbarao Kambhampati, On the Utility of Plan-space (Causal) encodings, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1999, pp. 557-563.