ENHANCING GASAT BY USING THE DISCRETE LAGRANGE-MULTIPLIER ALGORITHM

Yousef Kilani
Prince Al-Hussein Bin Abdullah II School of Information Technology
The Hashemite University
AL-Zarqa, Jordan
email: y_kilani@yahoo.com

ABSTRACT
The satisfiability problem (SAT), as one of the six basic core NP-complete problems, has been the deserving object of many studies in the last two decades [3, 2]. GASAT [4, 3, 2] is one of the current state-of-the-art genetic algorithms for solving SATs. Besides, the discrete lagrange-multiplier (DLM) [7, 8] is one of the current state-of-the-art local search algorithms for solving SATs. GASAT is a hybrid algorithm of the genetic and tabu search techniques. GASAT uses tabu search to avoid restarting the search once it converges. In this paper, we improve GASAT by replacing the tabu search by the DLM algorithm. We show that the performance of the new algorithm is far better than the performance of GASAT in solving most of the benchmark instances.

KEY WORDS
GASAT, DLM, and SAT.

1 Introduction

The *satisfiability problem* (SAT), as one of the six basic core NP-complete problems, has been the deserving object of many studies in the last two decades [3]. In addition to its theoretical importance, SAT has a large number of practical applications such as VLSI test and verification, the design of asynchronous circuits, sports planning and so on [3].

There are many methods of search techniques for solving the SATs like backtrack-based, genetic, neural network and local search algorithms. The genetic algorithm for the SATs (GASAT) [4, 3, 2] and SAT-WAGA [6] are two of the current state-of-the-art genetic algorithms for solving the SATs. Besides, the discrete lagrange-multiplier (DLM) [7, 8] and the exponentiated subgradient algorithm (ESG) [10] are the current state-of-the-art local search algorithms for solving the SATs. GASAT is a hybrid algorithm based on SAT specific crossover operators combined with tabu search [2]. Within GASAT, specific crossover operators are used to identify particular promising search areas while tabu search performs an intensified search of solutions around these areas [2]. In such a way, we hope to be able to achieve a good compromise between intensification and diversification in the search procedure [2].

In this paper, we replace the tabu search algorithm in GASAT by DLM [7, 8, 13]. We name the new algorithm DGASAT. We show through experiments that the results of the DGASAT is far better than the results of GASAT in solving most of the benchmark instances.

The rest of this paper is organized as follows. Section 2 presents the SAT. Section 3 introduces the state-of-the-art genetic algorithm: GASAT. Section 4 presents the DLM local search algorithm. After that, section 5 presents our new algorithm DGSAT. Section 6 compares the results of GASAT with DGASAT. The last section gives the conclusion remarks.

2 The Satisfiability Problem

In a SAT, a *propositional variable* can take the value of either 0 (false) or 1 (true). A *literal* is either a variable x or its complement $\bar{x}$. A literal l is *true* if l assumes the value 1; l is *false* otherwise. A *clause* is a disjunction of literals, which is true when one of its literals is true. For simplicity we assume that no literal appears in a clause more than once, and no literal and its negation appear in a clause. A SAT is a propositional logic formula which consists of a finite set of clauses (treated as a conjunction). Let $\overline{l}$ denote the complement of literal l: $\overline{l} = \bar{x}$ if $l = x$, and $\overline{l} = x$ if $l = \bar{x}$. Let $\overline{L} = \{\overline{l} \mid l \in L\}$ for a literal set L. The *conjunctive normal form* propositional logic formula is a conjunction of a set of clauses. There are many encoding techniques to convert a propositional logic formula which is not in a conjunctive normal form to a conjunctive normal form (see for more details [11, 12]). A *valuation* is a complete assignment to all the variables in the SAT. A *solution* is a valuation that satisfies all the clauses in the SAT.

3 The GASAT Algorithm

Lardeux et al. [3] and Hao et al. [4] introduce GASAT [1]. Figure 1 presents GASAT. In GASAT, each member of the population is a binary string of n bits, where the n bits represent the variables in the problem. Each bit can take the value of either 0 or 1 which are the same values a Boolean

[1] The source code is downloadable from http://www.satlive.org/satkwd.jsp?kwd=79

variable can take. GASAT starts the search by generating an initial population randomly in which each variable in each member takes a value randomly.

1- p = CreatePopulation(),
   Improve each member in p by Tabu Search
2- while (no solution found and the
        time is not over)
3- b = SelectBestIndividuals(p)
4- Choose x and y from b /*the parents*/
5- z = Crossover(x, y) /*z is the child*/
6- z = Tabu(z, $maxflip$)
    /*improve z by Tabu Search*/
7- if z is better than the worst member in b
8- add z to p
9- else discard z
10- If solution found return a solution
11-else return the best answer found

Figure 1: GASAT

Then the fitness value of every member is calculated. The fitness value of a member is equal to the number of clauses which are not satisfied by this member. Therefore, the low value of a member's fitness indicates that this member is good. Zero is the optimal fitness. GASAT chooses a set of the best individuals b from the population, where the size of b is a parameter. It then chooses randomly parents from b. In order to avoid the case in which the selected parents are similar and to keep diversity in the population, GASAT accepts the selected parents if there is a k hamming distance between these selected parents, where k is a parameter. GASAT selects two parents and uses the crossover operator to produce a child. This child is further improved by using the *tabu search local search algorithm*. In each step (move) of tabu search, GASAT chooses the best variable to flip which is not in the tabu list. The best variable is the variable once flipped satisfies the maximum number of clauses with compare to the other variables. The size of the tabu list and the number of the tabu search moves are parameters in GASAT. The tabu list is a fixed size array of size x. It stores the last x flipped variables. Once a variable is flipped, it is not allowed to be flipped in the coming x flips. After a variable is flipped, the first flipped variable in the tabu list is removed and is replaced by the newly flipped variable (in a FIFO manner). But, the tabu search used in GASAT has an exception; if the move leads to a better state than the best ever found, GASAT allows this move even if this move is in the tabu list. The tabu search used in GASAT is the same as the one used by Mazure et. al. [1].

The main goal of the crossover operator in GASAT is to create diversified and potentially promising new individuals [3]. GASAT has three different crossover operators: corrective clause, corrective clause and truth maintenance, and Fleurent and Ferland. In an experimental comparisons, Lardeux et al. [3] show that the performance of these crossover operators are far better than the classic uniform crossover operator. The three crossover operators produce one child from the parents: Crossover(x, y) $\Rightarrow$ child z. GASAT uses one of the three crossovers at a time according to the following three cases: a clause is satisfied by the two parents, a clause is satisfied by one parent, and a clause is not satisfied by the two parents.

The classic uniform crossover operator assigns value for each variable by choosing one of the values assigned for this variable by the parents.

The corrective clause crossover: when a clause c is not satisfied by the two parents x and y, this crossover flips one of the variables v in c to satisfy c. In order to choose v, it passes by all the clauses which are currently unsatisfied by x, y and the child z, where the variables in z initially have no values, and computes $\delta(f)$ for every variable f appears in these clauses, where $\delta(f) = imp(x, f) + imp(y, f)$, $imp(d, e) = satisfied(d, e)$-$unsatisfied(d, e)$, $satisfied(d, e)$ returns the number of unsatisfied clauses in valuation d that becomes satisfied by flipping e, and $unsatisfied(d, e)$ returns the number of satisfied clauses in valuation d that becomes unsatisfied by flipping e. The corrective clause crossover flips the variable f which has the maximum value of $\delta(f)$.

The corrective clause and truth maintenance crossover applies the corrective clause crossover and then applies the following procedure. Compute $\delta(f)$ for every variable f appears in clause c, where c is a clause satisfied by the two parents and is not satisfied by the child z. Finally, the corrective clause and truth maintenance crossover chooses the variable to flip with maximum δ.

Fleurent and Ferland's crossover: when a clause c is satisfied by one parent and not satisfied by the other parent, this crossover assigns the values for the variables appearing in c the same values taken in the parent that satisfies c. These variables and their values are then copied to the child z.

After applying the previous crossovers, if any variable in z is left with no value, GASAT chooses one of the values for this variable from its parents randomly, that is, if v in z has no value, and v has the values 0 and 1 in the parents x and y respectively, GASAT choose either 0 or 1 randomly.

After a child z is produced (line 5) by the crossover operator and further improved by the tabu search algorithm (line 6), it is inserted (line 8) in the population p if it is better than the worst member in b. The tabu search moves $maxflip$ times before returning the answer. Note that the size of the population p is increasing.

4 The DLM Algorithm

DLM [7, 8, 13] is a discrete Lagrange-multiplier-based local-search method for solving the SATs, which are first transformed into a discrete constrained optimization problem. Figure 4 shows the core of DLM[2]. Experiments

[2]Downloadable from www.manip.crhc.uiuc.edu/Wah

confirm that the discrete Lagrange multiplier method is highly competitive with other satisfiability problems solving methods [7].

1- DLM(c)
2- let s be a random valuation for $var(c)$
3- λ = 1; $nflips$ = 0
4- while ($L(s, \lambda) > 0$ and
($nflips \leq maximum\_flips$))
5- $min := L(s, \lambda)$
6- $best := \{\}$
7- $unsat$:= the literals in unsat clauses
8- for each literal $l \in unsat$
9- $s' := s - \{l\} \cup \{\bar{l}\}$
10- if ($L(s', \lambda) < min$)
/*it is a better downhill move*/
11- $min := L(s', \lambda)$
12- $best := \{l\}$
13- $s := s'$
14- else if (($L(s', \lambda) = min$) and
(l is not in the tabu list))
15- $best := best \cup \{l\}$
16- if ($best$ is empy) it is trap, do learning
17- else s := s - {var := a randomly chosen
element from $best$} $\cup$ { $\overline{var}$ }
$nflips$ = $nflips$ + 1
18- if (Lag. M. update condition holds)
19- $\lambda := \lambda + x$
20-return s

Figure 2: DLM (core algorithm)

In DLM, each clause c is given a penalty function on states, so $c_i(s) = 0$ if state s satisfies constraint c_i, and $c_i(s) = 1$ otherwise. DLM performs a search for a saddle point of the Lagrangian function

$$L(s, \lambda) \;=\; \lambda \cdot c(s) \;=\; \Sigma_i \lambda_i \times c_i(s)$$

where λ are a vector of Lagrange multipliers, one for each constraint, which give the "penalty" for violating that constraint. The saddle point search changes the state to decrease the Lagrangian function, or increase the Lagrange multipliers.

In Figure 4, line 1 shows that the input to DLM is a set of clauses c. Line 2 makes a random initialization to all the variables. Line 3 initializes λ and $nflips$ to 1 and 0 respectively. Line 4 repeats the search until it finds a solution or reaches a maximum number of flips. $L(s, \lambda) = 0$ means no constraint is violated, i.e. $c(s) = 0$, for each constraint c in the problem.

Lines 5, 6 and 7 set min, $best$ and $unsat$ to the Lagrangian function of the current state s, empty and the set of all the literals in the unsatisfied clauses respectively. We call the local move if it is to a better and equal neighbours a *downhill* and *flat* moves respectively. Lines 9-15 save the best neighbors in $best$. Note that every variable in $best$ must either make a downhill move or it is not in the tabu list making a flat move. DLM restricts the tabu list to the flat moves only. If $best$ is empty then it is a trap, line 16 makes *learning*. In learning, DLM increases the Lagrangian multipliers of the unsatisfied/all clauses according to a parameter. Line 17 moves from the current state to one of the best neighbours by flipping the variable var which has been chosen randomly from $best$. Lines 18-19 update the Lagrangian multipliers according to a parameter.

5 The DGASAT Algorithm

It is a common knowledge that DLM performs better than tabu search technique. In our implementation, we use the GASAT and DLM algorithms as provided by their creators while incorporating DLM in GASAT. Using the source code of these algorithms as provided by their creators ensures that these algorithms are in the form as their creators want them to be.

The DGASAT algorithm is the same as the GASAT algorithm except that we replace the call to tabu search by a call to DLM. The underline part (line 6) in figure 3 is the new change over GASAT. The Call_DLM function (figure 4) calls DLM after initializing λ and $nflips$ to 1 and 0 respectively and restricting the number of move to $maxflips$ before the DLM returns the answer. DLM returns the solution if it finds it before reaching the moves to $maxflips$ or returns the last valuation if it could not find the solution. The remaining parts of GASAT and DGASAT are the same.

1- p = CreatePopulation()
Improve each member in p by Tabu Search
2- while (no solution found and
the time is not over)
3- b = SelectBestIndividuals(p)
4- Choose x and y from b /*the parents*/
5- z = Crossover(x, y) /*z is the child*/
6- z = Call_DLM(c, z, $maxflip$)
/*improve z by DLM */
7- if z is better than the worst mem. in b
8- add z to p
9- else discard z
10- If solution found return a solution
11-else return the best answer found

Figure 3: The DGASAT algorithm

Line 1 in GASAT and DGASAT is the same since while experimenting with DGSAT, using tabu search for the initial population shows better performance than using DLM in this population.

1- Call_DLM(c, z, $maxflips$)
/*c is the set of clauses, z is the current valuation*/
2- let $s = z$ be the current valuation
3- λ = 1; $nflips$ = 0
4- while ($L(s, \lambda) > 0$ and ($nflips \leq maxflips$))
5- lines 5-20 from figure 2

Figure 4: The Call_DLM function

6 Experiments

In our experiments, we use the source code of the GASAT and DLM algorithms as provided by their creators. In addition, we consult the authors while tuning the parameters. Luckily, the authors replied to all of our queries.

We used a suite of CSPs taken from [5]. We first transform the problem instances into the satisfiability problems. Table 1 shows these instances and the number of variables and clauses in each instance. We use a PC with Pentium III 800 Mhz and 256 MB memory to get the results.

We tuned the parameters of the GASAT but we found that the default parameters are the best as recommended by their creators as well.

The default parameters of GASAT are:

- the initial size of the population: 30,
- the size of the best individuals: 15,
- the minimum hamming distance between two parents: 1,
- the length of the tabu list: (the number of variables in the problem)/(20),
- the number of flips in the tabu search ($maxflips$): 1000 .

In DGASAT, we use the same parameter for DLM, the number of search moves ($maxflips$), used in GASAT for tabu search.

DLM distribution has five different sets of parameters. We tuned these parameters for each set of the benchmark instances. In this tuning, in each type of the benchmark instances, we choose one instance randomly and we run DGASAT for 20 runs. We then choose the best set of the DLM parameters for this type of the benchmark instances. Table 2 shows the PS, parameter sets, values for these instances.

In order to make sure the fair comparisons between GASAT and DGASAT, we use the same time function for both algorithms. GASAT and DLM use the gcc compiler under Linux or Unix platform. The time function we use calculates the sum of the user time and the system time.

Tables 3 and 4 show the results of GASAT and DGASAT respectively. We run each instance for 20 runs and each run is terminated if it reaches a 10000 CPU time without finding an answer. The information presented in each table includes:

Instance	vars	Cls
N queens		
10 queen	100	1,480
20 queen	400	12,560
50 queen	2,500	203,400
100 queen	10,000	1,646,800
Random permutation generation		
pp50	2,475	159,138
pp60	3,568	279,305
pp70	4,869	456,129
pp80	6,356	660,659
pp90	8,059	938,837
pp100	9,953	1,265,776
Increasing permutation generation		
ap10	121	6,71
ap20	441	4,641
ap30	961	14,911
ap40	1,681	34,481
Latin square		
magic-10	1,000	9,100
magic-15	3,375	47,475
magic-20	8,000	152,400
magic-25	15,625	375,625
magic-30	27,000	783,900
magic-35	42,875	1,458,975
Hard graph-coloring		
g125n-18c	2,250	70,163
g250n-15c	3,750	233,965
g125n-17c	2,125	66,272
g250n-29c	7,250	454,622
Tight random		
rcsp-120-10-60-75	1,200	331,445
rcsp-130-10-60-75	1,300	389,258
rcsp-140-10-60-75	1,400	451,702
rcsp-150-10-60-75	1,500	518,762
rcsp-160-10-60-75	1,600	590,419
rcsp-170-10-60-75	1,700	666,795

Table 1. Benchmark of the satisfiability instances.

Instance	PS
N queens	2
Random permutation generation	1
Increasing permutation generation	5
Latin square	5
Hard graph-coloring	4
Tight random	4

Table 2. The DLM parameter sets (PS) for the Benchmark instances used in DGASAT.

- the success ratio of the 20 runs.
- the average time in seconds for the success runs of the 20 runs.

It is clear from tables 3 and 4 that the result of DGASAT is far better than the result of GASAT in time and in success ratio except in solving the graph coloring problems. DGASAT could solve instances that GASAT could not, like the pp60 and pp70 in random permutation generation problems and magic20 and magic25 in latin square problems.

7 Conclusion

In this paper, we have improved the GASAT algorithm by replacing the tabu search local search algorithm by the DLM algorithm. The resulted algorithm, DGASAT, is far better than GASAT in solving most of the benchmark instances. In addition, DGASAT could solve instances that GASAT could not.

References

[1] Mazure, B., Sais, L. and Gregoire, E., Tabu search for SAT, *Proceeding of the AAAI-97/IAAI-97*, Providence, Rhode Island, 1997, 281-285.

[2] Lardeux, F., Saubion, F. and Hao, J., GASAT: a genetic local search algorithm for the satisfiability problem, *Evolutionary Computation Journal*, 14(2), 2006, 223-253.

[3] Lardeux, F., Saubion, F. and Hao, J., GASAT: a genetic local search algorithm for the satisfiability problem, *Proceeding of International Conference in Evolutionary Computation*, 2005.

[4] Hao, J., Lardeux, F. and Saubion, F., A hybrid genetic algorithm for the satisfiability problem, *Proceeding of the 1st International Workshop on Heuristics*, 2002.

[5] Choi, K., Lee, J. and Stucky, P. A Lagrangian reconstruction of GENET. *Journal of Artificial intelligence*, 123, 2000, 1-39.

[6] Yingbiao, L., A genetic algorithm based on adapting clause weights, *Chinese Journal of Computer*, 7, 2005, 20-32.

[7] Wu, Z. and Wah, B., Trap escaping strategies in discrete Lagrangian methods for solving hard satisfiability and maximum satisfiability problems, *Proceeding of the 16th National Conference on Artificial Intelligence*, 1999a, 673-678.

[8] Wu, Z. and Wah, B., Solving hard satisfiability problems: a unified algorithm based On discrete lagrange

Instance	Succ	Time
N queens problem		
10queen	20/20	0.03
20queen	20/20	0.13
50queen	20/20	5201.23
100queen	0/20	-
Random permutation generation problems		
pp50	20/20	4811.02
pp60	0/20	-
pp70	0/20	-
pp80	0/20	-
pp90	0/20	-
pp100	0/20	-
Increasing permutation generation problems		
ap10	10/20	5331.14
ap20	0/20	-
ap30	0/20	-
ap40	0/20	-
ap50	0/20	-
Latin square problems		
magic-10	20/20	0.50
magic-15	4/20	9311.12
magic-20	0/20	-
magic-25	0/20	-
magic-30	0/20	-
magic-35	0/20	-
Hard graph-coloring problems		
g125n-18c	20/20	4812.01
g250n-15c	20/20	555.03
g125n-17c	0/20	-
g250n-29c	0/20	-
Tight random CSPs		
rcsp-120	20/20	17.00
rcsp-130	20/20	14.13
rcsp-140	20/20	19.33
rcsp-150	20/20	12.09
rcsp-160	20/20	17.36
rcsp-170	20/20	33.43

Table 3. The results of GASAT

Instance	Succ	Time
N queens problem		
10queen	20/20	0.00
20queen	20/20	0.01
50queen	20/20	3394.95
100queen	0/20	-
Random permutation generation problems		
pp50	20/20	1003.15
pp60	20/20	2687.24
pp70	13/20	7036.12
pp80	0/20	-
pp90	0/20	-
pp100	0/20	-
Increasing permutation generation problems		
ap10	20/20	12.5
ap20	0/20	-
ap30	0/20	-
ap40	0/20	-
ap50	0/20	-
Latin square problems		
magic-10	20/20	0.01
magic-15	20/20	35.45
magic-20	20/20	1661.90
magic-25	20/20	5205.20
magic-30	0/20	-
magic-35	0/20	-
Hard graph-coloring problems		
g125n-18c	0/20	-
g250n-15c	20/20	892.95
g125n-17c	0/20	-
g250n-29c	0/20	-
Tight random CSPs		
rcsp-120	20/20	0.02
rcsp-130	20/20	0.02
rcsp-140	20/20	0.04
rcsp-150	20/20	0.03
rcsp-160	20/20	0.06
rcsp-170	20/20	0.90

Table 4. The results of DGASAT

multipliers, *Proceeding of the 11th IEEE International Conference on Tools with Artificial Intelligence*, 1999b, 210-217.

[9] Goldberg, E., Genetic algorithms in search. *Proceeding of International Conference in Optimization & Machine Learning*, 1989.

[10] Schuurmans, D., Southey, F. and Holte, R., The exponentiated subgradient algorithm for heuristic boolean programming, *Proceeding of the International Joint Conference on Artificial Intelligence*, 2001, 334-341.

[11] Thiffault, C., Bacchus, F. and Walsh, T., Solving non-clausal formulas with DPLL search, *Principles and Practice of Constraint Programming*, 2004, 663-678.

[12] Tseitin, G., On the complexity of proofs in propositional logics, *Automation of Reasoning: Classical Papers in Computational Logic*, 2, 2001, 50-57.

[13] Wah, B.W. and Wu, Z., Penalty formulations and trap-avoidance strategies for solving hard satisfiability problems, *Journal of Computer Science and Technology, Springer-Verlag*, 20(1), 2005, 3-17.