Speeding Up the ESG Algorithm

Yousef Kilani[1] and Abdullah. Mohdzin[2]

[1]Prince Hussein bin Abdullah Information Technology College, Al Al-Bayt University, Jordan
[2]Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia , Malaysia
E-mail: y_kilani@yahoo.com and amz@ftsm.ukm.my

Abstract

Local search (LS) methods heuristically find a solution for constraint satisfaction problems (CSP). LS starts the search for a solution from a random assignment. LS then examines the neighbours of this assignment to determine a better neighbour valuation to move to. It repeats this process of moving from the current assignment to a better assignment until it finds a solution that satisfies all constraints.

ICM considers some of the constraints as hard constraints that are always satisfied. In this way, ICM reduces the possible neighbours in each move and hence the overall search space. ICM chooses the hard constraints in such away that the space of valuations that satisfies these constraints is connected in order to guarantee that a local search can reach any solution from any valuation in this space.

In this paper, we incorporate ICM into one of the most recent local search algorithm, ESG, and we show the improvement of the new algorithm. We ran this algorithm in different SAT problems.

Introduction

A (*CSP)* (Makworth, 1977) is a tuple (Z, D, C), where Z is a finite set of variables, D defines a finite set D_x, called the domain of x, for each $x \in Z$, and C is a finite set of constraints restricting the combination of values that the variables can take (Fang et al. 2002). A *solution* is an assignment of values from the domains to their respective variables so that all constraints are satisfied simultaneously (Fang et al. 2002). CSPs are known to be NP-hard in general (Fang et al. 2002).

LS techniques, for example GSAT (Selman, Levesque, and Mitchell 1992), WalkSAT (Selman and Kauts 1993), DLM (Wu and Wah, 1999, Wu and Wah 2000) the min-conflicts heuristic (Minton 1992) and ESG (Wu and Wah 2000) have been successful in solving large CSPs (Fang et al. 2002). LS starts the search by generating an initial random variable assignment (or state). It then makes local adjustments to the assignment iteratively until a solution is reached. The local adjustment happened by changing one variable assignment for instance. One criteria for this change is to change the variable's assignment to a new assignment so that the new state after this change satisfies more number of constraints.

LS searches the search space to look for solutions using some heuristic function. (Schuurmans and Southey, 2000).introduced three measures of LS performance: depth, mobility and coverage. Depth measures how many clauses remain unsatisfied as the search proceeds, mobility measures how rapidly a local search moves in the search space and coverage measures how systematically the search explores the entire space. The efficiency of a LS algorithm depends on three things (Fang et al. 2002): (1) the size of the search space (the number of variables and the size of the domain of each variable), (2) the search surface (the structure of each constraint and the topology of the constraint connection) and (3) the heuristic function (the definition of neighbourhood and how a ``good'' neighbour is picked). The Island Confinement Method (ICM) aims to reduce the size of the search space (Fang et al. 2002).

In this paper, we incorporate ICM into ESG and we show the gained performance by experimenting with different SAT instances.

The rest of this paper is organized as follows. First, we give the necessary background and definitions. Second, we demonstrate the general process of any LS algorithm. Third, we describe the idea of ICM we used to improve ESG. After that we demonstrate how we incorporate the ICM into ESG. This is followed by the experimental results, conclusion and the future work.

Background

Given a CSP(Z, D, C) (or simply C), we use *var(c)* to denote the set of variables that occur in constraint $c \in C$. If $|var(c)| = 2$ then c is a *binary constraint*. In a binary CSP, each constraint $c \in C$ is binary. A *valuation* for variable set $\{x_1,\ldots, x_n\} = Z$ is a mapping from variables to values denoted $\{x_1 \rightarrow a_1, \ldots, x_n \rightarrow a_n\}$, where $a_i \in Dx_i$. A *state* of a CSP is a valuation for Z. A state s is a *solution* of a constraint c if s makes c true. A state s is a solution of a CSP (Z, D, C) if s is a solution to all constraints in C simultaneously. Let *unsat* to be the set of literals occurring in the unsatisfied clauses.

SAT problems are a special problem of CSPs. A *propositional variable* can take the value of either 0 (false)

or 1 (true). A literal is either a variable x or its complement 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 *satisfiability problem* (SAT) consists of a finite set of clauses (treated as a conjunction). Let *l'* denote the complement of literal *l*: *l'* = *x'* if *l* = *x*, and *l'* = *x* if *l* = *x'*. Let $L' = \{l' \mid l \in L\}$ for a literal set *L*. Since we are dealing with SAT problems we will often treat states as sets of literals. A state $\{x_1 \rightarrow a_1, \ldots, x_n \rightarrow a_n\}$ corresponds to the set of literals $\{ x_j \mid a_j = 1\} \cup \{x'_j \mid a_j = 0\}$.

In this research, we focus on a specific class of SAT problems, namely those encoding from a binary CSP. We can encode any binary CSP (Z, D, C) to a SAT problem as follows. Every CSP variable $x \in Z$ is mapped to a set of propositional variables $\{x_{a1}, \ldots, x_{an}\}$ where $D_x = \{a_1, \ldots, a_n\}$. For every $x \in Z$, SAT(Z, D, C) contains the *at-least-one-on* clause $x_{a1} \vee \ldots \vee x_{an}$ which ensures that any solution to the SAT problem gives a value to x. Each binary constraint $c \in C$ with var(c) = {x, y} is mapped to a series of clauses. If $\{x \rightarrow a, y \rightarrow b\}$ is not a solution of *c* we add the clause $x_a' \vee y_b'$ to SAT(Z, D, C), where $\{x, y\} \subseteq Z$. This ensures that the constraint *c* holds in any solution to the SAT problem. We call these clauses *problem clauses*.

The previous formulation allows the possibility that in a solution, some CSP variable *x* is assigned two values. Choosing either value is guaranteed to solve the original CSP. This method is used in the encoding of CSPs into SAT in the DIMACS archive. When a binary CSP (Z, D, C) is translated to a SAT problem SAT(Z, D, C), all the clauses have the form $x' \vee y'$ except the *at-least-one-on* clauses.

There are two types of algorithms for solving CSP: complete search algorithms and incomplete search algorithms. The complete search algorithms include chronological backtracking and dynamic backtracking. The incomplete search algorithms include: neural network, the genetic and LS algorithms. Our interes is in LS.

Local Search

A LS solver moves from one state to another by a local move. The *neighbourhood* n(s) is the set of neighbour states to s that are reachable in a single move from state *s*. The single move depends on the actual heuristic function used. This function decides the neighbour state to move to. A *Hamming distance* between states s_1 and s_2 measures the number of differences in variable assignment of s_1 and s_2. A *vector variable* $vec(x) = (x_1, \ldots, x_n)$. In this paper, we are interested in SAT problems. We assume n(s) as the states which are at a Hamming distance of 1 from the state *s*. We refer to flipping a literal *l* to mean flipping the variable occurring in the literal. A local move from state *s* is a transition, $s \Rightarrow s'$, from s to $s' \in n(s)$.We consider a SAT problem as a vector of clauses vec(c). The general LS algorithm starts the search from a random assignment s. This assignment represents the current state. Some LS algorithms may start the search from a heuristically chosen valuation instead of a random one. LS then moves from the current state s to a better neighbour n(s). If there is no better neighbour then it is local minima, *trap*. A trap is a non-solution state in which no further improvement can be made. Different LS algorithms follow different ways in escaping this local minimum. GSAT (Selman, Levesque, and Mitchell 1992) and the min-conflicts heuristic (Minton 1992) use random restart, while DLM (Wu and Wah, 1999, Wu and Wah 2000) and ESG (Schuurmans and Southey 2000). modify the landscape of the search surface. It escapes this trap. Some LS algorithms may include a *restart* and/or *tabu list*. The restart prevents LS from spending very long time in searching one part of the search space. During the process of searching, some variables may be chosen frequently for flipping within a short time which in turn stagnates the search. Therefore, once a variable is flipped, tabu list prevents this variable from flipping in the coming few flips.

If the search could not find a solution within a number of flips it restarts the search. In general, different local search algorithms have different structure. For instance, some local search algorithms do not use the tabu list, others do not do a restart after a certain number of flips, others have different way of escaping from the trap etc. Sometimes, even though the difference is very tiny but it has very big influence in the results.

The theoretical understanding of the behavior of the local search algorithm is still not yet known and most of the research in this area depends on the experiments.

The Island Confinement Method

The ICM (Fang et al. 2002). is based on the observation: the solution space of the conjunction of any subset of constraints in *P* encloses all solutions of *P*. Solving a CSP thus amounts to locating this space to all the constraints in *P*, which could be either points or regions scattered around in the entire search space. The solution space of constraints *D* is connected if the search can move between any two solutions of *D* without violating any constraint in *D*. The idea of ICM works by finding a set of constraints which are connected, it then starts the search from an assignment which satisfies all these constraints and finally restrict LS to search within this space instead of searching in the entire problem space. Therefore, the search space becomes smaller and it contains all the solutions giving higher chance to LS to find a solution in a short time.

Let *sol(C)* denotes the set of all solutions to a conjunction of a set of constraints *C*, in other words the solution space of C. A set of constraints C is an *island* if, for any two states $s_0, s_n \in sol(C)$, there exist states $s_1, \ldots, s_{n-1} \in sol(C)$

such that $s_i \Rightarrow s_{i+1}$ for all $i \in \{0,\ldots, n-1\}$. That is we can move from any solution of C to any other solution using local moves that stay within the solution space of C.
Let *lit(c)* denote the set of all literals of a set of clause *c*. A set *C* of clauses is *non-conflicting* if there does not exist a variable x such that *x, x' ∈ lit(C)*. A non-conflicting set C of clauses forms an island (Fang et al. 2002). Therefore, in a SAT encoded from a binary CSP, the problem clauses are an island*/island clauses* since no variable and its complement appear in these clauses. Given a SAT problem, we can incorporate ICM into any LS algorithm by the following steps: we split the clauses into c_i and c_r, where c_i and c_r are the island and the at least-one-on clauses respectively. Make an initial valuation that satisfies c_i; getting inside the island. c_i consists of clauses of the form $x' \vee y'$. An arbitrary extension of $lit(c_i)$ to all variables can always be such an initial valuation. Restricting the search to search within the at-least-one-on clauses while satisfying the problem (island) clauses. To do so, when we are in a state s, we exclude literals l from flipping when $s' = s - l \cup l'$ does not satisfy c_i. Hence we only examine states that are in n(s) and satisfy c_i'.

Incorporating ICM into the ESG Algorithm

The exponentiated subgradient algorithm (ESG) (Downloadable from http://ai.uwaterloo.ca/\~dale/papers.html) is a general-purpose Boolean linear program (BLP) search technique (Schuurmans and Southey 2000). A BLP is a constrained optimization problem where one must choose a set of binary assignments to variables $vec(x) = (x_1,\ldots, x_n)$ to satisfy a given set of *m* linear inequalities $vec(d_1)$. $vec(x) \leq b_1$,..., $vec(d_m)$ $vec(x) \leq b_m$, where $vec(d_i)$ and b_i are constants, where i=1,…, m, while simultaneously optimizing a linear side objective vec(a). vec(x) (Schuurmans and Southey, F. 2000). (Schuurmans and Southey, F. 2000) describe how to encode a SAT problem as BLP. The Lagrangian function (LG) for a SAT problem of *m* clauses $vec(c) = (c_1,\ldots, c_m)$ and k_i literals in each c_i can be given by the following equation:

$$L(s, vec(c)) = \sum_{i=1}^{m} y_i ZO(c_i)$$

where y_i is the real valued Lagrangian multiplier (LM) associated with constraint c_i, $ZO(c_i) = 1$ or 0 when c_i is violated or satisfied by *s* respectively and s is the current assignment. The objective is to minimize *L(s, vec(c)).* The following is the ESG algorithm*:*

```
ESG(vec(c))
  Let s be a random valuation
  vec(y) := 1,  tabulit  := ∅
  while (max number of tries not reached yet)
    while L(s, vec(c)) > 0 and (max n. flips is not over))
      un = unsat
      if (un is empty) then it is a trap
        rand  := choose a random number between 0-100
        var    := choose any variable randomly
        if ((rand) < noise) s - {var} ∪ {var'}
        else learn: increase the LMs for the unsat clauses
      else choose the best l from unsat
          s = s - {l} ∪  {l'}
      If (weight update condition holds) update vec(y)
    return s
```

ESG starts the search from a random assignment. It then searches for a solution and stops if it reaches a maximum number of flips. In each try, ESG restarts the search. ESG flips one of the best variables from the unsatisfied clauses of the current assignment to move to a better neighbour state. The better neighbour state is the state which has smaller LG than the LG of the current state. Note that flipping a variable from the unsatisfied clauses to move to a better neighbour is a common practice in local search algorithms like DLM and WalkSAT. When there is an ESG trap, with probability *noise*, where *noise* ≤ 100, ESG makes noise by flipping any variable randomly and with probability *100-noise,* ESG *learns* by increasing the LMs of the currently unsatisfied clauses. Note that a clause has higher LM means this clause has been involved in a higher number of traps. Therefore, in each move, ESG chooses to flip a variable from the unsatisfied clauses to satisfy the clauses of higher LMs in the next move in order to reduce LG. LG = 0 means all the clauses are satisfied. ESG updates the LMs after a certain number of flips to prevent the clauses from having very high LMs.
The following is the EI algorithm, the ICM incorporating into ESG:

```
EI(vec(c))
  split vec(c) into vec(c_i) and vec(c_r)
  make an initial valuation s that satisfies vec(c_i)
  vec(y) := 1,  tabulit  := ∅
  while (max number of tries not reached yet)
    while (L(s, vec(c)) > 0 and (max n. flips is not over))
      un = {l ∈ unsat | s ∉ sol(c_r) and
              (s - l ∪ l') ∈ sol(c_i)}
      if (un is empty) then it is an island trap
        rand := choose a random number between 0-100
        vars := ∪ {every var in the problem | if
                    (s-{var} ∪ {var'}) ∈ sol(c_i)}
        var  := choose a variable from vars randomly
        if ((rand) < noise) s - {var} ∪ {var'}
        else escapes this island trap in the same way
             mentioned in (Fang et al. 2002)
      else choose the best l from un
          s = s - { l } ∪ { l'}
  return s
```

We implemented EI (the source code file can be taking by emailing the authors) by modifying the code of distribution

of ESG. The input to EI is the set of SAT clauses. EI splits the clauses into island clauses and none island clauses. EI starts the search from a random valuation inside the island by initial assignment which satisfies all the island clauses. By doing so, we are getting inside the island. As we mentioned previously that an arbitrary extension to lit(c_i') can be such an initial assignment. A *free literal* is a literal once flipped the search will not violate any of the island clauses so that the search will remain searching inside the island. *un* saves the free literals from the unsatisfied clauses. An *island trap* happens when flipping any of the literals in the unsatisfied clauses violates at least one of the island clauses and hence gets the search outside of the island. To escape this trap, EI makes noise by flipping any free literal if the chosen probability is less than *noise*, where *noise* is a parameter. EI escapes the island trap using the same strategy used when escaping from the island trap in DLMI (Fang et al. 2002). In this strategy, we heuristically choose to free one of the literals in the unsatisfied clauses by flipping some literals from the satisfied clauses. For instance, suppose we want to free the literal x from the unsatisfied clauses. We know that flipping x violates the island clauses, say, c_1 and c_2. Flipping x violates c_1 and c_2 means x is the only true literal in each of c_1 and c_2. Therefore, we try to make one more literal true in each of c_1 and c_2. After doing so, there are two true literals, where x is one of them, in each of c_1 and c_2. Therefore, c_1 and c_2 still be satisfied after flipping x. If there is no island trap or after escaping from the island trap, EI flips the best literal *l* from *unsat* in order to move to a better neighbour.

The Experiments

The ESG implementation has the following parameters: –mf : max flips before restarting, -mr: max restarts before aborting, -cp : number of reweights between corrections, -alpha: scaled reweight step size (1+alpha*n/m), -rho: rate of weight shrinkage to mean for SAT clauses, -noise: probability of noise in a trap and -rawalpha: raw reweight step size, (never used with -alpha together). The -cp, -alpha, -rho and -rawalpha are the parameters used to update the LMs for the clauses. We have chosen to set the -alpha parameter instead of -rawalpha because of better performance. In all ESG experiments reported by (Schuurmans and Southey 2000), the -nr flag is used to fix the random number generator seed to 0. We follow the same practice. We ran all the instances on the same machine: a PC with Pentium III 8000 Mhs and 256 memory.

The ESG distribution does not come with any recommended parameters sets. We tuned, with the help of the original authors, the parameter settings for each of the benchmark sets.

The following table gives the parameter sets adopted for ESG and EI. -rho and –mr are equal to 0.99 and 10 respectively for all the instances.

Set	-alpha	-noise	-cp	-rawalpha	-mf
1	0.995	0.02	50	-	500
2	0.999	0.09	300	-	1000
3	1.0	0.03	1000	-	10000000
4	0.999	0.09	500	-	100000
5	0.995	0.02	400	-	7000000
6	0.2401	0.24	400	-	2000000
7	-	0.008	500	1.3	100000000
8	0.995	0.09	300	-	100000000

We compare EI with ESG using the best parameter settings for ESG.

The following two tables gives a comparison of ESG and EI. The first table gives the results of ESG while the second gives the results of EI. We give the P value for EI, the parameter needed when escaping from the island trap.

ESG			
Instance	Succ	Time	Flips
Increasing permutation generation problems set =3			
ap-10	20/20	1.00	104173
ap-20	3/20	5623.22	40057253
Latin Square problems, set =4			
Magic-10	20/20	0.90	699
Magic-15	20/20	0.31	2426
Magic-20	20/20	1.29	5711
Magic-25	20/20	14.08	10655
Magic-30	20/20	16.74	18564
Magic-35	20/20	54.20	38428
Hard graph-coloring problem, set = 5			
g125n-18c	20/20	3.27	19147
g250n-15c	20/20	2.30	2420
g125n-17c	20/20	2494.2	1134850
g250n-29c	20/20	20650.3	22785693
Tight random CSP, set = 6			
rcsp-120-75	20/20	21.7	14965
rcsp-130-75	20/20	24.55	16012
rcsp-140-75	20/20	44.2	17699
rcsp-150-75	20/20	68.04	23576
rcsp-160-75	20/20	83.21	26497
rcsp-170-75	20/20	679.27	165708
Phase transition CSPs, set = 7			
rcsp-120-5.9	Seg-	Ment-	Ation
rcsp-130-5.5		fault	
rcsp-140-5.0	Seg-	Ment-	Ation
rcsp-150-4.7		fault	
rcsp-160-4.4	Seg-	Ment-	Ation
rcsp-170-4.1		fault	

Slightly easier phase transition CSPs, set =8			
rcsp-120-5.8	20/20	1205.2	12844605
rcsp-130-5.4	17/20	24890.2	239966515
rcsp-140-4.9	20/20	196.9	3425882
rcsp-150-4.6	20/20	537.6	7843960
rcsp-160-4.3	20/20	965.3	11813274
rcsp-170-4.0	20/20	221.7	1863285

EI			
Instance	Succ	Time	Flips
Increasing permutation generation problems set =3 and P = 0.3			
ap-10	20/20	0.14	8681
ap-20	20/20	320.27	4213672
Latin Square problems, set = 4, P = 0.8			
Magic-10	20/20	0.02	342
Magic-15	20/20	0.22	973
Magic-20	20/20	0.51	2253
Magic-25	20/20	6.87	4908
Magic-30	20/20	5.2	8874
Magic-35	20/20	20.35	26204
Hard graph-coloring problem, set = 5, P = 0.07			
g125n-18c	20/20	1.98	15385
g250n-15c	20/20	0.51	1196
g125n-17c	20/20	95.28	1549445
g250n-29c	20/20	310.65	1084154
Tight random CSP, set = 6, P =0.7			
rcsp-120-75	20/20	2.17	3699
rcsp-130-75	20/20	1.61	2196
rcsp-140-75	20/20	2.50	3160
rcsp-150-75	20/20	2.16	2301
rcsp-160-75	20/20	2.60	2572
rcsp-170-75	20/20	6.60	7025
Phase transition CSPs, set = 7, P = 0.7			
rcsp-120-5.9	20/20	117.2	2082685
rcsp-130-5.5	20/20	346.28	10592183
rcsp-140-5.0	20/20	89.09	1330117
rcsp-150-4.7	20/20	252.01	1984370
rcsp-160-4.4	20/20	108.9	1073215
rcsp-170-4.1	20/20	140.60	1598530
Slightly easier phase transition CSPs, set =8			
rcsp-120-5.8	20/20	32.7	882129
rcsp-130-5.4	20/20	143.50	5084328
rcsp-140-4.9	20/20	13.55	274665
rcsp-150-4.6	20/20	33.89	673475
rcsp-160-4.3	20/20	18.60	727281
rcsp-170-4.0	20/20	7.50	289210

The tables give the success ratio, average solution time (in seconds) and average flips on solved instances for ESG and EI.

ESG gives segmentation fault while running the phase transition instances.

The advantages of the ICM is evident in improving ESG. EI gives substantial improvement over ESG in term of both time and number of flips in all benchmark instances. In addition, ESG has 17/20 success ratio for one instance in the slightly easier phase transition CSPs while EI could have 20/20 success ratio for the same instance.

Conclusion

We have presented the EI algorithm which is the ICM incorporated into ESG. We have seen the improvement gained by EI. We believe that there is a plenty of scope for using the ICM concept to improve other LS algorithms, such as WalkSAT and the min-conflicts heuristic.

References

Makworth, A. 1977. Consistency in Networks of Relations. *AI* 8(1): 99–118.

Fang, H.; Kilani, Y.; Lee, J.; and Stucky., P. 2002. Reducing Search Space in Local Search for Constraint Satisfaction. *In Proceeding of AAAI*, 200–207. AAAI Press/MIT Press.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. *In Proceeding of AAAI*, 440–446. AAAI Press/MIT Press.

Selman, B., and Kauts, H.; 1993 Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. *In proceeding of IJCAI*, 290–295.

Wu, Z., and Wah, B. 1999 Trap Escaping Strategies in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. *In Proceeding of AAAI*, 673–678. AAAI Press/MIT Press

Wu, Z., and Wah, B. 2000 An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. *In Proceeding of AAAI*, 310–315. AAAI Press/MIT Press.

Minton, S.; Johnston, M.; Philips, A.; and Laird, P. 1992. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *AI* 58: 161–205.

Wu, Z., and Wah, B. 2000. An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. *In Proceeding of AAAI*, 310–315. AAAI Press/MIT Press.

Selman, B.; Kauts, H.; and Cohen, B. 1994. Noise Strategies for Improving Local Search. *In proceeding of AAAI*, 337–343. AAAI Press/MIT Press.

Schuurmans, D., and Southey, F. 2000. Local Search Characteristics of Incomplete SAT Procedures. *In Proceeding of AAAI*, 297–302. AAAI Press/MIT Press.