The ICM in the DLM algorithm

YOUSEF KILANI
Prince Hussein bin Abdullah Information Technology College
Al al-Bayt University
Jordan
and
Abdullah Mohd Zin
Faculty of Information Science and Technology
Universiti Kebangsaan Malaysia
MALAYSIA
y_kilani@yahoo.com and bmz@ftsm.ukm.my

September 4, 2004

Abstract:- Local search methods for solving constraint satisfaction problems such as GSAT, WalkSAT and DLM starts the search for a solution from a random assignment. Local search then examines the neighbours of this assignment, using the penalty function to determine a better neighbour valuations to move to. It repeats this process until it finds a solution which satisfies all constraints.

ICM [8] considers some of the constraints as hard constraints that are always satisfied. In this way, the constraints reduce the possible neighbours in each move and hence the overall search space.

We choose 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 a solution from any valuation in this space.

We show in this paper how incorporating learning in the island traps and restart improves the DLMI algorithm [8].

Keywords: The DLM local search algorithm, SAT problems.

1 Introduction

A *constraint satisfaction problem* (CSP) [3] 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 [8]. A *solution* is an assignment of values from the domains to their respective variables so that all constraints are satisfied simultaneously [8]. CSPs are known to be NP-hard in general.

Local search techniques, for example GSAT [7], WalkSAT [4, 1], DLM [10, 11], the min-conflicts heuristic [2], GENET [5] and ESG [6] have been successful in solving large constraint satisfaction problems [8]. In the context of constraint satisfaction, local search(LS) first generates an initial variable assignment (or state) before making local adjustments (or repairs) to the assignment iteratively until a solution is reached. Local search algorithms can be trapped in a *local minimum (trap)*, a non-solution state in which no further improvement can be made. To help escape from the local minimum, GSAT [7] and the min-conflicts heuristic [2] use random restart, while Davenport *et al.* ([5]), Morris ([9]), DLM and ESG [6] modify the landscape of the search surface.

Local search algorithms traverse the search space to look for solutions using some heuristic function. Schuurmans and Southey [6] in-

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 [8]: (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 [8]. DLMI [8] is the DLM algorithm after incorporating the ICM in it. In this paper, we introduce a new version of DLMI.

The rest of this paper is as follows. Section 2 gives the necessary background and definitions. In this section, we introduce the SAT problems, LS and the DLM algorithm. Section 3 shows the SAT translated from a CSP and the data of the experiments presented in this paper is this kind of SAT. Section 4 presents the notion of the island confinement method. Section 5 presents the DLMI2004 algorithm and the results of this algorithm. Last section gives conclusion remarks.

2 Background and Definitions

In this section, we illustrate some terminologies we use in this paper and we illustrates the DLM algorithm.

Given a CSP (Z, D, 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 $\{x_1, \ldots, x_n\} \subseteq Z$ is a mapping from variables to values denoted $\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$ where $a_i \in D_{x_i}$.

A *state* of a CSP problem (Z, D, C) (or simply C) 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. in the unsatisfied clauses.

2.1 SAT

SAT problems are a special case 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 $\bar{x}$. 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 *satisfiability problem (SAT)* 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.

Since we are dealing with SAT problems we will often treat states as sets of literals. A state $\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$ corresponds to the set of literals $\{x_j \mid a_j = 1\} \cup \{\bar{x}_j \mid a_j = 0\}$.

2.2 Local Search

A LS solver moves from one state to another using a local move. The *neighbourhood* $n(s)$ of a state s is the states that are reachable in a single move from state s. The neighbourhood states are the states reachable in one move from the current state regardless of the actual heuristic function used to choose the neighbour state to move to.

The *Hamming distance* between states s_1 and s_2 is defined as

$$Hd(s_1, s_2) = |s_1 - (s_1 \cap s_2)| = |s_2 - (s_1 \cap s_2)|.$$

It measures the number of differences in variable assignment of s_1 and s_2. A *vector variable* $\vec{x} = (x_1, \ldots, x_n)$.

For the purpose of this paper, we are interested in SAT problems. We assume the neighbourhood function $n(s)$ returns the states which are at a Hamming distance of 1 from the state s. In an abuse of terminology we will also refer to flipping a literal l which simply means flipping the variable occurring in the literal. A *local move* from state s is a transition, $s \Rightarrow s'$, from s to $s' \in n(s)$.

tor of clauses $\vec{c}$ (which we will often also treat as a set). The general LS algorithm starts the search from a random valuation. This valuation represents the current state. Some LS algorithms may start the search from a heuristically chosen valuation. Local search then moves from the current state to a better neighbour. If there is no better neighbour then it is local minima, *trap*. It escapes this trap. Some LS algorithms may include a *restart* and/or *tabu list*. If the search could not find a solution within a number of flips it restarts the search. It uses tabu list to avoid flipping the same variable in the next coming number of steps.

2.3 The DLM Algorithm

DLM [10] is a discrete Lagrange-multiplier-based local-search method for solving SAT problems, which are first transformed into a discrete constrained optimization problem. Experiments confirm that the discrete Lagrange multiplier(LM) method is highly competitive with other SAT solving methods.

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

$$L(s, \vec{\lambda}) = \vec{\lambda} \cdot \vec{c}(s) \quad \text{(that is } \Sigma_i \lambda_i \times c_i(s)\text{)}$$

where $\vec{\lambda}$ are LM, 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 (LM). Figure 1 shows the core of DLM. [1] In this figure, line 1 shows that the input to DLM is a set of clauses $\vec{c}$.

Line 2 makes random initialization to all the variables.

Line 3 initializes $\vec{\lambda}$ to 1.

Line 4 repeats the search until it finds a solution or reaches a maximum number of flips. $L(s, \vec{\lambda}) = 0$ means no constraint is violated, i.e. $\vec{c}(s) = 0$.

[1] Downloadable from `http://www.manip.crhc.uiuc.edu/Wah/programs/SAT_DLM_2000.tar.gz`.

```
2-  let s be a random valuation for var(c⃗)
3-  λ⃗ = 1
4-  while (L(s, λ⃗) > 0 and (max flips is not over))
5-      min := L(s, λ⃗), best := {}
6-      unsat := the literals in unsat clauses
7-      for each literal l ∈ unsat
8-          s' := s − {l} ∪ {l̄}
9-          if (L(s', λ⃗) < min)//a downhill move
10-             min := L(s', λ⃗), best := {s'}, s := s'
11-         else if (((L(s', λ⃗) = min)
                    and (l is not in tabu list))
12-             best := best ∪ {s'}
13-     if (best is empy) then it is trap do learning
14-     else s := s - {var := a randomly
            chosen element from best} ∪ { var̄ }
15-     if (LM update condition holds)
16-         λ⃗ := λ⃗ + c⃗(s)
17-return s
```

Figure 1: DLM (core algorithm)

Lines 5 and 6 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 8-12 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 14 chooses one of the best neighbours and flip it. Lines 15-16 update the lagrangian multipliers according to a parameter.

Although DLM does not appear to examine all the neighbours at Hamming distance 1 in each move, this is an artifact of mixing of the description of neighbourhood and the heuristic functions. Since only literals appearing in unsatisfied clauses (*unsat*) can decrease the Lagrangian function, (the heuristic func-

ignore/discard neighbours resulting from flipping a variable not in one of these literals. The full DLM algorithm also includes many other features, see [11] for details.

3 Encoding CSP as SAT

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

The above 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 $SAT(Z, D, C)$ each clause has the form $\bar{x} \vee \bar{y}$ except for a single clause for each variable in Z.

4 The ICM

The ICM is a generic method which can be incorporated in any local search algorithm. The ICM is based on the observation: the solution space 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 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 starts the search from an assignment which satisfies all these constraints and finally restrict LS to search in this space.

Let $sol(C)$ denotes the set of all solutions to 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 clause c. Let $lit(C) = \cup_{c \in C} lit(c)$. A set C of clauses is *non-conflicting* if there does not exist a variable x such that $x, \bar{x} \in lit(C)$. A non-conflicting set C of clauses forms an island [8]. Therefore, the problem clauses are an island. Given a SAT problem, we can incorporate ICM into any LS algorithm by the following steps: We split the clauses to $\vec{c}_i$ and $\vec{c}_r$, where $\vec{c}_i$ and $\vec{c}_r$ are the island clauses and the at least-one-on clauses respectively. Make an initial valuation that satisfies $\vec{c}_i$; getting inside the island. $\vec{c}_i$ consists of clauses of the form $\bar{x} \vee \bar{y}$. An arbitrary extension of $lit(\vec{c}_i)$ 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, we exclude literals l from flipping when $s' = s - \{l\} \cup \bar{l}$ does not satisfy $\vec{c}_i$. Hence we only examine states that are in $n(s)$ and satisfy $\vec{c}_i$.

5 The ICM in DLM

Figure 2 shows DLMI2004, the ICM incorporated into DLM [11]. DLMI2004 is an enhanced version of DLMI2002 [8]. Table 1 shows the results of DLMI2002 and DLMI2004 for the same instances appeared in [8]. We ran all the instances on the same machine; a PC with Pentium III 800 Mhz and 256 MB memory. There are two new features of DLMI2004: restart and learn. The following is the detail description

uation that gets the search inside the island. Line 6 restarts the search after each $cutoff$ flips, where $cutoff$ is a parameter. Line 9 sets $unsat$ to the set of *free* literals in the unsatisfied clauses so that flipping any of these literals will not violate any island clause. It is an island trap if $unsat$ is empty. Lines 6 and 10 contain the new features of DLMI2004. We do learning in the same way DLM does learning in the DLM traps. line 10 learns when the number of traps reaches a certain limit. If there is no island traps, lines 13-14 make a DLM move. Note that DLM trap never happened and this is because every literal appears only once in the at-least-one-on clauses and flipping this literal will only satisfy the clause in which this literal occurs. In other words, if the literal x is free then flipping x makes a downhill move.

The parameter appeared in table 1 is for DLMI2004. The PS, P and $cutoff$ parameters are one of the five sets of parameters mentioned in [11], the probability used when escaping from the island trap and the cutoff value used before we restart the search respectiely. The PS value is the same for both algorithms DLMI2002 and $DLMI2004$. The table shows the success ratio, average solution time (in seconds) and average flips on solved instances for DLM, DLMI2002 and DLMI2004.

DLMI2004 shows substantial improvement in time and in number of flips over DLMI2002 in increasing permutation generation, latin square, hard graph-coloring and tight random CSP problems. Note that DLMI2004 could solve ap30 instance which $DLMI2002$ could not. The rest of the instances DLMI2004 performs almost the same as DLMI2004.

6 Conclusion

We have presented the DLMI2004 algorithm which is the new improved algorithm over DLMI2002 and we have seen that there is significant improvement in the results of the increasing permutation generation, latin square, hard graph-coloring and tight random csps. We believe there is a plenty of scope for using the ICM concept to improve other LS algorithms, such as WalkSAT, ESG and others.

2- split $\vec{c}$ into $\vec{c_i}$ and $\vec{c_r}$
3- make an initial valuation s that satisfies $\vec{c_i}$
4- $\vec{\lambda} = 1$
5- while ($L(s, \vec{\lambda}) > 0$ and (max flips is not over))
6- restart after each $cutoff$ flips
7- $min := L(s, \vec{\lambda})$, $best := \{\}$
8- $unsat := \cup\{l \mid l \in unsatisfied\ clauses \mid c \in \vec{c_r}, s \notin sol(c)$ and $(s - l \cup \bar{l}) \in sol(c_i)\}$
9- if ($unsat$ is empty) then an island trap
10- learn after $learn$ island traps
11- escape an island trap in the same way mentioned in [8]
12- else lines 8-12 from figure 1
13- $s :=$ choose randomly element from $best$
14- lines 15-16 from figure 1
15- return s

Figure 2: The DLMI algorithm.

References

[1] Selman B.; Kauts H. A. and Cohen B. Noise strategies for improving local search. In *AAAI*, pages 337–343, 1994.

[2] Minton S.; Johnston M.D.; Philips A.B. and Laird P. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. In *AI 58*, pages 161–205, 1992.

[3] Mackworth; A.K. Consistency in networks of relations. In *AI 8(1)*, pages 99–118, 1977.

[4] Selman B. and Kautz H. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *IJCAI*, pages 290–295, 1993.

[5] Davenport A.; Tsang E.; Wang C. and Zhu K. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *AAAI*, pages 325–330, 1994.

[6] Schuurmans D. and Southey F. Local search characteristics of incomplete sat procedures. In *AAAI*, pages 297–302,

Instance	Succ	Time	Flip	Succ	Time	Flip
N queens problem: PS = 2, learn = 100, P = 70 and cuttoff = 2,000						
10queen	20/20	0.00	110	20/20	0.00	95
20queen	20/20	0.01	116	20/20	0.01	120
50queen	20/20	0.12	175	20/20	0.18	194
100queen	20/20	0.88	244	20/20	0.70	199
Random permutation generation problems: PS = 4, learn = 200, P = 70 and cuttoff = 500,000						
pp50	20/20	0.13	204	20/20	0.18	280
pp60	20/20	0.24	308	20/20	0.36	295
pp70	20/20	0.36	323	20/20	0.34	344
pp80	20/20	0.49	308	20/20	0.50	360
pp90	20/20	0.73	311	20/20	0.86	269
pp100	20/20	0.94	269	20/20	1.08	270
Increasing permutation generation problems: PS = 3, learn = 1, P = 70 and cuttoff = 1,000,000						
ap10	20/20	0.03	6,446	20/20	0.03	*2,015*
ap20	20/20	33.39	3,266,368	20/20	14.48	211,031
ap30	20/20	—	—	—	443.35	1,907,253
Latin square problems: PS = 4, learn =16, P = 90 and cuttoff = 1,000,000						
magic-10	20/20	0.02	401	20/20	0.02	315
magic-15	20/20	0.11	1706	20/20	0.08	709
magic-20	20/20	0.52	6824	20/20	0.28	1,473
magic-25	20/20	2.53	25240	20/20	0.87	2,389
magic-30	20/20	60.23	513,093	20/20	2.44	3,845
magic-35	3/20	723.42	3,773,925	20/20	5.29	5,631
Hard graph-coloring problems: PS = 3, learn = 36, P = 85 and cuttoff = 1000000						
g125n-18c	20/20	0.81	15,314	20/20	0.49	8,929
g250n-15c	20/20	0.47	2,815	20/20	7.23	3,608
g125n-17c	20/20	188.61	4,123,124	20/20	40.85	1,099,926
g250n-29c	20/20	128.81	867,396	20/20	79.67	560,737
Tight random CSPs: PS = 4, learn = 16, P = 70 and cuttoff = 3000						
rcsp-120-10-60-75	20/20	1.33	2,919	20/20	0.70	1,179
rcsp-130-10-60-75	20/20	1.30	2,528	20/20	0.89	1,379
rcsp-140-10-60-75	20/20	2.08	3,682	20/20	1.11	1,627
rcsp-150-10-60-75	20/20	1.44	2102	20/20	0.80	7,90
rcsp-160-10-60-75	20/20	2.33	3,306	20/20	1.15	1,196
rcsp-170-10-60-75	20/20	2.56	3,435	20/20	2.31	2,792
Phase transition CSPs: PS = 3, learn = 36, P = 70 and cuttoff = 1,000,000						
rcsp-120-10-60-5.9	19/20	28.71	1,909,746	20/20	27.73	1,734,696
rcsp-130-10-60-5.5	16/20	103.92	6,445,009	20/20	167.02	9,675,405
rcsp-140-10-60-5.0	20/20	14.07	850,886	20/20	16.74	1,012,200
rcsp-150-10-60-4.7	19/20	90.71	5,273,978	20/20	113.36	6,222,070
rcsp-160-10-60-4.4	19/20	31.129	1,695,978	20/20	41.43	2,162,274
rcsp-170-10-60-4.1	19/20	24.17	131,357	20/20	41.18	2,046,899
Slightly easier phase transition CSPs: PS =3, learn = 36, P = 70 and cuttoff = 1,000,000						
rcsp-120-10-60-5.8	18/20	9.61	641,175	20/20	8.54	558,616
rcsp-130-10-60-5.4	19/20	16.82	1,062,060	20/20	21.61	1,313,539
rcsp-140-10-60-4.9	20/20	3.28	195,881	20/20	7.86	453,190
rcsp-150-10-60-4.6	20/20	8.47	499,480	20/20	8.05	460,211
rcsp-160-10-60-4.3	20/20	10.36	574,386	20/20	9.36	485,895
rcsp-170-10-60-4.0	19/20	3.74	197,756	20/20	4.65	227,894

Table 1: Comparative empirical results DLMI2004 versus DLMI2002.

2000.

[7] Selman B.; Levesque H. and Mitchell D.G. A new method for solving hard satisfiability problems. In *AAAI*, pages 440–446, 1992.

[8] Fang H.; kilani Y.; Lee J. and Stucky P. Reducing search space in local search for constriant satisfaction. In *AAAI 2002*, pages 200–207, 2002.

[9] Morris P. The breakout method for escaping from local minima. In *AAAI*, pages 40–45, 1993.

[10] Wu Z. and Wah B.W. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *AAAI*, pages 673–678, 1999.

[11] Wu Z. and Wah B.W. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *AAAI*, pages 310–315, 2000.