Asian Journal of Information Technology 4 (12): 1120-1126, 2005
© Medwell Online, 2005

Improving Dlmi Algorithm by Incorporating New Features

Abdullah Mohdzin and Yousef Kilani
Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia

Abstract: The DLMI Algorithm is a local search algorithm for solving constraint satisfaction problems that incorporates the use of Island confinement method. Local search starts the search for a solution from a random assignment. It 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 that satisfies all constraints. The island confinement method 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 *C* 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 without violating *C*. A previous study has shown that the DLMI algorithm performs better than the original DLM algorithm. In this paper, we describe how incorporating learning in the island traps and restart improves the DLMI algorithm.

Key words: DLM algorithm, DLMI algorithm, local search, SAT problems

INTRODUCTION

Many problems in computing can be modelled as a Constraint Satisfaction Problem (CSP). CSP is a problem defined by a set of variables, each of which can take a number of values from some domain and a set of constraining conditions involving one or more variables[1]. The task is to find a solution which satisfies all these constraining conditions.

There are two methods of search techniques to find such solution: Systematic and nonsystematic. The systematic search method explores the whole search space in a systematic manner by following techniques such as depth-first or breadth-first with respect to particular variables and their values ordering or by maintaining a database that store part of the search space which has been visited and does not contain a solution. Examples of such methods include chronological backtracking[2] and dynamic backtracking[3]. The nonsystematic method came to stochastically search the search space in only a probabilistic sense in attempt to find a solution. Examples of such nonsystematic techniques include genetic algorithms, neural network and the local search algorithms.

Local search algorithms traverse the search space to look for solutions using some heuristic function. These algorithms, for example GSAT[4], WalkSAT[5,6], DLM [7,8], the min-conflicts heuristic[9], GENET[10] and ESG[11] have been successful in solving large constraint satisfaction problems.

There are many examples of CSP problems, such as machine vision[12], scheduling[13], temporal reasoning[14] and circuit design[15]. our interest in this study is in using local search algorithms in solving a special type of CSP known as the Satisfiability Problem (SAT)[16].

In the context of constraint satisfaction, local search 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 and the min-conflicts heuristic use random restart, while GENET, the breakout method, DLM and ESG modify the landscape of the search surface.

The efficiency of a local search algorithm depends on three things:

- The size of the search space (the number of variables and the size of the domain of each variable),
- The search surface (the structure of each constraint and the topology of the cons-traint connection) and
- The heuristic function (the definition of neighbourhood and how a good neigh-bour is picked).

The Island Confinement Method (ICM) aims to reduce the size of the search space[17]. Previous work has successfully incorporate ICM into DLM, which is a local search algorithm. The algorithm, known as DLMI has

Correspomding Author: Abdullah Mohdzin, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Malaysia

been shown to be faster than the original DLM algorithm. This paper, describes a study that has been carried out to improve the original DLMI algorithm by adding new features. The aim of this study is to get a new algorithm, known as DLMI2004, which is better than the original DLMI algorithm.

In this section, we illustrate some terminologies and concepts that are used in this study .

CSP: Given a CSP *(Z, D, C)*, we use *var(c)* to denote the set of variables that occur in constraint c ∈ C. If |*var(c)*| = 2 then *c* is a *binary constraint*. In a *binary CSP*, each constraint c ∈ C is binary. A *valuation* for variable set $\{x_1,\ldots, x_n\} \subseteq Z$ is a mapping from variables to values denoted $\{x_1 \rightarrow a_1,\ldots, x_n \rightarrow a_n\}$ where each x_i is a variable and $a_i \in D_{xi}$, where $a_i \in D_{xi}$. 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.

Solution of a CSP: A state *s* is a *solution* of a CSP *(Z,D,C)* if *s* is a solution to all constraints in *C* simultaneously. An *unsat* is the set of literals occurring in the unsatisfied clauses.

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 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' | l ∈ 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

$\{x_j \mid a_j = 1\} \cup \{x_j' \mid a_j = 0\}$.

Encoding CSP as SAT: In this study, 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:

- 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 ∈ Z, *SAT(Z, D, C)* contains the clause, $x_{a1} \vee \ldots \vee x_{an}$, 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 ∈ *C* with *var(c)* = {*x*, *y*} is mapped to a series of clauses. If {*x* → *a*, y → a"} is not a solution of *c* we add the clause $x'_a \vee y'_{a"}$ to SAT(*Z*, *D*, *C*), where x_a' and $y'_{a"} \in Z$. 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 *x'* ∨ *y'* except for a single clause for each variable in *Z*.

Local search: A local search 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) = $(x_1,\ldots, x_n)$.

The general local search algorithm is given in (Fig .1).

```
1- LS(_c)
2- let s be a random valuation for var(_c)
3- while (solution not found and time not over)
4-    s := b(n(s))
5-    If (there is no such s) then it is local minima
          escape this minima
```

Fig . 1: A general local search algorithm

The local search algorithm starts the search from a random valuation. This valuation represents the current state. Some local search 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 local search 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 the tabu list to avoid flipping the same variable in the next coming number of steps.

Local search for 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* to its neighbour s' is a transition, s _⇒ s', where s' ∈ n(s). We will consider a SAT problem as a vector of clauses vec(c) (which we will often also treat as a set).

The island confinement method: The ICM[17] 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 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 starts the search from an assignment which satisfies all these constraints and finally restrict local search 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*, *x'* ∈ *lit(C)*. A non-conflicting set *C* of clauses forms an island[17]. Therefore, the problem clauses are an island.

Incorporating ICM into LS algorithm: Given a SAT problem, we can incorporate ICM into any local search algorithm by the following steps:

- 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 x' ∨ 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 ∪ l' does not satisfy $vec(c)_i$. Hence we only examine states that are in *n(s)* and satisfy $vec(c)_i$.

The DLM algorithm: DLM[7] 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.

DLM performs a search for a saddle point of the Lagrangian function

$$L(s, vec(\lambda)) = vec(\lambda).vec(c)(s)$$

$$\left(\text{That is } \sum_i \lambda_i .c_i(s)\right)$$

where vec(λ) are LM, one for each constraint, which give the penalty ' for violating that constraint and *c(s)* = *0* if state *s* satisfies constraint *c* andc (s) = 1. The saddle point search changes the state to decrease the Lagrangian function, or increase the (LM).

Fig. 2 shows the core of DLM (Downloadable from: http://www.manip.crhc.uiuc.edu/Wah/-programs/SAT_DLM_2000.tar.gz). The full DLM algorithm also includes many other features,[8] for details.

```
1- DLM(vec(c))
2- let s be a random valuation for var(vec(c))
3- vec(λ) = 1
4- while (L(s, vec(λ)) >0 and (max flips is not over))
5-   min := L(s, vec(λ)), best := {}
6-   unsat := the literals in unsat clauses
7-   for (each literal l ∈ unsat)
8-     s" := s - l ∪ l'
9-     if (L(s", vec(λ) < min) //a downhill move
10-      min := L(s", vec(λ)), best := {s"}, s := s"
11-    else if (((L(s", vec(λ)) = 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} ∪ var'
15-  if (LM update condition holds)
16-    vec(λ) := vec(λ) + vec(c)(s)
17- return s
```

Fig . 2: DLM (the core algorithm)

Line 2 makes random initialization to all the variables. Line 3 initializes *vec(λ)* to 1. Line 4 repeats the search until it finds a solution or reaches a maximum number of flips. L(*s*, *vec(λ)*) = 0 means no constraint is violated, i.e. *vec(c)(s)* = 0.

Lines 5 and 6 set *min* and *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 13 makes *learning*. In learning, DLM increases the lagrangian multipliers of the unsatisfied/all clauses according to parameters.

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

The DLMI Algorithm: The first DLMI algorithm was implemented by incoporating the ICM into the original DLM algorithm[17].

IMPROVING THE DLMI ALGORITHM

To enhance the DLMI algorithm, we have added two new features.

Firstly, the use of cutoff parameter and restart. We restart the search when the number of flips reaches certain limit. This helps us getting the solution faster.

Secondly, the use of learn when the number of island traps reaches certain limit. In learning, we increase the lagrangian multipliers of the unsatisfied clauses. When there is an island trap, we choose the variables to free from the unsatisfied clauses of the higher lagrangian multipliers since the clauses of the highest lagrangian multipliers involved in island traps more than the clauses of a lower lagrangian multipliers. Freeing the variables from the currently unsatisfied clauses of the highest lagrangian multipliers gives more chances for these clauses to be satisfied in the coming flips.

The improved algorithm is shown in (Fig . 3).

The following is the detail description of this algorithm. Line 3 makes an initial valuation that gets the search inside the island. Line 6 restarts the search after each *cutoff* flips, where *cutoff* is a parameter. Line 8 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, line 13 makes 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.

```
1- DLMI(vec(c))
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(λ) = 1
5- while (L(s, vec(λ)) > 0 and (max flips is not over))
6-   restart after each cutoff flips
7-   min := L(s, vec(λ)), best := {}
8-   unsat := ∪ {l | l ∈ unsatisfied clauses |
     c ∈ vec(c)_r, s _∉ sol(c) and (s - l ∪ l') ∈ sol(c)_i}
9-   if (unsat is empty) then an island trap
10-    learn after learn island traps
11-    escape an island trap
12- else
13-    s" := s - l ∪ l'
14-    if (L(s", vec(λ) < min) //a downhill move
15-      min := L(s", vec(λ)), best := {s"}, s := s"
16-   else if (((L(s", vec(λ)) = min)
          and (l is not in tabu list))
17-      best := best ∪ {s"}
18-      s := s - {var := a randomly
           chosen element from best} ∪ var'
19-  if (LM update condition holds)
20-    vec(λ) := vec(λ) + vec(c)(s)
21- return s
```

Fig . 3: The DLMI2004 algorithm.

EVALUATION

To evaluate the effectiveness of the new algorithm, we have conducted an experiment to compare the performances of the DLM algorithm, the DLMI algorithm (now known as DLMI2002) and the new DLMI algorithm (now known as DLMI2004).

Eight problems have been selected for this experiment as shown in Table 1.

Table 1: Problems to be solved

	Problems
P1	Queen Placement Problems
P2	Random permutation generation problems
P3	Increasing permutation generation problems
P4	Latin square problems
P5	Hard graph-coloring problems
P6	Tight random CSPs
P7	Phase transition CSPs
P8	Slightly easier phase transition CSPs P

Encoding the problem into binary CSP: Table 2 shows the instances that are used for experimenting and the number of variables and clauses in each of these instances. These instances have been translated from a binary CSP.

Table 2: Number of Variables and Clauses in each of the Instances.

	Instance	Vars	Clauses
P1	10queen	100	1,480
	20queen	400	12,560
	50queen	2,500	203,400
	100queen	10,000	1,646,800
P2	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
P3	ap10	121	671
	ap20	441	4,641
	ap30	961	14,911
P4	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
P5	g125n-18c	2,250	70,163
	g250n-15c	3,750	233,965
	g125n-17c	2,125	66,272
	g250n-29c	7,250	454,622
P6	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
P7	rcsp-120-10-60-5.9	1,200	25,276
	rcsp-130-10-60-5.5	1,300	27,670
	rcsp-140-10-60-5.0	1,400	29,190
	rcsp-150-10-60-4.7	1,500	31,514
	rcsp-160-10-60-4.4	1,600	33,581
	rcsp-170-10-60-4.1	1,700	35,338
P8	rcsp-120-10-60-5.8	1,200	24,848
	rcsp-130-10-60-5.4	1,300	27,168
	rcsp-140-10-60-4.9	1,400	28,605
	rcsp-150-10-60-4.6	1,500	30,843
	rcsp-160-10-60-4.3	1,600	32,818
	rcsp-170-10-60-4.0	1,200	34,476

The DLM Algorithm: Table 3 shows the result of using the DLM algorithms in solving the problems stated in Table 1. PS is one of a set of five parameters that is used in the DLM algorithm.

The DLMI2002 Algorithm: The result of using this algorithms in solving the problems stated in Table 1 is shown in Table 4. PS and P are two of a set of five parameters that are used in the DLM algorithm.

The DLMI2004 Algorithm: Tables 5 shows the results of using DLMI2004 in solving similar problems. L is the value of the learning parameter, while C is the cutoff value.

Table 3: The results of the DLM algorithm

	DLM			
	Instance	Succ	Time	Flip
P_1	10queen	20/20	0.0	383
	20queen	20/20	0.04	312
	50queen	20/20	4.33	1,251
	100queen	20/20	144.23	4,922
P^2	pp50	20/20	4.75	1,496
PS=4	pp60	20/20	12.75	2,132
	pp70	20/20	28.33	2,876
	pp80	20/20	54.97	3,607
	pp90	20/20	98.87	4,486
	pp100	20/20	164.6	5,378
P_3	ap10	20/20	0.54	38,620
PS=3	ap20	20/20	563.75	14,369,433
	ap30	20/20	-----	-------
P_4	Magic-10	20/20	0.05	899
PS=4	Magic-15	20/20	2.75	3,709
	Magic-20	20/20	24.19	14,218
	Magic-25	20/20	226.76	13,547
	Magic-30	20/20	937.28	72,203
	Magic-35	20/20	3,583.72	169,956
P_5	g125n-18c	20/20	5.06	7,854
PS=3	g250n-15c	20/20	15.96	2,401
	g125n-17c	20/20	146.93	797,845
	g250n-29c	20/20	331.91	334,271
P_6	rcsp-120-..	20/20	9.73	4,857
PS=4	rcsp-130-..	20/20	12.52	5,420
	rcsp-140-..	20/20	16.07	6,125
	rcsp-150-..	20/20	20.21	6,426
	rcsp-160-..	20/20	25.75	7,575
	rcsp-170-..	20/20	28.68	6,760
P_7	rcsp-120-..	20/20	158.03	1,507,786
PS=3	rcsp-130-..	20/20	875.67	7,304,724
	rcsp-140-..	20/20	109.89	888,545
	rcsp-150-..	20/20	613.62	3,966,684
	rcsp-160-..	20/20	382.84	2,244,334
	rcsp-170-..	20/20	293.8	1,383,200
P_8	rcsp-120-..	20/20	47.67	443,665
PS=3	rcsp-130-..	20/20	155.75	1,242,907
	rcsp-140-..	20/20	43.68	319,386
	rcsp-150-...	20/20	60.5	422,370
	rcsp-160-..	20/20	112.58	554,154
	rcsp-170-..	20/20	46.73	244,413

DISCUSSION

We ran all the instances on the same machine; a PC with Pentium III 800 MHz and 256 MB memory. Tables 3, 4 and 5 show the success ratio, average solution time (in seconds) and average flips for all the instances used. The underlined and bold-typed results of success ratio, time and number of flips in Table 5 shows where DLMI2004 gives worse results than DLMI2002 and DLM respectively.

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. In addition, DLMI2004 has 20/20 success ratio for all the instances of phase transition and slightly phase transition

Table 4: The results of the DLMI2002 algorithm

	DLMI2002			
	Instance	Succ	Time	Flip
P_1	10queen	20/20	0.00	110
PS=2	20queen	20/20	0.01	116
P=70	50queen	20/20	0.12	175
	100queen	20/20	0.88	244
P_2	pp50	20/20	0.13	204
PS=4	pp60	20/20	0.24	308
P=70	pp70	20/20	0.36	323
	pp80	20/20	0.49	308
	pp90	20/20	0.73	311
	pp100	20/20	0.94	269
P_3	ap10	20/20	0.03	6,446
PS=3	ap20	20/20	33.39	3,266,368
P=70	ap30	20/20	---	---
P_4	Magic-10	20/20	0.02	401
PS=4	Magic-15	20/20	0.11	1706
P=70	Magic-20	20/20	0.52	6824
	Magic-25	20/20	2.53	25240
	Magic-30	20/20	60.23	513,093
	Magic-35	3/20	723.42	3,773,925
P_5	g125n-18c	20/20	0.81	15,314
PS=3	g250n-15c	20/20	0.47	2,815
P=70	g125n-17c	20/20	188.61	4,123,124
	g250n-29c	20/20	128.81	867,396P6
P_6	rcsp-120-..	20/20	1.33	2,919
PS=4	rcsp-130-..	20/20	1.30	2,528
P=70	rcsp-140-..	20/20	2.08	3,682
	rcsp-150-..	20/20	1.44	2102
	rcsp-160-..	20/20	2.33	3,306
	rcsp-170-..	20/20	2.56	3,435
P_7	rcsp-120-..	19/20	28.71	1,909,746
PS=3	rcsp-130-..	16/20	103.92	6,445,009
P=70	rcsp-140-..	20/20	14.07	850,886
	rcsp-150-..	19/20	90.71	5,273,978
	rcsp-160-..	19/20	31.129	1,695,978
	rcsp-170-..	19/20	24.17	131,357
P_8	rcsp-120-..	18/20	9.61	641,175
PS=3	rcsp-130-..	19/20	16.82	1,062,060
P=70	rcsp-140-..	20/20	3.28	195,881
	rcsp-150-..	20/20	8.47	499,480
	rcsp-160-..	20/20	10.36	574,386
	rcsp-170-..	19/20	3.74	197,758

Table 5: The resukts of the DLMI 2004 Algorithm

	DLMI2004			
	Instance	Succ	Time	Flip
P_1	10queen	20/20	0.00	95
PS=2	20queen	20/20	0.01	120
P=70	50queen	20/20	0.18	194
L=100	100queen	20/20	0.70	199
C=2000				
P_2	pp50	20/20	0.18	280
PS=4	pp60	20/20	0.36	295
P=70	pp70	20/20	0.34	344
L=200	pp80	20/20	0.50	360
C=0.5M	pp90	20/20	0.86	269
	pp100	20/20	1.08	270
P_3	ap10	20/20	0.03	2,015
PS=3	ap20	20/20	14.48	211,031
P=70	ap30	20/20	443.35	1,907,253
L=1				
C=1M				
P_4	magic-10	20/20	0.02	315
PS=4	magic-15	20/20	0.08	709
P=90	magic-20	20/20	0.28	1,473
L=16	magic-25	20/20	0.87	2,389
C=1M	magic-30	20/20	2.44	3,845
	magic-35	20/20	5.29	5,631
P_5	g125n-18c	20/20	0.49	8,929
PS=3	g250n-15c	20/20	7.23	3,608
P=85	g125n-17c	20/20	40.85	1,099,926
L=36	g250n-29c	20/20	79.67	560,737
C=1M				
P_6	rcsp-120-..	20/20	0.70	1,179
PS=4	rcsp-130-..	20/20	0.89	1,379
P=70	rcsp-140-..	20/20	1.11	1,627
L=16	rcsp-150-..	20/20	0.80	790
C=3000	rcsp-160-..	20/20	1.15	1,196
	rcsp-170-..	20/20	2.31	2,792
P_7	rcsp-120-..	20/20	27.73	1,734,696
PS=3	rcsp-130-..	20/20	167.02	9,675,405
P=70	rcsp-140-..	20/20	16.74	1,012,200
L=36	rcsp-150-..	20/20	113.36	6,222,070
C=1M	rcsp-160-..	20/20	41.43	2,162,274
	rcsp-170-..	20/20	41.18	2,046,899
P_8	rcsp-120-..	20/20	8.54	558,616
PS=3	rcsp-130-..	20/20	21.61	1,313,539
P=70	rcsp-140-..	20/20	7.86	453,190
L=36	rcsp-150-..	20/20	8.05	460,211
C=1M	rcsp-160-..	20/20	9.36	485,895
	rcsp-170-..	20/20	4.65	227,894

CSPs while DLMI2002 could not make a 20/20 success ratio for all the instances of these CSPs. Note that DLMI2004 gives slightly worse results in some of the instances. However, the gained improvement for DLMI2004 is more than loss. Therefore, we recommend the use of DLM2004. In the rest of the instances, DLMI2004 performs almost the same as DLMI2004.

If we compare DLMI2004 with DLM, we find that DLM2004 gives substantial improvement in time over DLM in all the instances. But, DLM2004 is still back to DLM in number of flips in some instances. In our opinion, the gained performance in time is more important than the gained performance in number of flips.

Note that the cost of flips in DLM is much more than the cost of flips in DLMI2004. This is because the search space of DLMI2004 is much less than the search space of DLM.

CONCLUSIONS

In this study we have presented the DLMI2004 algorithm which is the improved algorithm over DLMI2002. DLMI2002 is also an improvement over the original DLM algorithm by incorporating the Island Confiement Method.

We have evaluated the new algorithm by comparing it with the original DLM algorithm and DLM2002 in solving eight problems. The result of the study has shown that there is a significant improvement in the performance of the new algorithm when solving the problems such as the increasing permutation generation, latin square, hard graph-coloring and tight random CSPs.

Study are currently being carried out to improve the algorithm further. We are also working on improving other local search algorithms, such as WalkSAT and ESG by incorporating the ICM and other features.

REFERENCES

1. Mackworth, A.K., 1977. Consistency in Networks of Relations, AI J., 8: 99-118.
2. Tsang, E. 1993. Foundations of Constraint Satisfaction.Academic Press.
3. Ginsberg, M. and D. McAllester, 1994. Gsat anddynamic backtracking. In: Fourth Conference On Principle of Knowledge Representation and Reasoning, pp: 226–237.
4. Selman, B., H. Levesque and D.G. Mitchell, 1992. A New Method for Solving hard satisfiability problems, AAAI, pp: 440-446.
5. Selman, B. and H. Kauts, 1993. Domain-independent extensions to GSAT: Solving large strucured satisfiability problems, IJCAI, pp: 290-295.
6. Selman, B., H.A. Kauts and B. Cohen, 1994. Noise Strategies for Improving Local Search, AAAI, pp: 337-343.
7. Wu, Z. and B.W. Wah, 1999. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems, AAAI, pp: 673-678.
8. Wu, Z. and B.W. Wah, 2000. An efficient global Search strategy in discrete lagrangian methods for solving hard satisfiability problems, AAAI, pp: 310-315.
9. Minton, S., M.D. Johnston, A.B. Philips and P. Laird, 1992. Minimizing conflicts: A Heuristic Repair Method For Constraint Satisfaction And Scheduling Problems, AI J., 58: 161-205.
10. Davenport, A., E. Tsang, C. Wang and K. Zhu, 1994. GENET: A connectionist architecture for Solving constraint satisfaction problems by iterative improvement, AAAI, pp: 325-330.
11. Morris, P., 1993. The Breakout Method fo rEscaping From Local Minima, AAAI, pp: 40-45.
12. Chakravarty. I., 1979. A generalized line and junction labelling scheme with applications to scene analysis. In: IEEE Transactions On Patternanalysis And Machine Intelligence 1: 202–205.
13. Fox, M., N. Sadeh and C. Baykan, 1989. Constrained heuristic search. In proceedings of the eleventh international joint conference on artificial intelligence, pp: 309–315.
14. Allen, J., 1983. Maintaining knowledge about temporal intervals. In communications of the ACM, pp: 832–843.
15. de Kleer, I. and G. Sussman, 1980. Propagation of constraints applied to circuit synthesis. In circuit theory and applications pp: 127–144.
16. Selman, B., H. Levesque and D. Mitchell, A new method for solving hard satisfiability problems. In AAAI, pp: 440–446.
17. Fang, H., Y. Kilani, J. Lee and P. Stucky, 2002. Reducing Search Space in Local Search for Constraint Satisfaction, AAAI, pp. 200- 207.