J Heuristics (2007) 13: 557–585
DOI 10.1007/s10732-007-9020-8

The island confinement method for reducing search space in local search methods

H. Fang · Y. Kilani · J.H.M. Lee · P.J. Stuckey

Received: 1 November 2005 / Revised: 18 July 2006 /
Accepted: 29 August 2006 / Published online: 19 April 2007

Abstract Typically local search methods for solving constraint satisfaction problems such as GSAT, WalkSAT, DLM, and ESG treat the problem as an optimisation problem. Each constraint contributes part of a penalty function in assessing trial valuations. Local search examines the neighbours of the current valuation, using the penalty function to determine a "better" neighbour valuation to move to, until finally a solution which satisfies all the constraints is found. In this paper we investigate using some of the constraints as "hard" constraints, that are always satisfied by every trial valuation visited, rather than as part of a penalty function. In this way these constraints reduce the possible neighbours in each move and also the overall search space. The treating of some constraints as hard requires that the space of valuations that are satisfied is "connected" in order to guarantee that a solution can be found from any starting position within the region; thus the concept of islands and the name

A preliminary version of this paper appeared in AAAI'2002.

H. Fang
Department of Computer Science, Yale University, New Haven, USA
e-mail: hai.fang@yale.edu

Y. Kilani · J.H.M. Lee
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

Y. Kilani
e-mail: ykilani@cse.cuhk.edu.hk

J.H.M. Lee
e-mail: jlee@cse.cuhk.edu.hk

P.J. Stuckey (✉)
NICTA Victoria Laboratory, Department of Computer Science and Software Engineering, University of Melbourne, Melbourne, Australia
e-mail: pjs@cs.mu.oz.au

P.J. Stuckey (✉)
e-mail: peter.stuckey@nicta.com.au

"island confinement method" arises. Treating some constraints as hard provides new difficulties for the search mechanism since the search space becomes more jagged, and there are more deep local minima. A new escape strategy is needed. To demonstrate the feasibility and generality of our approach, we show how the island confinement method can be incorporated in, and significantly improve, the search performance of two successful local search procedures, DLM and ESG, on SAT problems arising from binary CSPs.

Keywords Local search · SAT · Constraint satisfaction

1 Introduction

A *constraint satisfaction problem* (CSP) (Mackworth 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. A *solution* is an assignment to each variable of a value in its domain so that all constraints are satisfied simultaneously. CSPs are well-known to be NP-complete in general.

Local search techniques, for example GSAT (Selman et al. 1992), WalkSAT (Selman and Kautz 1993; Selman et al. 1994), Novelty+ (Hoos 1999), the min-conflicts heuristic (Minton et al. 1992), GENET (Davenport et al. 1994), DLM (Wu and Wah 1999, 2000), and ESG (Schuurmans et al. 2001) have been successful in solving large and tight CSPs. 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*, 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 Davenport et al. (1994), Morris (1993), DLM and ESG modify the landscape of the search surface. Following Morris, we call these *breakout methods*. WalkSAT introduces noise into the search procedure to avoid a local minima.

Local search algorithms traverse the search surface of a usually enormous search space to look for solutions using some heuristic function. The time taken to solve a CSP depends on both the problem and the algorithm employed. Four important factors are: (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), (3) the definition of neighbourhood, and (4) the heuristic function (how a "good" neighbour is picked). (1) and (2) are part of the nature of the problem, while (3) and (4) concern the characteristics of particular local search algorithms.

In this paper, we are concerned with (1) and demonstrate that some parts of a search space are guaranteed not to contain any solution, and can be skipped during search. In doing do we also effectively alter (3), restricting the usable parts of the neighbourhood. We propose the *island confinement method*, a general method for modifying local search algorithms for avoiding non-fruitful search regions during search, thus effectively reducing the size of the search space. The method is based on a simple observation: a solution of a CSP P must lie in the intersection of the solution space of all constraints of P. Solving a CSP thus amounts to locating this intersection space, which could be either points or regions scattered around in the entire search space. In addition, the solution space of any subset of constraints in P must enclose all solutions of P. The idea of our method is thus to identify a suitable subset of constraints in P so that the solution space of the subset is "connected," and then restrict our search for solutions to P to this region. By connectedness, we mean the ability to move from one point to any other point within the region by a series of local moves without moving out of the region. Therefore, we are guaranteed that searching within this confined space would not cause us to miss any solution. We call such a region an *island*, and the constraints forming the region *island constraints*. The entire search space is trivially an island but we would like to do better.

In this paper we restrict attention to an important special form of CSP, namely SAT problems resulting from the encodings of binary CSPs. We illustrate a general method for choosing a subset of the clauses which defines an island of connected solutions. We then show how, on encodings of binary CSPs into SAT problems, we can use this method to define an island that incorporates many of the problem clauses. The restriction to search only on the island complicates the search procedure because it may defeat the original traversal strategy of the underlying search procedure. We show how to modify DLM and ESG, both very competitive local search procedures for SAT problems based on subgradient optimisation for Lagrangian relaxation, so that they handle island constraints. The modifications include a redefinition of the neighbourhood function to enforce search remaining on the island. This causes a new type of more difficult local minima, namely island traps, for the search procedures. We propose an escape strategy for island traps, and give empirical results showing where the island confinement method can give substantial improvements in solving some classes of SAT problems.

While ESG benefits entirely from the island confinement method, the island version of DLM exhibits difficulties in traversing a smaller but rougher search surface in a few problem instances. We propose enhancing the island DLM with random restart to improve its ability to maneuver in jagged landscapes. An interesting observation is that random restart, a common technique in local search procedures, is not useful for the original DLM (as results in Sect. 6 will show) but proves to benefit the island DLM. Experiments confirm that the island DLM with random restart is substantially more efficient and robust in terms of success ratios than the original DLM and the island DLM.

This paper significantly extends the original version of this work published in (Fang et al. 2002). The rest of the paper is organised as follows. Section 2 defines CSPs, SATs, and local search before describing the DLM and ESG procedures in detail. In Sect. 3, we give the notion of island constraints formally and illustrate this with examples. We give a sufficient condition for a set of clauses to form an island. Section 4 describes the island confinement method, and defines island traps, followed by an escape strategy for handling island traps in the context of DLM. The escape strategy is similar for ESG. The results are DLMI and ESGI, which are DLM and ESG modified with the island confinement method respectively. In Sect. 6, we first introduce the set of benchmarks adopted for our experiments, and then give and analyse the results for DLMI and ESGI. In Sect. 7, we motivate and suggest extending

DLMI with random restart to make it more efficient and robust. Section 8 summarises our contributions and sheds light on future directions of research.

2 Background and definitions

In the following, we recall common definitions and notations of CSPs and SAT problems. We describe the skeleton of generic local search algorithms, followed by an exposition on two specific local search SAT solvers: DLM (Wu and Wah 1999, 2000) and ESG (Schuurmans et al. 2001).

A *constraint satisfaction problem* (CSP) (Z, D, C) comprises a finite set of variables Z, a domain D assigning a set of possible values D_z to each variable $z \in Z$ and a set of constraints C defined over the variables in Z. 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 constraint c can be considered as the set of valuations over the variable set $var(c)$ which satisfy the constraint.

A *state* of problem $P = (Z, D, C)$ (or simply C) is a valuation for Z. The *search space* of P is the set of all possible states of P. A state s is a *solution* of a constraint c if s restricted to $var(c)$ satisfies c. A state s is a *solution* of a CSP (Z, D, C) if s is a solution to all constraints in C simultaneously.

2.1 SAT

SAT problems are a special form of CSPs. In SAT problems each variable is propositional. A (*propositional*) *variable* can take the value of either 0 (false) or 1 (true). Hence $D_z^{\mathrm{SAT}} = \{0, 1\}$ for all $z \in Z$.

A *literal* is either a variable x or its complement $\bar{x}$ (representing the negation of x, $\neg x$). A literal l is *true* in a valuation s if $l = x$ and $\{x \mapsto 1\} \subseteq s$, or $l = \bar{x}$ and $\{x \mapsto 0\} \subseteq s$. Otherwise l is *false* in s.

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 (which would then be trivial). For SAT problems each constraint $c \in C$ is assumed to be a clause.

A *satisfiability problem* (SAT) consists of a finite set of clauses C. It is a CSP of the form $(var(C), D^{\mathrm{SAT}}, C)$.

Let $\bar{l}$ denote the complement of literal l: $\bar{l} = \bar{x}$ if $l = x$, and $\bar{l} = x$ if $l = \bar{x}$. Let $\bar{L} = \{\bar{l} \mid l \in L\}$ 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 local search solver moves from one state to another. We define the *neighbourhood* $n(s)$ of a state s to be all the states that are reachable in a single move from state s.

```
1- LS($C$)
2-     let $s$ be a random valuation for $var(C)$
3-     while ($s$ is not a solution) do
4-         $s' := b(n(s) \cup \{s\})$
5-         if $s <> s'$ then        %% $s$ and $s'$ are not the same state
6-             $s := s'$
7-         else
8-             invoke a breakout (or restart) procedure
9-     return $s$
```

Fig. 1 A generic local search algorithm

The neighbourhood states are meant to represent all the states reachable in one move, independent of the actual heuristic function used to choose which state to move to. A *local move* from state s is a transition, $s \Rightarrow s'$, from s to $s' \in n(s)$.

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 starting state s. The *Hamming distance* between states s_1 and s_2 is defined as

$$d_h(s_1, s_2) = |s_1 - s_1| = |s_2 - s_1|.$$

In other words, the Hamming distance measures the number of differences in variable assignment of s_1 and s_2.

This neighbourhood reflects the usual kind of local move in SAT solvers: *flipping* a variable. 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 search procedure* consists of at least the following components:

- A neighbourhood function n for all states;
- A heuristic function b that determines the "best" possible local move $s \Rightarrow s'$ for the current state s (note the best possible move may be not necessarily be a "downwards" move in terms of some ranking of states, it might be "sideways" or "upwards"); and
- Possibly an optional restart or breakout procedure to help escape from local minima.

We note that the notion of noise that has appeared in some solvers, such as WalkSAT (Selman and Kautz 1993; Selman et al. 1994), can be incorporated into the heuristic function b. We also decouple the notion of neighbourhood from the heuristic function since they are orthogonal to each other, although they are mixed together in the description of a local move in GSAT, WalkSAT, and other local search algorithms.

The LS procedure in Fig. 1 gives the backbone of a generic local search procedure. Search usually starts from a random state, although greedy heuristics can be employed in generating the initial state. The search goes into a loop until a solution is found. In each iteration, the current state s and its neighbourhood $n(s)$ are examined to locate a preferable state s' using the heuristic function b. It is possible that the preferred move given the neighbourhood $n(s)$ is to stay in the same state s, in which

```
1-  DLM($\vec{C}$)
2-      let $s$ be a random valuation for $var(\vec{C})$
3-      $\vec{\lambda} = 1$
4-      while $(L(s, \vec{\lambda}) > 0)$ do
5-          $min := L(s, \vec{\lambda})$,
6-          $best := \{s\}$
7-          $unsat := \cup\{lit(c) \mid c \in \vec{C}, s \notin sol(\{c\})\}$
8-          for each literal $l \in unsat$
9-              $s' := s - \{\bar{l}\} \cup \{l\}$
10-             if $(L(s', \vec{\lambda}) < min)$ //if so then it is a better downhill move
11-                 $min := L(s', \vec{\lambda})$
12-                 $best := \{s'\}$
13-             else if $(L(s', \vec{\lambda}) = min)$ then
14-                 $best := best \cup \{s'\}$
15-         $s :=$ choose randomly an element from $best$
16-         if (Lagrange multipliers update condition holds)
17-             $\vec{\lambda} := \vec{\lambda} + \vec{C}(s)$
18-     return $s$
```

Fig. 2 The DLM core algorithm

case s' is the same as s and the search is trapped in a local minimum. Some sort of restart or breakout procedure can then be invoked to escape from such a state. The loop might not terminate in theory. In practice, a resource limit is imposed to avoid infinite looping.

2.2.1 The DLM algorithm

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

We will consider a SAT problem as a vector of clauses $\vec{C}$ (which we will often also treat as a set). 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

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

where $\vec{\lambda}$ are Lagrange multipliers, one for each constraint, which give the "penalty" for violating that constraint. The Lagrange multipliers $\vec{\lambda}$ are all initialised to 1. The saddle point search changes the state to decrease the Lagrangian function, or increase the Lagrange multipliers.

Figure 2 gives the core of the DLM algorithm. In essence, the DLM procedure applies a greedy search through the valid assignment space for an assignment that minimises the Lagrangian (a weighted penalty of clause violations) while fixing the

Lagrange multipliers (clause weights), and then penalises the violated clauses by increasing their respective Lagrange multipliers. 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 function of) the DLM algorithm chooses to always ignore/discard neighbours resulting from flipping a variable not in one of these literals.

The Lagrangian $L(s,\vec{\lambda})$ is a non-negative quantity. The main loop terminates when $L(s,\vec{\lambda})$ becomes zero, in which case all constraints are satisfied simultaneously and the current state s is a solution. The full DLM algorithm includes also a tabu list and methods for updating Lagrange multipliers; see Wu and Wah (1999, 2000) for details.

2.2.2 The ESG algorithm

The exponentiated subgradient algorithm (ESG) is a general technique for tackling Boolean linear programs (BLPs). A BLP is a constrained optimisation 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 $\vec{c_1}\cdot\vec{x}\le b_1,\ \ldots,\ \vec{c_m}\cdot\vec{x}\le b_m$ while simultaneously optimising a linear objective $\vec{a}\cdot\vec{x}$ (Schuurmans et al. 2001), where $\vec{a}$ and $\vec{b}$ are constant integer vectors. The following is the canonical form of a *BLP* problem:

$$\min_{\vec{x}\in\{-1,1\}^n} \vec{a}\cdot\vec{x} \quad \text{subject to} \quad C\vec{x}\le\vec{b}.$$

Given a SAT problem with m clauses. Suppose each clause c_i is a disjunction of k_i literals for $1\le i\le m$. Such a SAT problem can be equivalently represented as a BLP as follows:

$$\min_{\vec{x}\in\{-1,1\}^n} \vec{0}\cdot\vec{x} \quad \text{subject to} \quad C\vec{x}\le\vec{k}-\vec{2}$$

where $\vec{0}$ and $\vec{2}$ are vectors of zeros and twos respectively, and

$$C_{ij}=\begin{cases}1 & \text{if } x_j \text{ in } c_i,\\ -1 & \text{if } \bar{x}_j \text{ in } c_i,\\ 0 & \text{otherwise.}\end{cases}$$

An assignment of $+1$ to x_j denotes "true," while that of -1 denotes "false." The idea is to represent clause c_i by a row vector[1] $\vec{c_i}$ in C so that $\vec{c_i}$ has k_i non-zero entries corresponding to the literals in c_i. A constraint is violated only when the $\{-1,+1\}$ assignment to $\vec{x}$ agrees with the coefficients in $\vec{c_i}$ on every non-zero entry, yielding a row sum of exactly k_i. Each disagreement in sign would cause the row sum to drop by 2, thereby making the constraint satisfied. Note the constant zero objective trivialising the problem to a satisfaction problem.

In subsequent presentation of the ESG algorithm, we refer to the standard BLP form. The ESG algorithm is similar to the DLM algorithm in that both are based on subgradient optimisation for Lagrangian relaxation, but with two distinct and subtle

[1] We use c_i to denote a clause in the SAT problem, and $\vec{c_i}$ to denote a row in the matrix C.

modifications. First, ESG uses an augmented Lagrangian by introducing a penalty function θ on constraint violations:

$$L_\theta(\vec{x}, \vec{\lambda}) = \vec{a} \cdot \vec{x} + \sum_{i=1}^{m} \lambda_i \theta(\vec{c_i} \cdot \vec{x} - \vec{b_i})$$

where λ_i is the real-valued Lagrange multiplier associated with constraint c_i. A penalty function of the form $\theta(v) = v$ gives a DLM-style Lagrangian, while ESG adopts the "hinge" penalty function of the form:

$$\theta(v) = \begin{cases} -\frac{1}{2} & \text{if } v \le 0, \\ v - \frac{1}{2} & \text{if } v > 0. \end{cases}$$

Note that v is an integer, and the penalty value can be positive or negative according to the sign of the θ result in the calculation of the augmented Lagrangian.

Second, instead of updating $\vec{\lambda}$ additively as in DLM (described in the last paragraph), ESG updates multiplicatively, which can be understood as following an exponentiated version of the subgradient. Thus, the Lagrange multipliers are *actually* updated by

$$\lambda_j := \lambda_j \alpha^{\theta(\vec{c_j} \cdot \vec{x} - b_j)}.$$

In addition, like DLM (Wu and Wah 2000), the ESG algorithm also employs a weight smoothing technique (Schuurmans and Southey 2000) to prevent Lagrange multipliers from moving away from the population average too quickly. This is especially important for multiplicative updates. ESG also adopts the noise strategy from WalkSAT, which controls how often a random move is used in the primal search phase. Interested readers are referred to Schuurmans et al. (2001) for details of the ESG algorithm.

3 Island constraints

We introduce the notion of island constraints. Central to a local search algorithm is the definition of the neighbourhood of a state since each local move can only be made to a state in the neighbourhood of the current state. We say that a conjunction of constraints is an island if we can move between any two states in the conjunction's solution space using a finite sequence of local moves without moving out of the solution space. The constraints comprising the island are island constraints.

More formally, let *sol*(C) denote 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. Each constraint $c \in C$ is an *island constraint*. We illustrate islands and non-islands in the following examples.

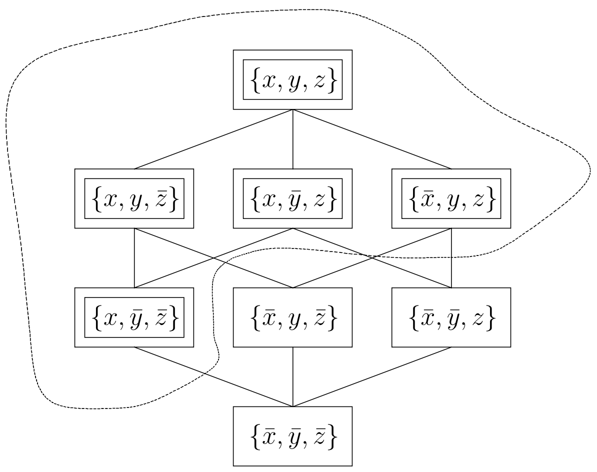
$\{x, y, z\}$
$\{x, y, \bar{z}\}$
$\{x, \bar{y}, z\}$
$\{\bar{x}, y, z\}$
$\{x, \bar{y}, \bar{z}\}$
$\{\bar{x}, y, \bar{z}\}$
$\{\bar{x}, \bar{y}, z\}$
$\{\bar{x}, \bar{y}, \bar{z}\}$

Fig. 3 An example of an island. Each state that satisfies all the constraints has a double border, and neigbouring solution states are surrounded by a *dashed circumference*. All the solution states are joined thus forming a single "island"

Example 1 Consider the SAT problem P_1 with the following three clauses:

$$x \vee y \vee z, \qquad x \vee y \vee \bar{z}, \qquad x \vee \bar{y} \vee z.$$

Figure 3 gives the search space P_1 and its topology. Each box in the diagram denotes a state, labelled by the set of literals corresponding to the state. Solutions are marked with a double border; the rest are non-solution states. Two states are connected by an arc if they are of Hamming distance 1. In other words, the two states can reach each other in one local move. We can verify easily that all solutions of P_1 (enclosed by the dashed circumference) are reachable from one another. Therefore, P_1 is an island.

Example 2 Consider another SAT problem P_2 with the following clauses:

$$\bar{x} \vee y \vee z, \qquad x \vee \bar{y} \vee \bar{z}, \qquad \bar{x} \vee y \vee \bar{z}, \qquad x \vee \bar{y} \vee z.$$

Figure 4 gives the search space of P_2 and its topology using the same convention as in Fig. 3.

The solutions of P_2 are clustered into two separate regions. There is no path (sequence of local moves) from, say, state $\{x, y, \bar{z}\}$ to state $\{\bar{x}, \bar{y}, z\}$ without going through a non-solution state. This is true for any two states, one from each of the two separate regions. Thus P_2 does not form an island.

We give a simple sufficient condition for when a set C of clauses results in an island. 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 $x, \bar{x} \in lit(C)$; otherwise the set is *conflicting*.

Theorem 1 *Assuming the neighbourhood for a SAT local search procedure is defined as the states arising from flipping a single variable, a non-conflicting set C of clauses forms an island.*

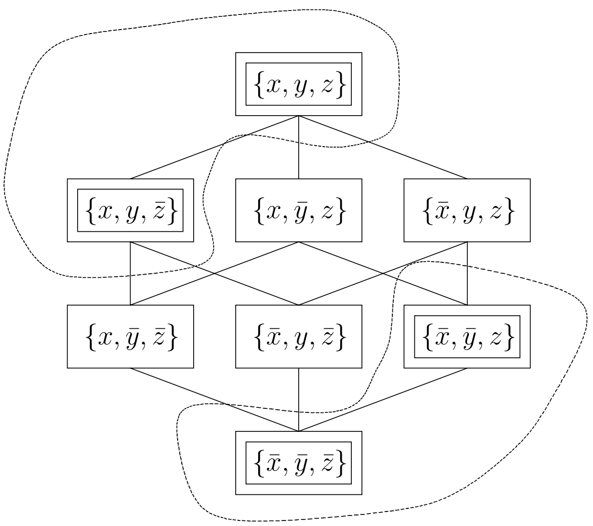
$\{x, y, z\}$
$\{x, y, \bar{z}\}$
$\{x, \bar{y}, z\}$
$\{\bar{x}, y, z\}$
$\{x, \bar{y}, \bar{z}\}$
$\{\bar{x}, y, \bar{z}\}$
$\{\bar{x}, \bar{y}, z\}$
$\{\bar{x}, \bar{y}, \bar{z}\}$

Fig. 4 An example of a non-island. Each state satisfying the constraints has a double border, and neighbouring solution states are surrounded by a *dashed circumference*. There are two separated "islands" of solutions

Proof Since C is non-conflicting $lit(C)$ can be extended to a state (it does not have both a literal and its complement). Any state $s \supseteq lit(C)$ clearly satisfies C. We show that, for any state s_0 satisfying C, there is a path $s_0 \Rightarrow s_1 \Rightarrow \cdots \Rightarrow s_i \Rightarrow \cdots \Rightarrow s_n = s$ where each s_i satisfies C. Since a path is reversible, there is a path between any two solutions s_0 and s'_0 of C via s and hence C is an island.

Let l be an arbitrary literal where s_i and s differ, that is $l \in s_i$ and $\bar{l} \in s$. Then $l \notin lit(C)$ and clearly $s_{i+1} = s_i - \{l\} \cup \{\bar{l}\}$ satisfies C since l does not occur in C and hence cannot be the only literal satisfying one of the clauses of C. □

Note that Theorem 1 gives only a sufficient but not a necessary condition. For instance, the set of clauses in P_1 in Example 1 is conflicting, but yet they form an island.

In the rest of the paper, we focus on a specific class of SAT problems, namely those encoding a CSP. We can map any CSP (Z, D, C) to a SAT problem, $SAT(Z, D, C)$. We illustrate the method for binary CSPs, which we will restrict our attention to, as follows.

- 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 $\{x \mapsto a \wedge y \mapsto a'\}$ is not a solution of c we add the clause $\bar{x}_a \vee \bar{y}_{a'}$ to $SAT(Z, D, C)$. This ensures that the constraint c holds in any solution to the SAT problem. We call these clauses *island clauses*, for reasons that will become clear shortly.

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. The set of all binary *island* clauses trivially forms a non-conflicting set, and hence is an island.

Many other forms of islands clauses are also possible. For example the at-least-one-on clauses also form a non-conflicting set. We can create non-conflicting sets for arbitrary CNF problems by greedy algorithms, and we can create other island clauses which are not based on non-conflicting sets. These islands are not relevant to the rest of the paper, but the interested reader can see (Fang et al. 2006).

4 The island confinement method

Given a CSP $P = (Z, D, C)$. It is easy to see that if $s \in sol(C)$, then $s \in sol(C')$ for all $C' \subseteq C$. By identifying a subproblem of a CSP P as an island, we know that this connected region must contain all solutions of P. Therefore, we can confidently restrict our search to be within only the island and be guaranteed that no solution be missed. Given a CSP, the *island confinement method* consists of collecting as many island constraints as possible, making the island search space as small as possible. We have just shown how to do this easily for the case of $SAT(Z, D, C)$ in the last section. The next step is to locate a random solution on the island, so as to initialise our search. This is also straightforward since the literals in the island clauses of a $SAT(Z, D, C)$ problem are all negative. All we have to do is to first generate a random value for each variable. Then we go through each island clause in turn picking a literal randomly to set to 1 (i.e. setting the variable to 0), if the clause is not already satisfied. This way, the resultant state ensures that all island clauses are satisfied since at least one literal in each clause is true.

What remains is to modify the neighbourhood definition of the local search algorithm so as to enforce search within the solution space of the island constraints. Handling island constraints is simple at first glance. Given a problem defined by a set of clauses $\vec{C}_i \wedge \vec{C}_r$ partitioned into island constraints $\vec{C}_i$ and remaining clauses $\vec{C}_r$, we simply modify the algorithm to treat the remaining clauses as penalty functions and give an initial valuation s which is a solution of $\vec{C}_i$. For $SAT(Z, D, C)$, $\vec{C}_i$ 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. We exclude literals l from flipping when $s' = s - \{\bar{l}\} \cup \{l\}$ does not satisfy $\vec{C}_i$. Hence we only examine states that are adjacent to s and satisfy $\vec{C}_i$. A new neighbourhood function can be defined in terms of the original function $n(s)$ as follows:

$$n(s, \vec{C}_i) = \{s' \in n(s) \mid s' \in sol(\vec{C}_i)\}$$

which takes into account, besides the current state s, also the island constraints $\vec{C}_i$. The rest of the algorithm remains unchanged. *A new problem arises.*

Example 3 Suppose we have the following clauses, where $\vec{C_i} = (c_1, c_2)$ and $\vec{C_r} = (c_3, c_4)$.

$$c_1 : \bar{w} \vee \bar{y}, \qquad c_3 : w \vee x,$$
$$c_2 : \bar{x} \vee \bar{z}, \qquad c_4 : y \vee z$$

and the current state s is $\{w, x, \bar{y}, \bar{z}\}$, which satisfies c_1, c_2, and c_3. The search space of the problem is depicted in Fig. 5. Again, the island states (that satisfy $\vec{C_i}$) are doubly boxed and enclosed by a dashed circumference. Considering states of Hamming distance 1 from s, state s has a total of four neighbours:

$$\begin{cases} s_1 = \{\bar{w}, x, \bar{y}, \bar{z}\} & \text{which satisfies } \{c_1, c_2, c_3\}, \\ s_2 = \{w, \bar{x}, \bar{y}, \bar{z}\} & \text{which satisfies } \{c_1, c_2, c_3\}, \\ s_3 = \{w, x, y, \bar{z}\} & \text{which satisfies } \{c_2, c_3, c_4\}, \\ s_4 = \{w, x, \bar{y}, z\} & \text{which satisfies } \{c_1, c_3, c_4\} \end{cases}$$

but only s_1 and s_2 are on the island, i.e. satisfying c_1, c_2. Thus, states s_3 and s_4 are out of the neighbourhood of s according to the new neighbourhood definition. We can check that none of s_1 and s_2 can bring improvement to the search, since they satisfy exactly the same set of clauses as s. The search is trapped in a local minimum.

Usually in this circumstance a Lagrange multiplier (or equivalently clause weighting) method, would try to update the Lagrange multipliers so as to modify the search landscape, but, in this situation, this is to no avail. This is a new type of local minima for Lagrangian-based local search algorithms. Since s_1, s_2, and s satisfy the same clauses, all three states would have the same Lagrangian value whatever the Lagrange multipliers for c_3 and c_4 are. Hence, *no matter how the Lagrange multipliers are updated, none of s's neighbours will be better than s*. We call this an island trap.

Island traps are manifestation of the island concept. In the following, we give a slightly larger and more complex example to illustrate the difference between ordinary local minima in local search algorithms and island traps.

Example 4 Suppose we have the following clauses, where $\vec{C_i} = (c_1, c_2, c_3)$ and $\vec{C_r} = (c_4, c_5, c_6)$.

$$c_1 : \bar{x}_1 \vee \bar{x}_5, \qquad c_4 : x_1 \vee x_2,$$
$$c_2 : \bar{x}_2 \vee \bar{x}_3, \qquad c_5 : x_3 \vee x_4,$$
$$c_3 : \bar{x}_3 \vee \bar{x}_6, \qquad c_6 : x_5 \vee x_6$$

and the current state s is $\{x_1, \bar{x}_2, x_3, \bar{x}_4, \bar{x}_5, \bar{x}_6\}$, which satisfies c_1 to c_5 inclusively. Without considering island constraints, s has six neighbours:

$$\begin{cases} s_1 = \{\bar{x}_1, \bar{x}_2, x_3, \bar{x}_4, \bar{x}_5, \bar{x}_6\} & \text{which satisfies } \{c_1, c_2, c_3, c_5\}, \\ s_2 = \{x_1, x_2, x_3, \bar{x}_4, \bar{x}_5, \bar{x}_6\} & \text{which satisfies } \{c_1, c_3, c_4, c_5\}, \\ s_3 = \{x_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5, \bar{x}_6\} & \text{which satisfies } \{c_1, c_2, c_3, c_4\}, \\ s_4 = \{x_1, \bar{x}_2, x_3, x_4, \bar{x}_5, \bar{x}_6\} & \text{which satisfies } \{c_1, c_2, c_3, c_4, c_5\}, \\ s_5 = \{x_1, \bar{x}_2, x_3, \bar{x}_4, x_5, \bar{x}_6\} & \text{which satisfies } \{c_2, c_3, c_4, c_5, c_6\}, \quad \text{and} \\ s_6 = \{x_1, \bar{x}_2, x_3, \bar{x}_4, \bar{x}_5, x_6\} & \text{which satisfies } \{c_1, c_2, c_4, c_5, c_6\}. \end{cases}$$

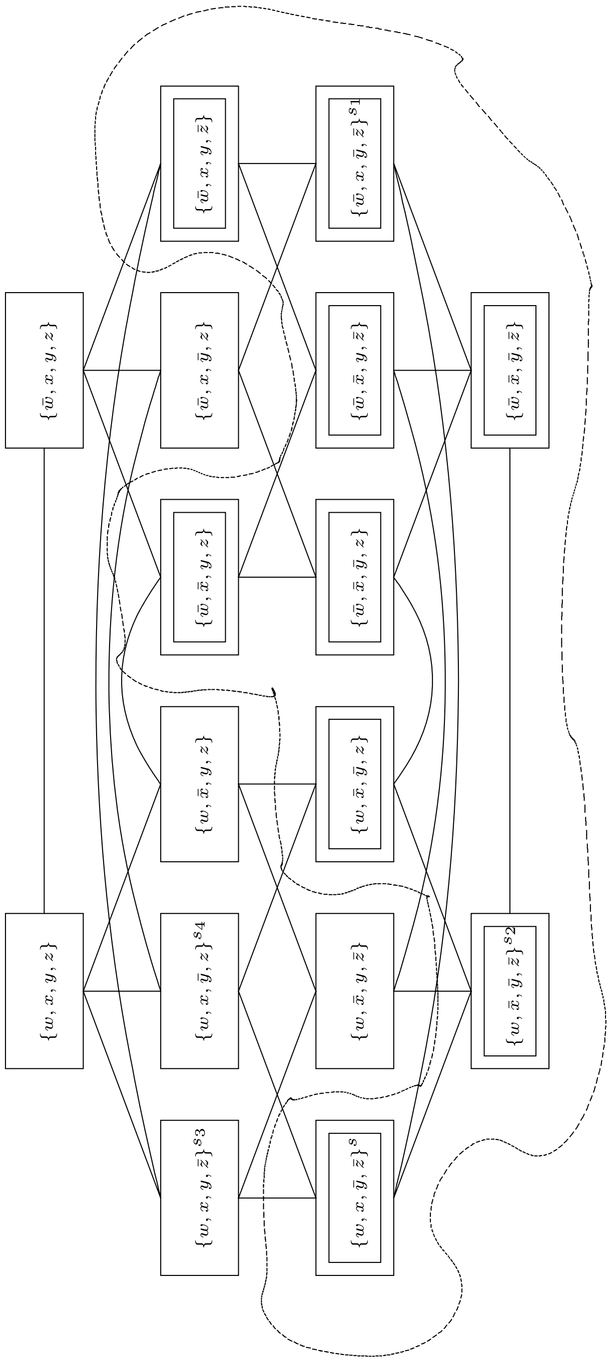
$\{\bar{w}, x, y, z\}$
$\{w, x, y, z\}$
$\{\bar{w}, x, y, \bar{z}\}$
$\{\bar{w}, x, \bar{y}, z\}$
$\{\bar{w}, \bar{x}, y, z\}$
$\{w, \bar{x}, y, z\}$
$\{w, x, \bar{y}, z\}^{s4}$
$\{w, x, y, \bar{z}\}^{s3}$
$\{\bar{w}, x, \bar{y}, \bar{z}\}^{s1}$
$\{\bar{w}, \bar{x}, y, \bar{z}\}$
$\{\bar{w}, \bar{x}, \bar{y}, z\}$
$\{w, \bar{x}, \bar{y}, z\}$
$\{w, \bar{x}, y, \bar{z}\}$
$\{w, x, \bar{y}, \bar{z}\}^{s}$
$\{\bar{w}, \bar{x}, \bar{y}, \bar{z}\}$
$\{w, \bar{x}, \bar{y}, \bar{z}\}^{s2}$

Fig. 5 Illustration of an island trap. The states satisfying the island constraints are *doubly boxed* and enclosed by a *dashed circumference*. The state s has four neighbours s_1, s_2, s_3, and s_4 but only s_1 and s_2 are on the island. Each state s, s_1 and s_2 satisfies the same set of remaining clauses $\{c_3\}$

If a local search algorithm uses only the number of constraint violation (or satisfaction) as the basis of hill-climbing, state s would be a local minimum. States s_1 to s_3 satisfy fewer clauses than s. States s_4 to s_6 are incomparable to s, since they satisfy the same number of clauses although possibly *differing* in some of the clauses that they satisfy. Even with Lagrange multipliers, state s could be a local minimum if the state is encountered at an initial stage of the search (before any of the Lagrange multipliers are changed). However, such a local minimum can be escaped from using either random restart, random walk, or Lagrange multipliers update.

If we take island constraints into account in determining the neighbourhood, however, states s_2 (for violating c_2), s_5 (for violating c_1), and s_6 (for violating c_3) are out of consideration. Among the remaining states, the clauses satisfied by s_1 and s_3 are *strict* subsets of those satisfied by s, while s and s_4 satisfy the same clauses. State s is an island trap.

DLM and ESG are not pure Lagrange multiplier methods, although the Lagrange multiplier is their principle tool for escaping local minima. In that sense they can possibly escape from an island trap. But while Lagrange multiplier updates completely fail in dealing with island traps, island traps also cause difficulties to other local minima escaping mechanisms. Random walk simply picks a state from the current neighbourhood, but the more stringent definition of neighbourhood in the island confinement method reduces the choices of the move. With random walk it will be very difficult to move from one region of the search space to another region if the connecting part of the island is very restricted.

Formally, an *island trap* for a problem $\vec{C}_i \wedge \vec{C}_r$ is a state s such that

$$\{c \in \vec{C}_r \mid s \in sol(\{c\})\} \supseteq \{c \in \vec{C}_r \mid s' \in sol(\{c\})\}$$

for all $s' \in n(s, \vec{C}_i)$. That no (island) neighbour state s' satisfies a constraint not satisfied by s. In the context of Lagrangian-based search algorithms, the condition can be equivalently stated as follows: for all states $s' \in n(s, \vec{C}_i)$ whatever the value of the Lagrange multipliers $\vec{\lambda}_r$ no neighbour would be better than s, i.e.

$$\forall s' \in n(s, \vec{C}_i) \forall (\vec{\lambda}_r > 0) \cdot L(s', \vec{\lambda}_r) \geq L(s, \vec{\lambda}_r).$$

The main difference between an ordinary local minimum and an island trap is that an ordinary local minimum requires $L(s', \vec{\lambda}_r) \geq L(s, \vec{\lambda}_r)$ *only* for the current Lagrange multiplier values.

5 Escaping from island traps in DLM

To incorporate the island confinement method into DLM, we modify DLM's neighbourhood definition to $n(s, \vec{C}_i)$. In the following, we detail an effective escape strategy for island traps. The idea is to flip some variable(s) to make an uphill or flat move(s). We aim to stay as close to the current valuation as possible, but change to a state s' where at least one variable x, which cannot be flipped in the current state s since it would go outside of the island, can now be flipped in s'.

Let

$$makes(l, s, \vec{C_i}) = \{c \in \vec{C_i} \mid (s - \{l\} \cup \{\bar{l}\}) \notin sol(\{c\})\}$$

be the island constraints that are satisfied in the current valuation s only by the literal l. If $makes(l, s, \vec{C_i})$ is non-empty then we cannot flip the literal l in the current state without going outside the island.

We now investigate what we need to do in order to make it possible to flip the literal l. The *freeme set* of literal l in state s is a set of literals, that if all flipped will allow l to be flipped while remaining in the island. More formally, the *freeme set* of literal l in state s, $freeme(l, s, \vec{C_i})$, be a set of literals (a subset of $lit(\vec{C_i})$) such that changing state s by flipping to these literals, arriving at state

$$s' = s - \overline{freeme(l, s, \vec{C_i})} \cup freeme(l, s, \vec{C_i}),$$

allows l to be flipped while staying within the island, i.e. $makes(l, s', \vec{C_i}) = \emptyset$. For the problems we are interested in, it is easy to compute the minimal set since each clause in $\vec{C_i}$ is binary. Hence we must flip to each literal for an island clause currently only satisfied by l.

$$freeme(l, s, \vec{C_i}) = \{l' \mid l \vee l' \in makes(l, s, \vec{C_i})\}.$$

The base island trap escaping strategy we propose is thus: choose the literal l in an unsatisfied clause in $\vec{C_r}$ according to state s such that $|freeme(\bar{l}, s, \vec{C_i})| > 0$ and minimal in size, and flip all literals in $freeme(\bar{l}, s, \vec{C_i})$ and then continue. Note that we do not actually flip the literal l. We only move to a state where l can be flipped. In this state, however, we may find it preferable to flip another literal.

Example 5 Continuing Example 4, we find in state $s = \{x_1, \bar{x}_2, x_3, \bar{x}_4, \bar{x}_5, \bar{x}_6\}$ that the unsatisfied clause is $x_5 \vee x_6$. Now, we have

$$makes(\bar{x}_5, s, \vec{C_i}) = \{c_1\} \quad \text{and} \quad makes(\bar{x}_6, s, \vec{C_i}) = \{c_3\},$$

and hence neither x_5 or x_6 can be flipped without leaving the island. Now

$$freeme(\bar{x}_5, s, \vec{C_i}) = \{\bar{x}_1\} \quad \text{and} \quad freeme(\bar{x}_6, s, \vec{C_i}) = \{\bar{x}_3\}.$$

Suppose we choose randomly to free $\bar{x}_6$, then we can make true all the literals in its freeme set ($\{\bar{x}_3\}$) obtaining the new state

$$s' = \{x_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5, \bar{x}_6\}.$$

We can now flip $\bar{x}_6$ while staying in the island in state

$$s'' = \{x_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5, x_6\}.$$

Flipping x_4 in state s'' leads immediately to a solution

$$\{x_1, \bar{x}_2, \bar{x}_3, x_4, \bar{x}_5, x_6\}.$$

Unfortunately the simple strategy of simply flipping the minimal number of literals to make a currently unflippable literal (since it would go outside the island) flippable is not enough. It is easy for the local search to end up back in the same state, by choosing to reverse all the flips made to escape the trap. In order to prevent this we add an additional tabu list, *tabulit*, of length 1, to cope with the most common case that *freeme* is of size 1. Unlike the regular tabu list, the literal in *tabulit* is not allowed to be flipped under any circumstances (variables in DLM's own tabu list can be flipped if the move is downhill). Occasionally we find difficult traps where the tabu literal is not enough to prevent falling back into the same trap. To avoid this we add a parameter P which gives the probability of picking a literal to free which requires more than the minimal number of flips to free.

The DLM algorithm modified for islands (DLMI) is shown in Fig. 6. Lines beginning in "|" are either different from their counterparts in the original DLM algorithm or new additions. For DLMI there are only Lagrange multipliers $\vec{\lambda}_r$ for the non-island clauses $\vec{C}_r$. A random valuation that satisfies the island clauses $\vec{C}_i$ is chosen (since $\vec{C}_i$ is non-conflicting this is straightforward). The candidate literals for flipping are restricted to those that maintain satisfiability of the island clauses and are not the literal in *tabulit*. If there are candidates then we proceed as in DLM; otherwise we are in an island trap. Note that *tabulit* introduces another kind of island trap where no flip will satisfy more clauses except flipping the literal in *tabulit*, which is disallowed. This trap is handled identically to the original island trap.

In an island trap we consider the literals (*free*) in the unsatisfied clauses which could not be flipped without breaking an island constraint. Note that $free \neq \emptyset$ otherwise we have a solution. We separate these into those requiring 1 other literal to be flipped to free them ($free_1$), and those requiring two or more ($free_{2+}$). If the random number is less than parameter P we choose a literal in $free_{2+}$ to free, and flip all the variables required to free it. Otherwise we choose, if possible, a variable in $free_1$ whose *freeme* is not the literal in *tabulit* and flip the literal in that set.

Note that in both cases, the selection of l, the literal to free, may fail. In the first case when $free_{2+}$ is empty, in which case we perform nothing relying on randomness to eventually choose the other case.

In the second case it may be that *every* literal in $free_1$ has its freeme set equal to *tabulit*. If $free_{2+}$ is non-empty, then we perform nothing relying again on randomness to eventually choose to work on $free_{2+}$. Otherwise, $free_{2+}$ is empty, and we have detected that *tabulit* must hold in any solution of $\vec{C}_i \wedge \vec{C}_r$, as stated in the following lemma.

Theorem 2 *Given a* $SAT(Z, D, C) = \vec{C}_i \wedge \vec{C}_r$ *where all clauses in* $\vec{C}_i$ *are binary and all literals in* $\vec{C}_i$ *are negative. If* $free_{2+} = \emptyset$ *and* $freeme(\bar{l}, s, \vec{C}_i) = tabulit = \{l_t\}$ *for all* $l \in free_1$ *in an island trap state* s *in the DLMI algorithm* (*Fig.* 6), *then* l_t *must hold true in any solution of* $\vec{C}_i \wedge \vec{C}_r$.

Proof Recall that clauses and literals in $\vec{C}_i$ are all binary and negative respectively, and literals in $\vec{C}_r$ are all positive. Note that l_t must be a negative literal since (1) l is in $\vec{C}_r$ and (2) l_t is in the freeme set of a negative literal $\bar{l}$.

There are two possibilities to be in an island trap in DLMI. First, *candidate* = *tabulit*, in which case l_t is positive and this is a contradiction. Second, *candidate* = $\emptyset$,

```
1-     DLMI($\vec{C_i}$,$\vec{C_r}$)
2-  |    let $s \in sol(\vec{C_i})$ be a random valuation for $var(\vec{C_i} \cup \vec{C_r})$
3-       $\vec{\lambda_r} = 1$ %% a vector of 1s
4-  |    $tabulit := \emptyset$
5-       while $(L(s, \vec{\lambda_r}) > 0)$ do
6-           $unsat := \cup\{lit(c) \mid c \in \vec{C_r}, s \notin sol(\{c\})\}$
7-  |        $candidate := \{l \in unsat \mid (s - \{\bar{l}\} \cup \{l\}) \in sol(\vec{C_i})\}$
8-  |        if $(candidate - tabulit \neq \emptyset)$ then %% not an island trap
9-               $min := L(s, \vec{\lambda_r})$
10-              $best := \{s\}$
11- |            $s_{old} := s$
12- |            foreach literal $l \in candidate - tabulit$
13-                  $s' := s - \{\bar{l}\} \cup \{l\}$
14-                  if $(L(s', \vec{\lambda_r}) < min)$ then
15-                      $min := L(s', \vec{\lambda_r})$
16-                      $best := \{s'\}$
17-                  else if $(L(s', \vec{\lambda_r}) = min)$ then
18-                      $best := best \cup \{s'\}$
19-              $s :=$ a randomly chosen element of $best$
20- |            $tabulit := (s = s_{old}\ ?\ tabulit : s_{old} - s)$ %% a singleton set
21- |        else %% island trap
22- |            $free := unsat - candidate$
23- |            $free_1 := \{l \in free \mid |freeme(\bar{l}, s, \vec{C_i})| = 1\}$
24- |            $free_{2+} := free - free_1$
25- |            $r :=$ random number between 0 and 1
26- |            if $(free_1 = \emptyset$ or $r < P)$ then %% free arbitrary literal
27- |                $l :=$ a randomly chosen element of $free_{2+}$
28- |                $s := s - \overline{freeme(\bar{l}, s, \vec{C_i})} \cup freeme(\bar{l}, s, \vec{C_i})$
29- |                $tabulit := \emptyset$
30- |            else if $(free_{2+} = \emptyset$ and $\forall l \in free_1 : freeme(\bar{l}, s, \vec{C_i}) = tabulit)$ then
    |                %% fixed value detected
31- |                fix the value of the variable in $tabulit$
32- |            else %% free literal requiring single flip
33- |                $l :=$ a randomly chosen element of $free_1$
    |                        where $freeme(\bar{l}, s, \vec{C_i}) \neq tabulit$
34- |                $s := s - \overline{freeme(\bar{l}, s, \vec{C_i})} \cup freeme(\bar{l}, s, \vec{C_i})$
35- |                $tabulit := \overline{freeme(\bar{l}, s, \vec{C_i})}$
36-          if (Lagrange multipliers update condition holds) then
37-              $\vec{\lambda_r} := \vec{\lambda_r} + \vec{C_r}(s)$
38-      return $s$
```

Fig. 6 The DLMI core algorithm

in which case both $free = free_1$ contains all literals in the unsatisfied clauses. Suppose one of the unsatisfied clauses is:

$$x_1 \vee \cdots \vee x_n$$

where $\{x_1, \ldots, x_n\} \subseteq free_1$. In this case, $\vec{C}_i$ must include the following clauses:

$$\bar{x}_1 \vee l_t \quad \cdots \quad \bar{x}_n \vee l_t.$$

By resolution, we can conclude that l_t must be true in any solution of $\vec{C}_r \wedge \vec{C}_i$. □

We are then justified to eliminate the variable in *tabulit* by unit resolution. In our code this unit resolution is performed dynamically at runtime. We could avoid this by simplifying the original SAT formulation so that all such occurrences are removed, using SAT simplification methods such as (Brafman 2001).

Example 6 Modifying clause c_3 in Example 4 slightly.

$$\begin{aligned} c_1 &: \bar{x}_1 \vee \bar{x}_5, & c_4 &: x_1 \vee x_2, \\ c_2 &: \bar{x}_2 \vee \bar{x}_3, & c_5 &: x_3 \vee x_4, \\ c_3 &: \bar{x}_1 \vee \bar{x}_6, & c_6 &: x_5 \vee x_6. \end{aligned}$$

We are in an island trap state $s = \{x_1, \bar{x}_2, x_3, \bar{x}_4, \bar{x}_5, \bar{x}_6\}$ and assume that *tabulit* is $\{\bar{x}_1\}$. The literals in unsatisfied clauses are $unsat = \{x_5, x_6\}$, and $candidate = \emptyset$ since neither literal can be flipped. Hence $free = \{x_5, x_6\}$. Both of these literals are placed in $free_1$, since $freeme(\bar{x}_5, s, \vec{C}_i) = freeme(\bar{x}_6, s, \vec{C}_i) = \{\bar{x}_1\}$. The selection of a literal l in $free_1$ will fail. Since $free_{2+} = \emptyset$, $\{\bar{x}_1\}$ must hold in any solution of $\vec{C}_i \wedge \vec{C}_r$ by Theorem 2. Applying resolution on the clauses

$$x_5 \vee x_6, \qquad \bar{x}_1 \vee \bar{x}_5, \qquad \bar{x}_1 \vee \bar{x}_6,$$

we obtain $\bar{x}_1$.

In the context of CSP, x_1 corresponds to a value in the domain of a CSP variable (say u) which is incompatible with all (two in this case) values in the domain of the other CSP variable (say v). That means that the domain value of u corresponding to x_1 is arc inconsistent with respect to the constraint involving u and v. Fixing x_1 to 0 means removing the value from the domain of u.

The feature just described in Example 6 is closely related to the lazy arc consistency technique (Stuckey and Tam 1998) developed for GENET (Davenport et al. 1994) and later adapted to $\mathcal{LSDL}$ (Choi et al. 2000), both of which are CSP local search solvers. An advantage of this technique is that it can detect unsatisfiability of the problem occasionally when the propositional variables, say $\{x_1, \ldots, x_n\}$, corresponding to all values in the domain of a CSP variable, say u, are fixed to 0, thereby making the non-island clause $x_1 \vee \cdots \vee x_n$ false.

Since ESG is also Lagrangian-based, the island confinement method can also be incorporated into ESG using the same neighbourhood function $n(s, \vec{C}_i)$ and a similar island trap escape strategy.

Table 1 Benchmark size

Instance	Vars	Cls	Instance	Vars	Cls
10-queens	100	1, 480	20-queens	400	12, 560
50-queens	2, 500	203, 400	100-queens	10, 000	1, 646, 800
pp-50	2, 475	159, 138	pp-60	3, 568	279, 305
pp-70	4, 869	456, 129	pp-80	6, 356	660, 659
pp-90	8, 059	938, 837	pp-100	9, 953	1, 265, 776
ap10	121	671	ap20	441	4, 641
ap30	961	14, 911	ap40	1, 681	34, 481
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
g125n-18c	2, 250	70, 163	g250n-15c	3, 750	233, 965
g125n-17c	2, 125	66, 272	g250n-29c	7, 250	454, 622
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
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
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, 700	34, 476

6 Experiments

To demonstrate the feasibility and efficiency of our proposal, we implement DLMI and ESGI by making minimal modifications to the code distributions of SAT-DLM-2000[2] and ESG-SAT[3] respectively, maintaining all the extra parts such as the tabu list, and penalty updating methods unchanged.

We adopt the suite of binary CSPs used by Choi et al. (2000) of different nature and difficulties, in our experiments. The problems include N-queens (n-queens), random permutation generation (pp-n), increasing permutation generation (ap-n), Latin square (magic-n), hard graph coloring (gxn-yc), and random CSPs (rcsp-n-d-p_1-p_2). The last class is further divided into three groups, namely tight, phase transition, and slightly easier phase transition random CSPs. We first transform the problem instances into SAT. Of the clauses in all instances, over 99% are island clauses. Table 1 lists the size of the benchmarks in terms of the number of variables and clauses when

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

[3]Downloadable from http://ai.uwaterloo.ca/~dale/software/esgsat.tar.gz.

Table 2 Comparative empirical results DLM versus DLMI

Instance	DLM			DLMI			
	Succ	Time	Flips	Succ	Time	D-Flips	I-Flips
$PS = 2$ and $P = 0.3$ for DLMI							
10-queens	20/20	0.01	186	20/20	0.00	39	29
20-queens	20/20	0.02	265	20/20	0.00	69	49
50-queens	20/20	1.20	1417	20/20	0.06	89	39
100-queens	20/20	88.11	5455	20/20	0.69	176	76
$PS = 4$ and $P = 0.3$ for DLMI							
pp-50	20/20	1.16	1451	20/20	0.08	115	65
pp-60	20/20	3.34	2113	20/20	0.17	145	85
pp-70	20/20	10.28	2868	20/20	0.30	242	172
pp-80	20/20	25.42	3554	20/20	0.41	193	113
pp-90	20/20	52.31	4373	20/20	0.58	202	112
pp-100	20/20	99.41	5333	20/20	0.76	200	100
$PS = 3$ and $P = 0.3$ for DLMI							
ap-10	20/20	0.25	23921	20/20	0.02	2822	2811
ap-20	20/20	424.75	15953036	20/20	30.81	1872874	1872853
ap-30	0/20	–	–	0/20	–	–	–
ap-40	0/20	–	–	0/20	–	–	–
$PS = 4$ and $P = 0.1$ for DLMI							
magic-10	20/20	0.03	779	20/20	0.01	174	74
magic-15	20/20	0.60	3462	20/20	0.05	428	203
magic-20	20/20	5.56	12278	20/20	0.21	886	486
magic-25	*	*	*	20/20	0.68	1443	818
magic-30	*	*	*	20/20	1.53	1992	1092
magic-35	*	*	*	20/20	3.21	3258	2033

encoded in the SAT formulation. Experiments on DLM and DLMI are conducted on a Sun Blade 1000(2 × 900 MHz US-III+) workstation with 2GB of memory running the Solaris 8 OS, and those for ESG and ESGI are conducted on a PC with a PIII 800 Mhz CPU and 256MB memory running the Linux OS. Timing and flipping results are average of 20 runs. We abort runs which fail to find solutions after 60,000,000 flips.

6.1 DLM

For each benchmark set, we first tune the best parameter settings for DLM of the five (Wu and Wah 2000) included in the distribution. These same parameter settings are adopted for DLMI. For the additional parameter P introduced by the island method, we tune and report the best P value for each type of benchmark instances. Tables 2 and 3 show a comparison of DLM and DLMI.

Table 3 Comparative empirical results DLM versus DLMI *(cont'd)*

Instance	DLM			DLMI			
	Succ	Time	Flips	Succ	Time	D-Flips	I-Flips
$PS = 3$ and $P = 0.15$ for DLMI							
g125n-18c	20/20	1.44	7519	20/20	0.23	5218	5093
g250n-15c	20/20	4.57	2287	20/20	2.03	24082	23832
g125n-17c	20/20	54.77	713542	20/20	21.79	585556	585431
g250n-29c	20/20	212.91	425284	20/20	50.07	253478	253228
$PS = 4$ and $P = 0.3$ for DLMI							
rcsp-120-10-60-75	20/20	3.99	4813	20/20	0.57	1055	935
rcsp-130-10-60-75	20/20	5.86	5736	20/20	0.66	1033	903
rcsp-140-10-60-75	20/20	7.73	6073	20/20	1.33	2139	1999
rcsp-150-10-60-75	20/20	9.06	6434	20/20	1.16	1568	1418
rcsp-160-10-60-75	20/20	12.98	7391	20/20	0.89	913	753
rcsp-170-10-60-75	20/20	15.46	7031	20/20	1.73	1995	1825
$PS = 3$ and $P = 0.3$ for DLMI							
rcsp-120-10-60-5.9	20/20	66.22	1066997	**15/20**	18.22	931333	931213
rcsp-130-10-60-5.5	20/20	562.77	7324350	**15/20**	82.56	4059895	4059765
rcsp-140-10-60-5.0	20/20	71.81	924185	**19/20**	8.16	412126	411986
rcsp-150-10-60-4.7	20/20	448.84	6010714	**19/20**	70.38	3388199	3388049
rcsp-160-10-60-4.4	20/20	376.73	3974725	20/20	23.08	1108738	1108578
rcsp-170-10-60-4.1	20/20	131.87	1339107	20/20	11.85	556487	556317
$PS = 3$ and $P = 0.3$ for DLMI							
rcsp-120-10-60-5.8	20/20	25.72	423898	20/20	5.37	280520	280400
rcsp-130-10-60-5.4	20/20	64.41	912136	20/20	12.71	634863	634733
rcsp-140-10-60-4.9	20/20	17.03	233552	20/20	3.04	152091	151951
rcsp-150-10-60-4.6	20/20	25.99	329755	**19/20**	5.46	270458	270308
rcsp-160-10-60-4.3	20/20	55.83	590711	20/20	8.03	390389	390229
rcsp-170-10-60-4.0	20/20	14.50	163382	20/20	2.30	110351	110181

For each set of benchmark instances, we give the parameter settings (PS) from SAT-DLM-2000 used for DLM and also DLMI. The tables give the success ratio, average solution time (in seconds) and average flips on solved instances for DLM and DLMI. There are two types of DLMI flips: D-Flips and I-Flips. D-Flips are normal DLM flips, and I-Flips are flips used for escaping from island traps. I-Flips are considerably cheaper since they do not require any computation of the Lagrangian function values. Entries marked "–" and "*" indicate no applicable data available and segmentation fault during execution respectively. Bold entries show when DLM betters DLMI.

DLMI shows substantial improvement over DLM using the same parameter sets on the test suite, and is able to solve all magic-* instances. Generally DLMI traverses a smaller search space and needs to do less maintenance for island clauses. This results in significant saving. In many cases DLMI is one to two orders of magnitude

Table 4 Tuned parameter sets for ESG and ESGI

PS	`-rho`	`-alpha`	`-noise`	`-cp`	`-mr`	`-mf`
1	0.99	0.995	0.02	50	10	500
2	0.99	0.999	0.09	300	10	10000
3	0.999	1.0	0.03	1000	10	10000000
4	0.999	0.999	0.09	500	10	100000
5	0.9995	0.9995	0.02	400	10	7000000
6	0.999	0.2401	0.24	400	10	2000000
7	0.999	1.3*	0.008	500	10	100000000
8	0.999	0.9995	0.09	300	10	100000000

better than DLM. DLMI is slightly less robust in success rate with the two classes of phase transition random CSPs. This occurs because the search surface is now considerably more jagged. In the next section, we give a simple modification to DLMI to smooth the search behavior.

6.2 ESG

The ESG implementation has the following parameters for tuning the solver behavior.

- `-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 random walk when stuck
- `-rawalpha`: raw reweight step size (never used with `-alpha` together)

We have chosen to set the `-alpha` parameter instead of `-rawalpha`, except for the phase transition random CSPs class of benchmarks which responds better to the `-rawalpha` parameter. In all ESG experiments reported (Schuurmans et al. 2001), the `-nr` flag is used to fix the random number generator seed to 0. We adopt the same practice.

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. Table 4 give the parameter sets adopted for ESG and ESGI accordingly. In parameter set $PS = 7$, the parameter for `-alpha` is actually for `-rawalpha` for the phase transition random CSPs. For the additional island confinement method parameter P, we use $P = 0.2$ for ESGI in the first part (easier instances) of the benchmark sets and $P = 0.3$ for the second part (harder instances).

Tables 5 and 6 give a comparison of ESG and ESGI. For each set of benchmark instances, we give the P value for ESGI. The tables give the success ratio, average solution time (in seconds) and average flips on solved instances for ESG and ESGI. Again, we differentiate between E-Flips and I-Flips in ESGI, where E-Flips are normal ESG flips and I-Flips are ones for escaping from island traps. I-Flips are much

Table 5 Comparative empirical results ESG versus ESGI

Instance	ESG			ESGI			
	Succ	Time	Flips	Succ	Time	E-Flips	I-Flips
$PS = 1$ and $P = 0.2$ for ESGI							
10-queens	20/20	0.02	235	20/20	0.00	46	35
20-queens	20/20	0.04	317	20/20	0.03	68	45
50-queens	20/20	1.03	1424	20/20	0.90	116	63
100-queens	20/20	15.23	7523	20/20	4.80	183	81
$PS = 2$ and $P = 0.2$ for ESGI							
pp-50	20/20	0.83	2198	20/20	0.08	144	84
pp-60	20/20	1.39	2580	20/20	1.04	252	170
pp-70	20/20	5.22	6099	20/20	1.23	266	174
pp-80	20/20	5.10	4956	20/20	1.67	283	178
pp-90	20/20	4.23	5632	20/20	2.55	273	161
pp-100	20/20	9.25	7283	20/20	2.88	303	179
$PS = 3$ and $P = 0.2$ for ESGI							
ap-10	20/20	1.00	104173	20/20	0.14	4249	4432
ap-20	3/20	5623.22	40057253	20/20	320.27	3111245	1102427
ap-30	0/20	–	–	0/20	–	–	–
ap-40	0/20	–	–	0/20	–	–	–
$PS = 4$ and $P = 0.2$ for ESGI							
magic-10	20/20	0.90	699	20/20	0.02	231	111
magic-15	20/20	0.31	2426	20/20	0.22	627	346
magic-20	20/20	1.29	5711	20/20	0.51	1398	855
magic-25	20/20	14.08	10655	20/20	6.87	2926	1982
magic-30	20/20	16.74	18564	20/20	5.20	5185	3689
magic-35	20/20	54.20	38428	20/20	20.35	14654	11550

cheaper than E-Flips. Entries marked "–" and "*" indicate no applicable data available and segmentation fault during execution respectively.

The advantages of the island confinement method are more evident in improving ESG. Unlike the case of DLMI over DLM in which DLMI exhibits difficulties with a few hard instances, ESGI gives substantial and consistent improvement over ESG in terms of both time and number of flips (even when both regular and island flips are taken into account) in all benchmark instances. Again the time improvement can be up to two orders of magnitude.

The island confinement method reduces the size of the search space by limiting the choices that a search can make at every move. In order not to get out of the island, the search might have to choose a different route from one that would normally be recommended by the search heuristic, possibly causing detours and/or cycling. We can observe that the advantages of smaller search space are sometimes offset by the need to traverse a more rugged landscape in DLMI, but this is not the case for ESGI.

Table 6 Comparative empirical results ESG versus ESGI *(cont'd)*

Instance	ESG			ESGI			
	Succ	Time	Flips	Succ	Time	E-Flips	I-Flips
$PS = 5$ and $P = 0.3$ for ESGI							
g125n-18c	20/20	3.27	19147	20/20	1.98	7860	7525
g250n-15c	20/20	2.30	2420	20/20	0.51	514	682
g125n-17c	20/20	2494.20	1134850	20/20	95.28	785856	763589
g250n-29c	20/20	20650.33	22785693	20/20	310.65	549986	534168
$PS = 6$ and $P = 0.3$ for ESGI							
rcsp-120-10-60-75	20/20	21.70	14965	20/20	2.17	2154	1545
rcsp-130-10-60-75	20/20	24.55	16012	20/20	1.61	1299	897
rcsp-140-10-60-75	20/20	44.20	17699	20/20	2.50	1859	1301
rcsp-150-10-60-75	20/20	68.04	23576	20/20	2.16	1371	930
rcsp-160-10-60-75	20/20	83.21	26497	20/20	2.60	1526	1046
rcsp-170-10-60-75	20/20	679.27	165708	20/20	6.60	4059	2966
$PS = 7$ and $P = 0.3$ for ESGI							
rcsp-120-10-60-5.9	*	*	*	20/20	117.20	1047043	1035642
rcsp-130-10-60-5.5	*	*	*	20/20	346.28	5633445	4958738
rcsp-140-10-60-5.0	*	*	*	20/20	89.09	668743	661374
rcsp-150-10-60-4.7	*	*	*	20/20	252.01	997737	986633
rcsp-160-10-60-4.4	*	*	*	20/20	108.90	552472	520743
rcsp-170-10-60-4.1	*	*	*	20/20	140.60	803731	794799
$PS = 8$ and $P = 0.3$ for ESGI							
rcsp-120-10-60-5.8	20/20	1205.20	12844605	20/20	32.70	469061	413068
rcsp-130-10-60-5.4	17/20	24890.22	239966515	20/20	143.50	2704314	2380014
rcsp-140-10-60-4.9	20/20	196.90	3425882	20/20	13.55	146103	128562
rcsp-150-10-60-4.6	20/20	537.60	7843960	20/20	33.89	358203	315272
rcsp-160-10-60-4.3	20/20	965.30	11813274	20/20	18.60	393059	334222
rcsp-170-10-60-4.0	20/20	221.70	1863285	20/20	7.50	153860	135350

Random restart is a simple and efficient technique in escaping from convoluted search space. In the next section, we study the effect of random restart in DLM and DLMI.

7 Random restart in DLM and DLMI

DLMI is behind DLM in robustness with the difficult random CSP instances. We investigate extending DLMI with random restart. As a control, we also examine the effect of random restart in DLM. The results are two fold. First, random restart has little effect on DLM. Second, the enhanced DLMI is comparable to the original DLMI on easy problems, and more efficient and robust on difficult problems.

Table 7 Benchmarking results of DLM_r

Instance	Succ	Time	Flips	Restarts
$PS = 2$				
10-queen	20/20	0.01	317	0
20-queen	20/20	0.02	271	0
50-queen	20/20	1.18	1392	0
100-queen	20/20	91.41	5441	0
$PS = 4$				
pp-50	20/20	1.15	1433	0
pp-60	20/20	3.80	2144	0
pp-70	20/20	11.85	2963	0
pp-80	20/20	25.66	3616	0
pp-90	20/20	52.85	4453	0
pp-100	20/20	105.89	5421	0
$PS = 3$				
ap-10	20/20	0.43	39918	0
ap-20	0/20	–	–	–
ap-30	0/20	–	–	–
ap-40	0/20	–	–	–
$PS = 4$				
magic-10	20/20	0.04	879	0
magic-15	20/20	0.64	3559	0
magic-20	20/20	6.11	12462	0
magic-25	0/20	*	*	*
magic-30	0/20	*	*	*
magic-35	0/20	*	*	*

Random restart is a common technique to avoid search getting stuck in potentially non-fruitful regions by introducing randomness into the search process. It can be applied either at local minima or after a certain number of local moves. In the context of DLM and DLMI, we perform a random restart after *cutoff* number of flips with all the Lagrange multiplier values retained. For DLMI, we also have to ensure that the restart point is within the island (i.e. satisfying the island constraints), which can be achieved using the random initialisation strategy given in Sect. 4. After some tuning, we set the *cutoff* value to be 1,000,000 for all the following experiments.

As a control, we implement DLM_r, a version of DLM with random restart. Results are reported in Tables 7 and 8 reporting the success ratio, time, number of flips, and also average number of restarts invoked. We use also the same parameter sets, PS, as in DLM for each class of problems respectively. By comparing to Tables 2 and 3, we can see that random restart gives little improvement over DLM, if not making it worse. We observe that, with the large *cutoff* value of 1,000,000, restart is invoked only for the very difficult instances, such as g125n-17c and the phase-transition random CSPs.

Table 8 Benchmarking results of DLM$_r$ *(cont'd)*

Instance	Succ	Time	Flips	Restarts
$PS = 3$				
g125n-18c	20/20	1.88	9210	0
g250n-15c	20/20	4.53	2224	0
g125n-17c	20/20	66.07	905295	0.45
g250n-29c	20/20	175.98	317992	0
$PS = 4$				
rcsp-120-10-60-75	20/20	4.33	4783	0
rcsp-130-10-60-75	20/20	5.70	4428	0
rcsp-140-10-60-75	20/20	9.27	7083	0
rcsp-150-10-60-75	20/20	9.06	6462	0
rcsp-160-10-60-75	20/20	12.44	7029	0
rcsp-170-10-60-75	20/20	14.54	6238	0
$PS = 3$				
rcsp-120-10-60-5.9	20/20	81.26	1232808	0.85
rcsp-130-10-60-5.5	20/20	725.95	10304493	9.80
rcsp-140-10-60-5.0	20/20	73.18	969920	0.55
rcsp-150-10-60-4.7	20/20	461.27	6267942	5.75
rcsp-160-10-60-4.4	20/20	135.13	1509024	1.05
rcsp-170-10-60-4.1	20/20	214.29	2034786	1.40
$PS = 3$				
rcsp-120-10-60-5.8	20/20	43.12	558389	0.15
rcsp-130-10-60-5.4	20/20	64.46	832706	0.40
rcsp-140-10-60-4.9	20/20	16.32	201521	0
rcsp-150-10-60-4.6	20/20	27.05	323166	0
rcsp-160-10-60-4.3	20/20	57.21	595700	0.30
rcsp-170-10-60-4.0	20/20	12.49	128033	0

Next, we examine the results of DLMI$_r$, which is DLMI incorporated with random restart, in Tables 9 and 10. We use the same P values for each benchmark type as in the DLMI experiments. Again, the same parameter sets, PS, as in the DLM experiments are adopted. By bolding numbers where DLMI$_r$ is bettered by DLMI, we have the following observations when comparing DLMI$_r$ with DLMI in Tables 2 and 3. First, in most of the cases where DLMI$_r$ is bettered by DLMI (except ap-20), the differences are slight. DLMI$_r$ is basically on par with DLMI on these instances. Second, in the case of ap-20, DLMI$_r$ is twice as slow as DLMI, but still over six times more efficient than DLM. Third, DLMI$_r$ solves the robustness problem in the difficult random CSPs with 100% success rate in all instances. Fourth, restart is needed only in the difficult instances. Fifth, DLMI$_r$ dominates over DLM completely both in terms of time and robustness.

Table 9 Benchmarking results of DLMI$_r$

Instance	Succ	Time	D-Flips	I-Flips	Restarts
$PS = 2$, and $P = 0.3$					
10-queen	20/20	0.002	21	16	0
20-queen	20/20	**0.01**	67	**56**	0
50-queen	20/20	**0.09**	**186**	**161**	0
100-queen	20/20	**1.04**	**464**	**414**	0
$PS = 4$, and $P = 0.3$					
pp-50	20/20	**0.09**	**253**	**226**	0
pp-60	20/20	**0.23**	**347**	**316**	0
pp-70	20/20	**0.40**	**396**	**361**	0
pp-80	20/20	**0.62**	**465**	**424**	0
pp-90	20/20	**0.90**	**497**	**453**	0
pp-100	20/20	**1.21**	**467**	**416**	0
$PS = 3$, and $P = 0.3$					
ap-10	20/20	0.02	2509	2503	0
ap-20	20/20	**66.47**	**3604335**	**3604332**	6.6
ap-30	0/20	–	–	–	–
ap-40	0/20	–	–	–	–
$PS = 4$, and $P = 0.1$					
magic-10	20/20	**0.02**	**268**	**172**	0
magic-15	20/20	0.05	**939**	**721**	0
magic-20	20/20	**0.22**	**1575**	**1185**	0
magic-25	20/20	0.63	**2459**	**1846**	0
magic-30	20/20	1.37	**3506**	**2621**	0
magic-35	20/20	2.78	**5620**	**4413**	0

8 Conclusion

The most successful local search methods for CSPs, in particular SAT problems, treat the satisfaction problem as an unconstrained optimisation problem, and attempt to minimise the number of unsatisfied constraints. In this work we show we can tackle CSPs as constrained optimisation problems by carefully defining which constraints are "hard," that will never be violated in the search, and which are "soft" and contribute to the objective function. The key requirement is that the solutions of the "hard" constraints form an *island*, that is a connected neighbourhood. We define the *island confinement method*, a generic modification of a local search procedure to include "hard" constraints. The main benefit of the island confinement method is that the search space can be dramatically reduced.

We have demonstrated on an important class of SAT problems, SAT formulations of binary CSPs, that we can choose an island that encompasses a large part of the constraints of the problem. The purpose of our work is not to compete with other CSP solvers, but to demonstrate that the incorporation of the island confinement method

Table 10 Benchmarking results of DLMI$_r$ *(cont'd)*

Instance	Succ	Time	D-Flips	I-Flips	Restarts
$PS = 3$, and $P = 0.15$					
g125n-18c	20/20	**0.25**	5059	4952	0
g250n-15c	20/20	0.23	1066	823	0
g125n-17c	20/20	**22.75**	582132	581967	0.65
g250n-29c	20/20	**50.84**	242425	242195	0.05
$PS = 4$, and $P = 0.3$					
rcsp-120-10-60-75	20/20	0.27	399	284	0
rcsp-130-10-60-75	20/20	0.37	482	360	0
rcsp-140-10-60-75	20/20	0.53	650	515	0
rcsp-150-10-60-75	20/20	0.76	920	779	0
rcsp-160-10-60-75	20/20	0.73	695	544	0
rcsp-170-10-60-75	20/20	1.15	1396	1231	0
$PS = 3$, and $P = 0.3$					
rcsp-120-10-60-5.9	20/20	**19.21**	926724	926501	1.40
rcsp-130-10-60-5.5	20/20	**159.00**	**7395450**	**7394214**	14.20
rcsp-140-10-60-5.0	20/20	**13.25**	**603533**	**603357**	0.70
rcsp-150-10-60-4.7	20/20	**91.06**	**3894558**	**3893714**	7.15
rcsp-160-10-60-4.4	20/20	**31.43**	**1375332**	**1374927**	2.30
rcsp-170-10-60-4.1	20/20	**14.46**	**639366**	**639133**	0.65
$PS = 3$, and $P = 0.3$					
rcsp-120-10-60-5.8	20/20	**9.28**	**469356**	**469219**	0.45
rcsp-130-10-60-5.4	20/20	**17.59**	**821111**	**820893**	1.10
rcsp-140-10-60-4.9	20/20	**3.53**	**162309**	**162179**	0.05
rcsp-150-10-60-4.6	20/20	3.94	177047	176911	0.05
rcsp-160-10-60-4.3	20/20	**9.21**	**415100**	**414930**	0.35
rcsp-170-10-60-4.0	20/20	1.59	69183	69039	0

into state-of-the-art local search SAT solvers, DLM and ESG, leads to significant improvements. We also show that the use of the island confinement method is not without difficulty, since we must develop new island trap escaping strategies.

We believe that there is plenty of scope for using the island confinement method to improve local search for other classes of CSPs, for example arbitrary SAT problems. It will be interesting to study if the method can be integrated effectively to other local search algorithms, which include (1) those based on clause weighting such as SAPS (Hutter et al. 2002) and PAWS (Thornton et al. 2004) and (2) those without clause weighting such as WalkSAT. The principal challenge lies in building an adequate island trap escaping strategy. It will also be worthwhile to investigate the tuning of the P parameter and the *cutoff* value.

Acknowledgements We thank the anonymous reviewers of AAAI'02 for their constructive and insightful comments. We are also grateful to Gordon Lam who helped to run some of the experiments. The work described in this paper was substantially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK4204/01E).

References

Brafman, R.: A simplifier for propositional formulas with many binary clauses. In: Proceedings of IJ-CAI'01, pp. 515–522 (2001)

Choi, K., Lee, J., Stuckey, P.: A Lagrangian reconstruction of GENET. Artif. Intell. **123**(1–2), 1–39 (2000)

Davenport, A., Tsang, E., Wang, C., Zhu, K.: GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement. In: Proceedings of AAAI'94, pp. 325–330 (1994)

Fang, H., Kilani, Y., Lee, J., Stuckey, P.: Reducing search space in local search for constraint satisfaction. In: Dechter, R., Sutton, R., Kearns, M. (eds.) Proceedings of the 18th National Conference on Artificial Intelligence, pp. 28–33 (2002)

Fang, H., Kilani, Y., Lee, J., Stuckey, P.: Islands for SAT. Technical report, Computing Research Repository (CORR) (2006). http://arxiv.org/abs/cs.AI/0607071

Hoos, H.: On the run-time behavior of stochastic local search algorithms for SAT. In: Proceedings of AAAI'99, pp. 661–666 (1999)

Hutter, F., Tompkins, D., Hoos, H.: Scaling and probabilistic smoothing: efficient dynamic local search for SAT. In: Proceedings of CP'02, pp. 233–248 (2002)

Mackworth, A.: Consistency in networks of relations. Artif. Intell. **8**(1), 99–118 (1977)

Minton, S., Johnston, M., Philips, A., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling. Artif. Intell. **58**, 161–205 (1992)

Morris, P.: The breakout method for escaping from local minima. In: Proceeding of AAAI'93, pp. 40–45 (1993)

Schuurmans, D., Southey, F.: Local search characteristics of incomplete SAT procedures. In: Proceedings of AAAI'00, pp. 297–302 (2000)

Schuurmans, D., Southey, F., Holte, R.: The exponentiated subgradient algorithm for heuristic boolean programming. In: Proceedings of IJCAI'01, pp. 334–341 (2001)

Selman, B., Kautz, H.: Domain-independent extensions to GSAT: solving large structured satisfiability problems. In: Proceedings of IJCAI'93, pp. 290–295 (1993)

Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: Proceedings of AAAI'92, pp. 440–446 (1992)

Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of AAAI'94, pp. 337–343 (1994)

Stuckey, P.J., Tam, V.: Extending GENET with lazy arc consistency. IEEE Trans. Syst. Man Cybern. **28**(5), 698–703 (1998)

Thornton, J., Pham, D., Bain, S., Ferreira, V. Jr.: Additive versus multiplicative clause weight for SAT. In: Proceedings of AAAI'04, pp. 191–196 (2004)

Wu, Z., Wah, B.: Trap escaping strategies in discrete Lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In: Proceedings of AAAI'99, pp. 673–678 (1999)

Wu, Z., Wah, B.: An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In: Proceedings of AAAI'00, pp. 310–315 (2000)