Reducing Search Space in Local Search for Constraint Satisfaction

H. Fang* and Y. Kilani† and J.H.M. Lee† and P.J. Stuckey†

Abstract

Typically local search methods for solving constraint satisfaction problems such as GSAT, WalkSAT and DLM treat the problem as an optimization 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 valuations to move to, until finally a solution which satisfies all constraints is found.

In this paper we investigate using some of the constraints, rather than as part of a penalty function, as “hard” constraints, that are always satisfied by every trial valuation visited. In this way the 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. Treating some constraints as hard also provides 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.

We show in this paper how, for DIMACS translations of binary CSPs, treating some constraints as hard can significantly improve search performance of the DLM local search method.

Keywords: SAT, local search, binary CSP.

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 of values from the domains to their respective variables so that all constraints are satisfied simultaneously. CSPs are well-known to be NP-hard in general.

Local search techniques, for example GSAT (Selman, Levesque, & Mitchell 1992), WalkSAT (Selman & Kautz 1993; Selman, Kautz, & Cohen 1994), DLM (Wu & Wäh 1999; 2000), the min-conflicts heuristic (Minton et al. 1992), and GENET (Davenport et al. 1994), have been successful in solving large constraint satisfaction problems. 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), and DLM modify the landscape of the search surface. Following Morris, we call these breakout methods. WalkSAT introduces noise into the search procedure so as to avoid and escape from local minima.

Local search algorithms traverse the search surface of a usually enormous search space to look for solutions using some heuristic function. The efficiency of a local search algorithm depends on three things: (1) the size of the search space (the number of variables and the size of the domain of each variable), (2) the search surface (the structure of each constraint and the topology of the constraint connection), and (3) the heuristic function (the definition of neighbourhood and how a “good” neighbour is picked). We propose the Island Confinement Method which aims to reduce 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 in only this region for solutions. By connectedness, we mean the ability to move from one point to any other point within the region without moving out of the region. Therefore, we are guaranteed that searching within this confined space will not cause
us to miss any solutions. The entire search space is trivially such a region but we would like to do better.

In this paper we illustrate one method for choosing a subset of the problem constraints 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 constraints.

The introductions of island constraints complicates the search procedure because it may defeat the local minima escaping strategy of the underlying search procedure. We show how to modify DLM, a very competitive local search procedure for SAT problems so that it handles island constraints, and give empirical results showing where the island confinement method can give substantial improvements in solving some classes of SAT problems.

**Background and Definitions**

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

A state of problem \((Z, D, C)\) (or simply \(C\)) is a valuation for \(Z\) (where \(Z = \bigcup_{c \in C} \text{var}(c)\)). A state \(s\) is a solution of a constraint \(c\) if \(s\) makes \(c\) true. A state \(s\) is a solution of a CSP \((Z, D, C)\) if \(s\) is a solution to all constraints in \(C\) simultaneously.

**SAT**

SAT problems are a special case of CSPs. A (propositional) variable can take the value of either 0 (false) or 1 (true). A literal is either a variable \(x\) or its complement \(\bar{x}\). A literal \(l\) is true if \(l\) assumes the value 1; \(l\) is false otherwise. A clause is a disjunction of literals, which is true when one of its literals is true. For simplicity we assume that no literal appears in a clause more than once, and no literal and its negation appear in a clause (which would then be trivial). A satisfiability problem (SAT) consists of a finite set of clauses (treated as a conjunction). Let \(\bar{I}\) denote the complement of literal \(l\): \(\bar{I} = \bar{x}\) if \(l = x\), and \(\bar{I} = x\) if \(l = \bar{x}\). Let \(L = \{l \mid l \in L\}\) for a literal set \(L\).

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

**Local Search**

A local search solver moves from one state to another using a local move. We define the neighbourhood \(n(s)\) of a state \(s\) to be all the states that are reachable in a single move from state \(s\). 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.

For the purpose of this paper, where 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

\[d_h(s_1, s_2) = |s_1 - (s_1 \cap s_2)| = |s_2 - (s_1 \cap s_2)|.\]

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 abuse of terminology we will also refer to flipping a literal \(l\) which simply means flipping the variable occurring in the literal.

A local move from state \(s\) is a transition, \(s \Rightarrow s'\), from \(s\) to \(s' \in n(s)\). 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 \(s\); and
- possibly an optional “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 & Kautz 1993; Selman, Kautz, & Cohen 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 others.

**Island Constraints**

We introduce the notion of island constraints, the solution space of which is connected in the following sense. 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 constraint is an island constraint if we can move between any two states in the constraint’s solution space using a finite sequence of local moves without moving out of the solution space.

Let \(\text{sol}(C)\) 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 \(s_0, s_n \in \text{sol}(C)\), there exist states \(s_1, \ldots, s_{n-1} \in \text{sol}(C)\) 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 \(C\).

We give a simple sufficient condition for when a set \(C\) of clauses results in an island. Let \(\text{lit}(c)\) denote the set of all literals of a clause \(c\). Let \(\text{lit}(C) = \bigcup_{c \in C} \text{lit}(c)\). A set \(C\) of clauses is non-conflicting if there does not exist a variable \(x\) such that \(x, \bar{x} \in \text{lit}(C)\).

**Theorem 1** A non-conflicting set \(C\) of clauses forms an island.

**Proof:** Since \(C\) is non-conflicting \(\text{lit}(C)\) can be extended to a state (it does not have both a literal and its complement). Any state \(s \supseteq \text{lit}(C)\) clearly satisfies \(C\). We show by induction that for any state \(s_0\) satisfying \(C\) there is a path \(s_0 \Rightarrow s_1 \Rightarrow \cdots \Rightarrow s_n = s\) where each \(s_i\) satisfies \(C\). Since a path is reversible, there is a path between any two solutions
s₀ and s₀' of C via s and hence C is an island. Let l be an arbitrary literal where s₁ and s differ, that is l ∈ s₁ and l ⊄ s. Then l ∉ lit(C) and clearly s₁+l = s₁ − {l} ∪ {l} satisfies C since l does not occur in C and hence cannot be the only literal satisfying one of the clauses of C.

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 ∈ Z is mapped to a set of propositional variables \{x₁, …, xₙ\} where Dₓ = \{a₁, …, aₙ\}.
- For every x ∈ Z, SAT(Z, D, C) contains the clause \(x₁ ∨ … ∨ xₙ\), which ensures that the any solution to the SAT problem gives a value to x.
- Each binary constraint c ∈ C with var(c) = \{x, y\} is mapped to a series of clauses. If \(x → a \land y → a'\) is not a solution of c we add the clause \(\overline{x}_a \lor \overline{y}_{a'}\) to SAT(Z, D, C). This ensures that the constraint c holds in any solution to the SAT problem.

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 \(\overline{x} \lor \overline{y}\) except for a single clause for each variable in Z. The first class of clauses forms a non-conflicting set trivially.

**The Island Confinement Method in DLM**

DLM (Wu & Wah 1999) 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 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

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

where \(\vec{\lambda}\) are Lagrange multipliers, one for each constraint, which give the “penalty” for violating that constraint. The saddle point search changes the state to decrease the Lagrangian function, or increase the Lagrange multipliers.

The core of the DLM algorithm can be extracted from Figure 1, by considering only lines without the “|” mark in addition to three slight modifications. First, the input to DLM is simply a set of clauses \(\vec{c}\). Second, in the second line, \(s\) should be initialized to any random valuation for var(\(\vec{c}\)). Third, all occurrences of \(c'_i\) and \(\vec{\lambda}_i\) should be changed to \(\vec{c}\) and \(\vec{\lambda}\) respectively. 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. We say such neighbours are invalid. The full DLM algorithm also includes a tabu list and methods for updating Lagrange multipliers; see (Wu & Wah 2000) for details.

Handling island constraints is simple at first glance. Given a problem defined by a set of clauses \(\vec{c}_i \land \vec{c}_j\) partitioned into island constraints \(\vec{c}_i\) and remaining clauses \(\vec{c}_j\), we simply modify the algorithm to treat the remaining clauses as penalty functions and give an initial valuation \(s\) which is a solution of \(\vec{c}_j\). For SAT(Z, D, C), \(\vec{c}_i\) consists of clauses of the form \(\overline{x} \lor \overline{y}\). An arbitrary extension of lit(\(\vec{c}_i\)) to all variables can always be such an initial valuation. We exclude literals \(l \in \text{unsat}\) from flipping when \(s' = s - \{\overline{l}\}\) does not satisfy \(\vec{c}_i\). Hence we only examine states that are adjacent to s and satisfy \(\vec{c}_i\). Let \(n(s, \vec{c}_i) = \{s' \in n(s) \mid s' \in \text{sol}(\vec{c}_i)\}\). The rest of the algorithm remains unchanged. A new problem arises.

**Example 1** Suppose we have the following clauses, where \(\vec{c}_i = (c_1, c_2, c_3)\) and \(\vec{c}_r = (c_4, c_5)\).

\[
\begin{align*}
c_1 & : \overline{x}_1 \lor \overline{x}_4 \\
c_2 & : \overline{x}_2 \lor \overline{x}_5 \\
c_3 & : \overline{x}_3 \lor \overline{x}_5 \\
c_4 & : x_1 \lor x_2 \lor x_3 \\
c_5 & : x_4 \lor x_5
\end{align*}
\]

and the current state is \{x₁, x₂, x₃, x₄, x₅\}, which satisfies \(c₁, c₂, c₃\) and \(c₄\). Three neighbours satisfy the island clauses \(c₁, c₂, c₃\): \{x₁, x₂, x₃, x₄, x₅\}, \{x₁, x₂, x₃, x₄, x₅\}, \{x₁, x₂, x₃, x₄, x₅\}, and \{x₁, x₂, x₃, x₄, x₅\}, all of which are invalid. Whatever the Lagrange multipliers for \(c₁, c₂, c₃\) and \(c₄\), none of the neighbours will be better than the current state. Hence DLM will always remain in this state no matter how the Lagrange multipliers are updated (in fact it will never consider the invalid moves), and cannot escape from this local minima. We call this an island trap.

More formally an island trap for a problem \(\vec{c}_i \land \vec{c}_j\) is a state s such that for all states \(s' \in n(s, \vec{c}_i)\) whatever the value of the Lagrange multipliers \(\vec{\lambda}\), no neighbour would be better than s, i.e. \(\forall s' \in n(s, \vec{c}_i) \land \vec{\lambda}, \exists l \in \text{unsat} \land \lambda_l > 0 \land \text{L}(s, \vec{\lambda}) \geq \text{L}(s', \vec{\lambda})\). This holds if and only if \(\{c \in \vec{c}_r \mid s \in \text{sol}(\{c\})\} \subseteq \{c \in \vec{c}_r \mid s' \in \text{sol}(\{c\})\}\) for all \(s' \in n(s, \vec{c}_i)\).

In order to escape from an island trap, we need to flip some variable(s) to make 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 \(\text{makes}(l, s, \vec{c}_i) = \{c \in \vec{c}_i \mid (s - \{l\} \cup \{l\}) \notin \text{sol}(\{c\})\}\) be the island constraints that are satisfied in the current valuation s only by the literal l. If \(\text{makes}(l, s, \vec{c}_i)\) is non-empty then we cannot flip the literal l in the current state without going outside the island.

Let \(\text{freeme}(l, s, \vec{c}_i) = \{l' \mid \{l \lor l'\} \notin \text{makes}(l, s, \vec{c}_i)\}\) be the set of literals that need to be made true in order that we can flip literal l to l' and stay within the island.
The base island trap escaping strategy we propose is thus: choose the literal \( l \) in an unsatisfied clause in \( \bar{c}_r \), according to state \( s \) such that \( |\text{freeme}(\bar{t}, s, \bar{c}_l)| > 0 \) and minimal in size, and flip all literals in \( \text{freeme}(\bar{t}, s, \bar{c}_l) \) 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 2** Continuing Example 1, we find that in state \( s = \{x_1, x_2, \bar{x}_3, \bar{x}_4, \bar{x}_5\} \), the unsatisfied clause is \( x_4 \lor \bar{x}_5 \). Now \( \text{makes}(\bar{x}_4, s, \bar{c}_l) = \{c_1\} \), and \( \text{makes}(\bar{x}_5, s, \bar{c}_l) = \{c_2\} \). In addition \( \text{freeme}(\bar{x}_4, s, \bar{c}_l) = \{\bar{x}_2\} \), and \( \text{freeme}(\bar{x}_5, s, \bar{c}_l) = \{\bar{x}_1\} \). Suppose we choose randomly to free \( \bar{x}_4 \), then we flip all the literals in its freeme set (\( \bar{x}_1 \)) obtaining the new state \( \{\bar{x}_1, x_2, \bar{x}_3, \bar{x}_4, \bar{x}_5\} \). We can now flip \( \bar{x}_4 \) while staying in the island and also arriving at the solution \( \{\bar{x}_1, x_2, \bar{x}_3, x_4, \bar{x}_5\} \).

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, \( \text{tabulit} \), of length 1, to cope with the most common case that \( \text{freeme} \) is of size 1. Unlike the regular tabu list, the literal in \( \text{tabulit} \) is not allowed to be flipped under any circumstances (variables in the DLM tabu list can be flipped if the move is downhill). In order to get out of very deep traps, we occasionally need to flip many variables. To make this happen 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 Figure 1. Lines beginning in “\( | \)” are either different from their counterparts in the original DLM algorithm or new additions. For DLMI there are only Lagrange multipliers \( \bar{\lambda}_r \) for the non-island clauses \( \bar{c}_r \). A random valuation that satisfies the island clauses \( \bar{c}_l \) is chosen (since \( \bar{c}_l \) 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 \( \text{tabulit} \). If there are candidates then we proceed as in DLM: otherwise we are in an island trap. Note that \( \text{tabulit} \) has introduced another kind of island trap where no flip will satisfy more clauses except flipping the literal in \( \text{tabulit} \), which is disallowed. This trap is handled identically to the original island trap.

In an island trap we consider the literals \( \{\text{free}_1\} \) in the unsatisfied clauses which could not be flipped without breaking an island constraint. Note that \( \text{free} \neq \emptyset \) otherwise we have a solution. We separate these into those requiring 1 other literal to be flipped to free them (\( \text{free}_1 \)), and those requiring two or more (\( \text{free}_2 \)). If the random number is greater than parameter \( P \) we choose a literal in \( \text{free}_2 \) to free, and flip all the variables required to free it. Otherwise we choose, if possible, a variable in \( \text{free}_1 \) whose \( \text{freeme} \) is not the literal in \( \text{tabulit} \) 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 \( \text{free}_2 \) 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 \( \text{free}_1 \) has its freeme set equal to \( \text{tabulit} \). In this case we have detected that \( \text{tabulit} \) must hold, and we can eliminate the variable involved by unit resolution. In our code this is performed, we could avoid it by simplifying the original SAT formulation so that all such occurrences are removed, using SAT simplification methods such as (Brafman 2000).

---

**Figure 1:** DLMI
Example 3  Modifying clause $c_2$ in Example 1 slightly.

$$
\begin{align*}
    c_1 & : \overline{x}_1 \lor \overline{x}_4 \\
    c_2 & : \overline{x}_1 \lor \overline{x}_5 \\
    c_3 & : \overline{x}_3 \lor \overline{x}_5
\end{align*}
$$

We are in state $s = \{x_1, x_2, \overline{x}_3, \overline{x}_4, \overline{x}_5\}$ and tabu list is $\{\overline{x}_1\}$. The literals in unsatisfied clauses are $\text{unsat} = \{x_4, x_5\}$, and $\text{candidate} = \emptyset$ since neither literal can be flipped. Hence $\text{free} = \{x_4, x_5\}$. Both of these literals are placed in $\text{free}_1$. Since $\text{freeeme}(\overline{x}_4, s, \overline{c}_1) = \text{freeeme}(\overline{x}_5, s, \overline{c}_1) = \{\overline{x}_1\}$. The selection of a literal $l$ in $\text{free}_1$ will fail. This provides a proof that $\{\overline{x}_1\}$ must hold in any solution of $\overline{c}_1 \land \overline{c}_1$. We have $x_4 \lor x_5 \in \overline{c}_1$ and $\overline{x}_1 \lor \overline{x}_4$ and $\overline{x}_1 \lor \overline{x}_5$ in $\overline{c}_1$, then by resolution we obtain $\overline{x}_1$. In the context of CSP, $x_1$ corresponds to a value in the domain of a CSP variable (say $u$) which is incompatible with the two (all) values in the domain of the other CSP variable (say $v$). That is why 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 $u$.  

Experiments

We implemented DLM in by modifying the code of distribution of SAT-DLM-2000, maintaining all the extra parts such as the tabu list, and penalty updating methods unchanged. We compare DLM with DLM using the best parameter settings for DLM of the five (Wu & Wah 2000) included in the distribution. For the additional parameter $P$ which we introduce, we use the setting 0.3 which performs the best overall. The results presented were obtained using a PC with Pentium III 800 Mhz and 256 MB memory.

Table 1 shows the comparison of DLM and DLM in on $N$-queens problems and a suite of binary CSPs from (Choi, Lee, & Stuckey 2000). We first transform the problem instances into SAT. Of the clauses in all instances, over 99% are island clauses. For each set of benchmark instances, we give the parameter settings (PS) from SAT-DLM-2000 used for DLM and also DLM. Runs failing to find solution in one hour are aborted. The table shows number of variables (Vars), and number of clauses (Cls) in the SAT formulation, then the success ratio, average solution time (in seconds) and average flips on solved instances for DLM and DLM.

DLM shows substantial improvement over DLM using the same parameter set on the test suite. Generally DLM traverses a smaller search space and needs to do less maintenance for island clauses and this is a significant saving. In many cases DLM is one to two orders of magnitude better than DLM. DLM is bettered marginally by DLM only in the hard graph coloring problem g125n-17c. DLM is also slightly less robust in success rate with the phase transition random CSPs. This occurs because the search surface is now considerably more jagged. DLM might appear to use more flips than DLM in a few cases, but many flips are used in escaping from island traps, and these are considerably cheaper since they do not require any computation of the Lagrangian function values.


Conclusion

The island concept can significantly reduce the search space of a local search procedure, by treating some constraints as “hard” so that they are never violated during search process. We have shown one instance where we can define an island which encompass a large part of the constraints in a problem: SAT formulations of binary CSPs. Interestingly in this case it corresponds to a local search on the original CSP where some CSP variables may not have values, but all constraints are always satisfied. We believe there is plenty of scope for using the island concept to improve other local search algorithms, such as WalkSAT and others. The difficulty lies in building an adequate island trap escaping strategy.

Acknowledgements

We thank the anonymous referees for constructive comments. 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


Table 1: Comparative empirical results DLM versus DLMI: "*" indicates a segmentation fault, and bold entries show when DLM betters DLMI.