Empirical Studies of Heuristic Local Search for Constraint Solving*

Jin-Kao Hao and Raphaël Dorne

LGI2P
EMA-EERIE
Parc Scientifique Georges Besse
F-30000 Nîmes
France
email: {hao, dorne}@eerie.fr

Abstract. The goal of this paper is twofold. First, we introduce a class of local search procedures for solving optimization and constraint problems. These procedures are based on various heuristics for choosing variables and values in order to examine a general neighborhood. Second, four combinations of heuristics are empirically evaluated by using the graph-coloring problem and a real world application - the frequency assignment problem. The results are also compared with those obtained with other approaches including simulated annealing, Tabu search, constraint programming and heuristic graph coloring algorithms. Empirical evidence shows the benefits of this class of local search procedures for solving large and hard instances.
Keywords: Local search, constraint solving, combinatorial optimization, graph coloring, frequency assignment.

1 Introduction

Constraint problems embodies a class of general problems which are important both in theory and in practice. Well-known examples include constraint satisfaction problems (CSP), maximal constraint satisfaction problems (MCSP) [5] and constraint satisfaction optimization problems (CSOP) [18]. Constraint problems can be considered as search problems, i.e. given a finite search space composed of a set of configurations, we want to find one or more particular configurations which minimize (or maximize) certain pre-defined criteria. Constraint problems have many practical applications related to scheduling, transportation, layout/circuit design, telecommunications and so on. In general, constraint problems are NP-hard. Consequently, there is little hope of finding any deterministic polynomial solution for this class of problems. Given their practical importance, many methods have been devised to tackle these search problems. This paper looks at one class of methods which are based on surprisingly simple, yet powerful local search techniques.

* Work partially supported by the CNET (French National Research Center for Telecommunications) under the grant No.940B006-01.

Local search (LS), also called neighborhood search, constitutes an important class of general heuristic methods based on the notion of neighborhood [14]. Starting with an initial configuration, a typical LS procedure replaces iteratively the current configuration by one of its neighbors, which is often of better quality, until some stop criteria are verified; for example, a fixed number of iterations is reached or a sufficiently good local optimum is found. Well-known examples of LS-based methods include simulated annealing (SA) [11], Tabu search [6] and various forms of hill-climbers [1]. Given that LS uses only a neighborhood function and possibly some other general notions, it can be applied to a large class of problems. Traditionally, LS was used with success to tackle well-known NP-hard combinatorial optimization problems such as TSP [12] and graph partitioning [10]. More recent applications of LS include the graph coloring problem [8, 9, 3], CSPs [13], and the satisfiability problem [7, 16].

Local search is essentially based on three components: a *configuration structure (encoding)*, a *neighborhood function* defined on the configuration structure, and a *neighborhood examination mechanism*. The first component defines the search space S of the application, the second associates a subset of S with each point of the search space while the third defines the way of going from one point to another. The configuration structure is often application-dependent and should be chosen in such a way that it reflects the natural solution space of the problem and facilitates its exploration and exploitation. For a given neighborhood, the way in which neighbors are examined is certainly the most determinant part of the performance of a LS procedure.

In this paper, we present a class of LS procedures for solving optimization and constraint problems. These LS procedures are based on different heuristics for examining a general neighborhood. Some heuristics are well-known and others less so. Computational tests are carried out on two NP-hard problems: graph coloring and frequency assignment in mobile radio networks. Experimental evidence shows the benefits of these procedures for solving large and hard instances. The results are compared with those obtained by two other LS procedures: SA and Tabu search and two other heuristic methods: constraint programming and heuristic graph coloring algorithms.

2 Constraint Problems and Local Search

2.1 Constraint Problems

In order to apply LS to constraint problems, we will consider constraint problems as combinatorial optimization problems, i.e., a constraint problem P is defined by a quadruple $< X, D, C, f >$ where

- $X = \{V_1, V_2 ... V_n\}$ is all the distinct variables in P,
- $D = \{D_1, D_2 ... D_n\}$ is all the domains of variables,
- $C = \{C_i(V_{i_1} ... V_{i_k})\}$ is the set of constraints, each C_i being a relation on $V_{i_1} ... V_{i_k} \in X$,
- f is a cost function to be minimized (maximized).

With this definition, several cases are possible. First, if P is a standard CSP, there will be no associated cost function. Therefore, any assignment of values to the variables satisfying the constraints of C will be a solution. Second, if P is a CSOP, then solving P implies finding assignments such that all the constraints are satisfied and the cost f minimized (or maximized). Third, if P is a MCSP, i.e. the underlying $CSP < X, D, C >$ is not satisfiable, then solving P is to maximize (minimize) the number of satisfied (unsatisfied) constraints.

Note that in this formulation, the *cost* function f is not necessarily the *evaluation* function which is required by any LS search procedure to evaluate the quality of configurations.

2.2 Configuration and Neighborhood

Given a constraint problem $P =< X, D, C, f >$, the configuration structure is defined as follows: a configuration s is any complete assignment such that $s = \{< V_i, v_i > \mid V_i \in X \text{ and } v_i \in D_i\}$. We use $T(s, V_i)$ to note the value of V_i in s, i.e. if $< V_i, v_i > \in s$ then $T(s, V_i) = v_i$. The search space S of the problem P is then defined as the set of all the possible configurations. Clearly the cardinality of S is equal to the product of the size of the domains, i.e. $\prod_{i=1}^{n} |D_i|$.

In general, the configuration structure is application-dependent. However, some configuration structures are general enough to be applied to many applications. The above structure is such an example. In fact, it can be used to model problems such as graph coloring, satisfiability and many CSPs. Another general configuration structure is "permutation" which can be used to model naturally the traveling salesman problem.

Given the configuration structure defined above, the neighborhood function $N : S \rightarrow 2^S$ may be defined in many ways. In this paper, we use the *one-difference* or *one-move* neighborhood[2]. Formally, let $s \in S$ be a configuration, then $s' \in N(s)$ if and only if there exists one and only one $i \in [1..n]$ such that $T(s, V_i) \neq T(s', V_i)$. In other words, a neighbor of a configuration s can be obtained by changing the current value of a variable in s. Since a variable V_i in s can take any of its $|D_i|$ values, s has exactly $\sum_{i=1}^{n} (|D_i| - 1) = \sum_{i=1}^{n} D_i - n$ neighbors. Note that this neighborhood is a reflexive and symmetric relation.

2.3 Neighborhood Examination

We now turn to different ways of examining the neighborhood, i.e., going from one configuration to another. In this paper, we use a two-step, heuristic-based examination procedure to perform a move.

1. first choose a *variable*,
2. and then choose a *value* for the selected variable.

It is easy to see that many heuristics are possible for both choices. In what follows, we present various heuristics for choosing variables and values.

[2] In general, a k-move neighborhood can be defined.

Heuristics for choosing the variable V_i:

- *var.1 random*: pick randomly a variable V_i from X;
- *var.2 conflict-random*: pick randomly a variable from the *conflict set* defined by $\{V_i \mid V_i \in X$ is implicated in an unsatisfied constraint$\}$;
- *var.3 most-constrained*: pick a most constrained variable, for instance, the one which occurs in the biggest number of unsatisfied constraints (break ties randomly).

Heuristics for choosing the value for V_i:

- *val.1 random*: pick randomly a value from D_i;
- *val.2 best-one (min-conflicts[13])*: pick a value which gives the greatest improvement in the evaluation function (break ties randomly). If no such value exists, pick randomly a value which does not lead to a deterioration in the evaluation function (the current value of the variable may be picked);
- *val.3 stochastic-best-one*: pick a best, different value which does not lead to a deterioration in the evaluation function. If no such value exists, with probability p, take a value which leads to the smallest deterioration;
- *val.4 first-improvement*: pick the *first* value which improves the evaluation function. If no such value exists, take a value which does not lead to a deterioration or which leads to the smallest deterioration[3];
- *val.5 probabilistic-improvement*: with probability p, apply the *val.1 random* heuristics; with $1-p$, apply the *val.2 best-one* heuristic.

Let us note first that *val.2* forbids deteriorative moves. As we will see later, this property will penalize its performance compared with others.

Both *val.3* and *val.5* use a probability in order to accept deteriorative moves. The purpose of accepting such moves is to prevent the search from being stuck in local optima by changing the search direction from time to time. This probability may be static or dynamic. A static probability will not be changed during the search while a dynamic one may be modified by using some pre-defined mathematic laws or may adapt itself during the search. In any case, this probability is determined more often empirically than theoretically.

val.5 is similar to the *random-walk* heuristic used by GSAT [17], but they are different since for the satisfiability problem, there is only one explicit choice, i.e. the choice of the variable to flip. Here the heuristic determines the value for a chosen variable, not the variable itself. Another interesting point is that varying the probability p will lead to different heuristics. If $p = 1$, *val.5* becomes *val.1 random*. If $p = 0$, *val.5* becomes *val.2 best-one*. Finally, both the p part and the $1-p$ part can be replaced by other heuristics.

Evidently, any combination of a *var.x* heuristic and a *val.y* heuristic gives a different strategy for examining the neighborhood. There are 15 possibilities in our case. One aim of this work is to assess the performance of these combinations. Note that an extensive study on the *val.2 min-conflicts* heuristic for solving CSPs has been carried out and conclusions have been drawn [13]. However, that work

[3] As for *val.3*, a probability can be introduced here to control deteriorative moves.

concerns essentially the *value* choice for a given conflict variable. In this work, we study various combinations for choosing both variables and values.

Note finally that in order to efficiently implement the above variable/value choice heuristics, special data structures are indispensable to be able to recognize the conflict or the most constrained variables and the appropriate new value for the chosen variable.

2.4 Heuristic Local Search Template

Using the above variable/value choice heuristics, various heuristic local search (HLS) procedures can be built. The general HLS template is given below:

Procedure Heuristic Local Search (HLS)

Input:
$P =< X, D, C, f >$: the problem to be solved;
f & L: objective function and its lower bound to be reached;
MAX: maximum number of iterations allowed;
Output:
s: the best solution found;
begin
  generate(s); /* generate an initial configuration */
  $I \leftarrow 0$; /* iterations counter */
  while $(f(s) > L)$ **and** $(I < MAX)$ **do**
    choose a variable $V_i \in X$; /* heuristics var.x */
    choose a value $v_i \in D_i$ for V_i; /* heuristics val.y */
    if $T(s, V_i) \neq v_i$ **then**
      $s \leftarrow s - \{< V_i, T(s, V_i) >\} + \{< V_i, v_i >\}$;
    $I \leftarrow I + 1$;
  output(s);
end

This HLS template uses two parameters L and MAX in the stop condition. L fixes the (optimization) objective to be reached and MAX the maximum number of iterations allowed. Therefore, the procedure stops either when an optimal solution has been found or MAX iterations have been performed. The complexity of such a procedure depends on MAX, the size of domains $|D_i|$ and the way in which the neighborhood is examined.

3 Experimentation and Results

In this section, we present empirical results of HLS procedures which are based on some representative combinations of heuristics introduced above for exploiting the neighborhood structure. Tests are carried out on two NP-complete problems: the graph-coloring (COL) and the frequency assignment (FAP). For the

COL problem, test instances come essentially from the archives of the second DIMACS Implementation Challenge[4]. For the FAP problem, instances (60 in total) are provided by the CNET (French National Research Center for Telecommunications)[5]. A limited number of instances for each problem are used to evaluate four combinations of heuristics for choosing variables/values. More instances are then solved to compare these HLS procedures with other approaches including Tabu search, SA, constraint programming, and heuristic graph coloring algorithms.

3.1 Tests and Problem Encoding

Graph Coloring

There are two main reasons to choose the graph-coloring as our test problem. First, it is a well-known reference for NP-complete problems. Second, there are standard benchmarks in the public domain.

The basic COL is stated as a *decision* problem: given k colors and a graph $G =< E, V >$, is it possible to color the vertices of E with the k colors in such a way that any two adjacent vertices of V have different colors. In practice, one is also interested in the *optimization* version of the problem: given a graph G, find the smallest k (the chromatic number) with which there is a k-coloring for G.

Many classic methods for graph-coloring are based on either exhaustive search such as branch-and-bound techniques or successive augmentation heuristics such as Brélaz's DSATUR algorithm, Leighton's Recursive Largest First (RLF) algorithm and the more recent XRLF by Johnson et al. [9]. Recently, local search procedures such as SA [9] and Tabu search [8, 3] were also applied to the coloring problem.

In order to apply our HLS procedures to the graph-coloring problem, the COL must first be encoded. Given a graph $G =< E, V >$, we transform G into the following constraint problem $< X, D, C, f >$ where

- $X = E$ is the the set of the vertices of G,
- D is the set of the k integers representing the available colors,
- C is the set of constraints specifying that the colors assigned to u and v must be different if $\{u, v\} \in V$,
- f is the number of colors used to obtain a proper (conflict-free) coloring.

With this encoding, a (proper or improper) coloring of G will be a complete assignment $\{< u, i > | u \in E, i \in D\}$. Coloring G consists in assigning integers in D (colors) to the vertices in E in such a way that all the constraints of C are satisfied while a minimum number of k colors is used.

In order to minimize k, a HLS procedure solves a series of CSPs (decision problems). More precisely, it begins with a big k and tries to solve the underlying

[4] DIMACS benchmarks are available from ftp dimacs.rutgers.edu.

[5] Another set of FAP instances are available from ftp ftp.cs.city.ac.uk. These tests correspond to sparse graphs and are consequently much less constrained.

$CSP < X, D, C >$. If a proper k-coloring is found, the search process proceeds to color the graph with $k-1$ colors and so on. If no coloring can be found with the current k colors, the search stops and reports on the last proper coloring found. In other words, it tries to solve a harder problem (CSP) with fewer colors if it manages to solve the current one. In order to solve each underlying CSP, our local search procedure needs an *evaluation* function to measure the relative quality of each configuration (which is usually an improper coloring). Several possibilities exist to define this function. In this paper, it is defined simply as the number of unsatisfied constraints.

The DIMACS archives contain more than 50 benchmarks. We have chosen a subset of them for our experiments. Note that the main objective of this work is not to improve on the best known results for these instances. In order to achieve any improvement, local search alone may not be sufficient. It must be combined with special coloring techniques. In this work, we use these instances to study the behavior of our HLS procedures. For this purpose, we have chosen some 16 small and medium size (< 500 vertices) instances from different classes.

Frequency Assignment Problem

Our second test problem concerns a real world application: the frequency assignment problem which is a key application in mobile radio networks engineering. As we will see later, the basic FAP can be easily shown to be NP-complete.

The main goal of the FAP consists in finding frequency assignments which minimize the number of frequencies (or channels) used in the assignment and the electro-magnetic interference (due to the re-use of frequencies). The difficulty of this application comes from the fact that an acceptable solution of the FAP must satisfy a set of multiple constraints, some of these constraints being orthogonal. The most severe constraint concerns a very limited radio spectrum consisting of a small number of frequencies (usually about 60). This constraint imposes a high degree of frequency re-use, which in turn increases the probability of frequency interference. In addition to this frequency constraint, two other types of constraints must be satisfied to ensure communication of a good quality:

1. *Traffic constraints*: the minimum number of frequencies required by each station S_i to cover the communications of the station, noted by T_i.
2. *Frequency interference constraints* belong to two categories: 1. *Co-station constraints* which specify that any pair of frequencies assigned to a station must have a certain distance between them in the frequency domain; 2. *Adjacent-station constraints* which specify that the frequencies assigned to two adjacent stations must be sufficiently separated. Two stations are considered as adjacent if they have a common emission area.

About 60 instances were used in our experiments. Some of them are not only very large in terms of number of stations and in terms of number of interference constraints, but also very difficult to solve. The FAP instances we used in our experiments belong to three different sets which are specified below.

- **Test-Set-No.1** *Traffic constraints*: one frequency per station. Consequently, there is no co-station constraint. *Adjacent constraints*: frequencies assigned to two adjacent stations must be different.
- **Test-Set-No.2** *Traffic constraints*: two frequencies per station. *Co-station constraints*: frequencies assigned to a station must have a minimum distance of 3. *Adjacent constraints*: frequencies assigned to two adjacent stations must have a minimum distance of 1 or 2 according to the station.
- **Test-Set-No.3** *Traffic constraints*: up to 4 frequencies per station. *Co-station and adjacent constraints*: the separation distance between frequencies assigned to the same station or adjacent stations varies from 2 to 4.

In fact, Test-Set-No.1 corresponds to the graph-coloring problem. To see this, frequencies should be replaced by colors, stations by vertices and adjacent constraints by edges. Finding an optimal, i.e., an interference-free, frequency assignment using a minimum number of distinct frequencies is equivalent to coloring a graph with a minimum number of colors.

In order to apply our HLS procedures, the FAP will be encoded as a constraint (optimization) problem $< X, D, C, f >$ such that

- $X = \{L_1, L_2, ..., L_{NS}\}$ where each L_i represents a list of T_i frequencies required by the i^{th} station and NS is the number of stations in the network,
- D is the set of the NF integers representing the NF available frequencies,
- C is the set of co-station and adjacent-station interference constraints,
- f is the number of frequencies used to obtain conflict-free assignments.

It is easy to see that with this encoding, a frequency assignment has the length of $\sum_{i=1}^{NS} T_i$. Fig.1 gives an example where the traffic of the three stations is respectively 2, 1 and 4 frequencies and $f_{i,j}$ represents the j^{th} frequency value of the i^{th} station S_i.

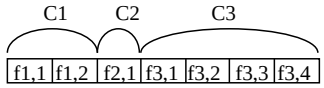

Fig. 1. FAP encoding

Solving the FAP consists in finding assignments which satisfy all the interference constraints in C and minimize NF, the number of frequencies used. To minimize NF, we use the same technique as that explained in the last section.

3.2 Comparison of Strategies for Neighborhood Examination

We have chosen to compare four combinations of heuristics: *var.1 random/val.2 conflict-random, var.2 conflict-random/val.2 min-conflicts, var.2 conflict-random/ val.3 stochastic-best-one, var.2 conflict-random/val.5 probabilistic-improvement.* The main reason for choosing these strategies is to have a good sample which combines two important aspects of a search strategy: randomness and guideness.

The *var.1 random/val.1 random* combination, which represents a random search strategy, was also tested. The results of this strategy were so bad that we have decided to omit them from the comparison. The probability p used in *var.2/val.3* is respectively fixed at 0.1 for the COL and 0.2 for the FAP. The probability p used in *var.2/val.5* is fixed at 0.05 for both problems.

Table 1 shows the comparative results of these four strategies for 6 COL instances (3 structured graphs $le450\_15[c-d].col$, $flat\_28\_0.col$ and 3 random graphs). For each instance and each strategy, we give the mean number of colors (Colors) over 5 runs and the mean number of evaluations (Eval.) in thousands, needed to find a coloring. These two criteria reflect respectively the quality of a solution and the efficiency of a strategy.

Table 1. Comparison of heuristics for COL

Problems	Runs	var1/val2		var2/val2		var2/val3		var2/val5	
		Colors	Eval.	Colors	Eval.	Colors	Eval.	Colors	Eval.
R125.1c.col	5	46.00	1141	47.80	2939	47.60	128	46.00	653
R250.5.col	5	72.60	9800	70.60	8500	70.00	5600	70.80	1420
DJSC250.5.col	5	33.20	2000	34.80	2100	31.60	1240	32.00	1280
le450_15c.col	5	22.60	6173	>25	>5100	21.60	2574	21.80	2907
le450_15d.col	5	22.60	3789	>25	>5100	21.40	3094	21.20	3906
flat300_28_0.col	5	35.00	20368	>38	>7100	34.00	2832	34.80	4486

From Table 1, we observe that compared with the other strategies, *var.2 conflict-random/val.2 min-conflicts* gives the worst results. In fact, for 5 out of 6 instances, it requires more colors and more evaluations than the others to find colorings. In particular, it has serious problems coloring the 3 structured graphs even with up to 10 colors more than the minimum. *var.1 random/val.2 conflict-random* is a little better than *var.2/val.2* for 5 instances, but worse than *var.2 conflict-random/val.3 stochastic-best-one* and *var.2 conflict-random/val.5 probabilistic-improvement*. Finally, the results of *var.2/val.3* and *var.2/val.5* are similar with a slightly better performance for the first: *var.2/val.3* performs a little better than *var.2/val.5* for 4 instances. Therefore, for the coloring problem, it seems that the following relation holds: $(var.2/val.3 \approx var.2/val.5) > var.1/val.2 > var.2/val.2$, where $\approx$ and $>$ mean respectively "comparable" and "better than" in terms of the solution-quality/efficiency. This relation was confirmed when we tried to solve other COL instances.

Table 2 shows the comparative results of these four strategies for 5 FAP instances. Each instance is specified by three numbers $nf.tr.nc$ which are respectively the lower bound of the number of distinct frequencies necessary to have an interference-free assignment, the sum total of the traffic (the number of integer variables) of all stations, and the number of co-station and adjacent-station interference constraints. For example, 16.300.12370 means that this instance has a total traffic of 300 and 12,370 interference constraints, and requires at least 16 frequencies to have an interference-free assignment. The same criteria as for the COL are used except that "Colors" criterion is replaced by the mean number of

the frequencies (NF) for having interference-free assignments.

From Table 2, we see once again that *var.2 conflict-random/val.2 min-conflicts* is the worst strategy. With even 10 frequencies more than the lower bound (16), it cannot find a conflict-free assignment for *16.300.12370*. For the other instances, it requires at least 3 to 6 frequencies more than the lower bound. At the other extreme, we notice that *var.2 conflict-random/val.3 stochastic-best-one* dominates the others for the 5 instances. Finally, the performance of *var.2 conflict-random/val.5 probabilistic-improvement* is between that of *var.1 random/val.2 conflict-random* and *var.*2/*val.*3. Therefore, for the FAP, it seems that the following relation holds: *var.*2/*val.*3 > *var.*2/*val.*5 > *var.*1/*val.*2 > *var.*2/*val.*2. This relation was confirmed when we tried to solve other FAP instances.

Table 2. Comparison of heuristics for FAP

Problems	Runs	var1/val2		var2/val2		var2/val3		var2/val5	
		NF	Eval.	NF	Eval.	NF	Eval.	NF	Eval.
8.150.3308	5	8.0	916	14.00	1229	8.00	47	8.00	50
15.300.8940	5	16.00	1769	18.00	741	15.00	1804	15.40	2016
16.150.3203	5	17.80	1246	21.80	6731	16.75	10236	17.60	2638
16.300.12370	5	18.75	5978	>26	>1350	17.00	2535	19.20	3120
30.600.45872	5	30.00	6513	33.20	11080	30.00	570	30.00	1700

Based on the data in Tables 1 and 2, we can make several remarks. First, we notice that *var.2 conflict-random/val.3 stochastic-best-one* and *var.2 conflict-random/val.5 probabilistic-improvement* turn out to be winners for both problems compared with *var.1 random/val.2 min-conflicts* and *var.2 conflict-random/val.2 min-conflicts*. As for most procedures of heuristic search, there is no theoretical justification for this. However, intuitive explanations help to understand the result. The tested instances represent hard problems and may contain many deep local optima. For a search procedure to have a chance of finding a solution, it must be not only able to converge efficiently towards optima, but also able to escape from local optima. Both *var.2/val.3* and *var.2/val.5* have this capacity thanks to deteriorative moves authorized by *var.2/val.3* and random moves in *var.2/val.5*. On the contrary, *var.1 random/val.2 min-conflicts* and *var.2 conflict-random/val.2 min-conflicts*, once they have reached a local optimum, is trapped since deteriorative moves are forbidden. Although they both allow side-walk moves by taking values that do not change the value of the evaluation function, this is not sufficient to escape from local optima.

If we trace the evolution of the evaluation function, i.e. the number of unsatisfied constraints as a function of the number of iterations, we observe that all four strategies are able to reduce rapidly the number of unsatisfied constraints after a relatively small number of iterations (descending phase). The difference between these strategies appears during the following phase. In fact, for *var.2/val.3* and *var.2/val.5*, the search, after the descending phase, goes into an oscillation phase composed of a long series of up-down moves, while for *var.1/val.2* and *var.2/val.2*, the search stagnates at plateaus.

In principle, any search strategy must conciliate two complementary aspects:

exploitation and exploration. Exploitation emphasizes careful examinations of a given area while exploration encourages the search to investigate new areas. From this point of view, we can say that *var.2/val.3* and *var.2/val.5* manage to exploit and explore correctly the search space by balancing randomness and guideness. On the contrary, due to the deterministic nature of the *min-conflicts* heuristic, *var.1/val.2* and *var.2/val.2* focus only on the aspect of exploitation.

3.3 Comparisons with Other Methods

This section lists the best results of the HLS procedures with *var.2/val.3* and *var.2/val.5* for the COL and the FAP. The controlling probability p used by *var.2/val.3* varied between 0.1 and 0.2. The probability used by *var.2/val.5* was around 0.05. Whenever possible, the results are compared with those obtained by other methods: Tabu search for the COL, SA, constraint programming (CP) and heuristic coloring algorithms (HCA) for the FAP.

Results are based on 1 to 10 independent runs according to the difficulty of the instance. For each run, a HLS procedure begins with k colors/frequencies (usually 10 colors/frequencies more than the best known value) and tries to find a conflict-free solution for the underlying CSP. For each given color/frequency, $t = 1, 2, 3$ tries are authorized, i.e. if the search cannot find a conflict-free solution within t tries, the current run is terminated. If a conflict-free solution is found (within t tries), the number of colors/frequencies is decreased and the search continues. The maximum number of iterations (moves) is fixed at 50,000 to 500,000 for each try in each run.

Three criteria are used for reporting results: the minimum and the mean number of colors/frequencies (NC/NF), and the mean time for finding a conflict-free solution. The first two criteria reflect the quality of solutions while the third reflects solving efficiency. In order to better assess the robustness of HLS, we also give in brackets how many times a solution with the minimum number of colors/frequencies has been found. The time (on a SPARCstation 10) is the total user time for solving an instance, including the time for solving the intermediate CSPs and the time for failed tries (up to 3 tries for each run). Note that timing is given here only as a rough indication.

Table 3 shows the results of the HLS procedures for some DIMACS benchmarks and two classes ($g100.5.col$ and $g300.5.col$) of 25 random graphs taken from [4]. The instances are specified as follows. For random graphs $gxxx.y.col$ and $Rxxx.y.col$, xxx and y indicate respectively the number of vertices and the density of the graph. For Leighton's graphs $lexxx\_yy[a-d].col$ and structured graphs $flat300\_yy\_0.col$, yy is the chromatic number. $DSJC.1000.5.col(res)$ is the *residual graph* (200 vertices and 9,633 edges) of $DSJC.1000.5.col$ (1,000 vertices and about 500,000 edges). This residual graph was obtained by eliminating 61 independent sets and all the related edges [4]. Table 3 also shows the results of Tabu search (using a faster SuperSPARC 50 machine) [3].

The top part of Table 1 presents results for graphs having up to 300 vertices. From the data in Table 1, we notice that for the 20 $g100.5.col$ instances, HLS finds the same result as that of Tabu search while for the 5 $g300.5.col$ instances,

it needs en average 1.3 more colors. For the $Rxxx.y.col$ family, HLS manages to find a best known coloring for 5 out of 6 instances, but has difficulty to color $R125.5.col$ for which it requires 2 colors more than Tabu. For the three $flat300\_yy\_0.col$ instances, HLS obtains the same results as Tabu search for 2 out of 3 instances. For $flat300\_26\_0.col$, HLS finds an optimal coloring for one out of 5 runs. For $DSJC.1000.5.col(res)$, HLS manages to color the graph with 24 colors in about half an hour (instead of the best known value of 23 obtained with Tabu after about 20 hours of computing). With 23 colors, HLS usually leaves 1 or 2 unsatisfied constraints at the end of its search.

Table 3. HLS performance for COL

Problems	Edges	HLS				Tabu			
			NC				NC		
		Runs	Min.(nb)	Ave.	T[sec.]	Runs	Min.	Ave.	T[sec.]
g100.5.col (20 inst.)	$\simeq$ 2500	1	14(1)	14.95	275.3	1	-	14.95	$\simeq$ 9.5
g300.5.col (5 inst.)	$\simeq$ 22000	1	34(1)	34.80	4793	1	-	33.50	$\simeq$ 353
R125.1.col	209	10	5(10)	5.00	0.5	10	5	5.00	0
R125.1c.col	7501	10	46(10)	46.00	129	10	46	46.00	4.1
R125.5.col	3838	10	37(1)	38.00	187	5	35	35.60	1380
R250.1.col	867	10	8(10)	8.00	2	10	8	8.00	0
R250.1c.col	30227	10	64(8)	64.20	2946	10	64	64.00	108
R250.5.col	14849	3	69(1)	69.75	6763	5	69	69.00	1664
flat300_20_0.col	21375	5	20(5)	20.00	1997	10	20	20.00	40
flat300_26_0.col	21633	5	26(1)	31.40	6710	3	26	26.00	8100
flat300_28_0.col	21695	5	33(5)	33.00	2402	3	33	33.00	4080
DSJC.1000.5.col(res)	9633	5	24(5)	24.00	1531	5	23	23.00	68400
le450_15a.col	8168	5	16(5)	16.00	354	10	15	15.00	248
le450_15b.col	8169	5	16(5)	16.00	273	10	15	15.00	248
le450_15c.col	16680	5	15(1)	16.00	4376	10	16	16.00	268
le450_15d.col	16750	5	15(2)	15.60	3990	10	16	16.00	791
le450_25a.col	8260	5	25(5)	26.00	61	5	25	25.00	4.0
le450_25b.col	8263	5	25(4)	25.60	27	5	25	25.00	3.9

The lower part of Table 1 presents the results for 6 Leighton graphs. We see that HLS finds an optimal coloring for 4 out of 6 instances. For $le450\_15c.col$ and $le450\_15d.col$, HLS requires one less color than Tabu. For $le450\_15a.col$ and $le450\_15b.col$, the reverse is true. It should be mentioned that, in order to color these graphs and more generally any graph having more than 300 vertices, Tabu uses the technique of independent sets mentioned above to produce first a much smaller residual graph which is then colored.

Finally, we make a general remark about the HLS procedures used to obtain the above results. The main goal of this work is to study the behavior of various heuristics, but not to improve on the best results for the coloring problem (In fact, this second point constitutes another ongoing work). Consequently, the HLS procedures used are voluntarily general and do not incorporate any specialized technique for the COL. With this fact in mind, the results of the HLS procedures may be considered to be very encouraging.

Table 4 shows the best results for 12 FAP instances obtained with HLS procedures combined with a technique for handling co-station constraints [2]. These instances are taken from a series of 60 instances and represent some of the hardest problems. It should be remembered that each instance *nf.tr.nc* is specified by the lower bound for the number of frequencies, the sum total of the traffic, and the number of interference constraints. As we can see from the table, some instances are very large and have a high density of constraints. Table 4 also shows the best results of SA, CP (ILOG-Solver) and HCA, all reported in [15]. These procedures have been run on HP Stations which are considered to be at least three times faster than the SPARCstation 10 we used. A minus "-" in the table means that the result is not available.

Table 4. HLS performance for FAP

Problems	HLS					HCA	CP		SA	
		NF								
	Runs	Min.(nb)	Ave.	Eval.	T[sec.]	NF	NF	T[sec.]	NF	T[sec.]
8.150.2200	10	8(10)	8.00	812	639	8	8	7200	8	509
15.300.8940	10	15(10)	15.00	1326	1606	20	17	3600	15	4788
15.300.13400	10	15(10)	15.00	2557	3600	27	25	1560	15	2053
16.150.3203	10	**16(1)**	16.9	1070	658	19	18	14400	**17**	1744
16.150.3323	10	17(8)	17.2	3246	2283	19	19	7200	17	1383
30.300.13638	10	30(10)	30.00	18	25	30	30	1	30	1558
30.600.47852	2	**30(1)**	30.50	46666	122734	47	46	4800	**36**	5309
40.335.11058	2	43(2)	43.00	9434	17823	-	-	-	-	-
40.966.35104	10	45(10)	45.00	707	1341	-	-	-	-	-
60.600.47688	10	60(10)	60.00	4234	9288	60	-	-	60	858
60.600.45784	10	60(10)	60.00	2039	3981	60	-	-	60	516

In Table 4, we notice first that HCA gives the worst results for all the solved instances; it requires up to 17 frequencies more than the lower bound. The results of CP are a little better than HCA for 6 out of 7 problems. It is interesting to see that instances which are difficult for HCA remain difficult for CP. On the contrary, the results of HLS and SA are much better than those of HCA and CP: HLS (SA) finds an optimal assignment for 8 (6) instances. In general, the harder the problem to be solved, the bigger the difference: for *15.300.13400* and *30.600.47852*, there is a difference of more than 10 frequencies. Note finally that HLS improves on the result of SA for two instances (in bold). In particular, for *30.600.47852*, HLS finds a conflict-free assignment with only 30 frequencies (36 for SA). This is rather surprising given the similar results of these two approaches for the other instances. The computing time for HLS and SA is generally similar to obtain solutions of the same quality. It is difficult to compare the computing time with CP since they give solutions which are too different.

To sum up, HLS gives the best result for the 12 hardest FAP instances. This remains true for 48 other instances which have been solved, but not reported here. However, we notice that for easier instances, all the methods behave similarly. Moreover, CP and HCA may be faster for some easy instances.

4 Conclusions & Future Work

In this paper, we have presented a class of local search procedures based on heuristics for choosing variables/values. The combinations of these heuristics give different strategies for examining a very general neighborhood. These heuristics can be applied to a wide range of problems.

Four representative combinations of these heuristics out of fifty possibilities have been empirically evaluated and compared using the graph-coloring problem and the frequency assignment problem. Two strategies turn out to be more efficient: *var.2 conflict-random/val.3 stochastic-best-one* and *var.2 conflict-random/val.5 probabilistic-improvement*. In essence, these two strategies are able to find a good balance between randomness and guideness, which allows them to explore and exploit correctly the search space. The controlling probability p used by these two strategies plays an important role in their performance and should be empirically determined. In our experiments, we have used values ranging from 0.1 to 0.2 for *var.2/val.3* and values around 0.05 for *var.2/val.5*. More work is needed to better determine these values. Moreover, the possibility of a self-adaptive p is also worth studying.

We also found that *var.1 random/val.2 min-conflicts*, and especially *var.2 conflict-random/val.2 min-conflicts* are rather poor strategies for both COL and FAP. In fact, these two strategies are too deterministic to be able to escape from local optima. It is interesting to contrast this finding with the work concerning the *min-conflicts* heuristic [13].

To further evaluate the performance of these heuristic LS procedures, they were tested on more than 40 COL benchmarks and 12 hard FAP instances. Although they are not especially tuned for the coloring problem, the HLS procedures give results which are comparable with those of Tabu search for many instances. At the same time, we noticed that HLS alone, like many other pure LS procedures, has difficulty coloring very large and hard graphs. For the FAP problem, the results of HLS on the tested instances are at least as good as those of simulated annealing, and much better than those obtained with constraint programming and heuristic coloring algorithms.

Currently, we are working on several related issues. At a practical level, we want to evaluate other combinations not covered in this paper. Secondly, we are developing specialized coloring algorithms combining the general heuristics of this paper and well-known coloring techniques. Indeed, in order to color hard graphs, all efficient algorithms use specialized techniques. It will be very interesting to see if the combination of our heuristics with these kinds of techniques can lead to better results.

At a more fundamental level, we try to identify the characteristics of problems which may be efficiently exploited by a given heuristic. This is based on the belief that a heuristic has a certain capacity to exploit special structures or characteristics of a problem. Thus, the heuristic may have thus some "favorite" problems. A long-term goal of the work is to look for a better understanding of the behavior of LS heuristics for solving problems and answering such fundamental questions as when and why a heuristic works or does not.

Acknowledgments

We would like to thank A. Caminada from the CNET for his assistance, P. Galinier for useful discussions and the referees for their comments on the paper.

References

1. D.H. Ackley, A connectionist machine for genetic hillclimbing. Kluwer Academic Publishers, 1987.
2. R. Dorne and J.K. Hao, "Constraint handling in evolutionary search: a case study about the frequency assignment", Submitted to the 4th Intl. Conf. on Parallel Problem Solving from Nature (PPSN'96), Berlin, Germany, Sept. 1996.
3. C. Fleurent and J.A. Ferland, "Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability". in D.S. Johnson and M.A. Trick (eds.) "2nd DIMACS Implementation Challenge" (to appear).
4. C. Fleurent and J.A. Ferland, "Genetic and hybrid algorithms for graph coloring", to appear in G. Laporte, I. H. Osman, and P. L. Hammer (eds.), Special Issue Annals of Operations Research on Meta-heuristics in Combinatorial Optimization.
5. E.C. Freuder and R.J. Wallace, "Partial constraint satisfaction", Artificial Intelligence, Vol.58(1-3), pp21-70, 1992.
6. F. Glover and M. Laguna, "Tabu Search", in C. R. Reeves (eds.) Modern Heuristics for Combinatorial Problems, Blackwell Scientific Publishing, Oxford, Great Britain.
7. P. Hensen and B. Jaumard, Algorithms for the maximum satisfiability problem, Computing Vol.44, pp279-303, 1990.
8. A. Hertz and D. de Werra, "Using Tabu search techniques for graph coloring". Computing Vol.39, pp345-351, 1987.
9. D.S. Johnson, C.R. Aragon L.A. McGeoch and C. Schevon, "Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning". Operations Research, Vol.39(3), 378-406, 1991.
10. B.W. Kernighan and S. Lin, "An efficient heuristic for partitioning graphs", Bell System Technology Journal, Vol.49, 1970.
11. S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, "Optimization by simulated annealing", Science No.220, 671-680, 1983.
12. S. Lin and B.W. Kernighan, "An efficient heuristic for the traveling-salesman problem", Operations Research, Vol.21, pp498-516, 1973.
13. S. Minton, M.D. Johnston and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems", Artificial Intelligence, Vol.58(1-3), pp161-206, 1992.
14. C.H. Papadimitriou and K. Steiglitz, "Combinatorial optimization - algorithms and complexity". Prentice Hall, 1982.
15. A. Ortega, J.M. Raibaud, M.Karray, M.Marzoug, and A.Caminada, "Algorithmes de coloration des graphes et d'affectation des fréquences". TR CNET, N-T/PAB/SRM/RRM/4353, August 1995.
16. B. Selman, H.J. Levesque and M. Mitchell, "A new method for solving hard satisfiability problems". Proc. of AAAI-92, San Jose, CA, pp.440-446, 1992.
17. B. Selman and H.Kautz, "Domain-independent extensions to GSAT: solving large structured satisfiability problems". Proc. of IJCAI-93, Chambery, France, 1993.
18. E. Tsang, "Foundations of constraint satisfaction", Academic Press, 1993.

This article was processed using the LaTeX macro package with LLNCS style