

Heuristic Search Methods for Combinatorial Programming Problems *

Jacques A. Ferland,
Département d'informatique et de recherche opérationnelle,
Université de Montréal, C.P. 6128, Succ. Centre-Ville,
Montréal(Québec) Canada H3C 3J7

and

Daniel Costa,
Distribution Operations Development
Management Services – Logistics
Nestec Ltd
1800 Vevey, Switzerland

March 2001

Abstract

In this paper, we summarize most of the basic heuristic search methods used to solve combinatorial programming problems. We start with two constructive methods : greedy method, and greedy randomized adaptive search procedure (GRASP). Then five neighborhood (local) search techniques are reviewed : descent method, tabu search, exchange procedure, simulated annealing, and threshold accepting method. Next we summarize three different population based methods : genetic algorithm, scatter search, and ant algorithm. We conclude with some examples of hybrid procedures combining two or more basic methods. The basic notions included in these methods are illustrated with the graph coloring problem.

*This research was supported by NSERC grant (OGP0008312), from Canada.

1 Introduction

In this paper we summarize most of the basic heuristic search methods used to solve combinatorial programming problems. Sometimes, the authors take advantage of the specific problem that they are solving to slightly modify a basic heuristic method in order to generate a more efficient variant.

In general, combinatorial problems are easy to formulate but difficult to solve because they are formulated as integer programming problems. The traveling salesman problem and the graph coloring problem are often mentioned as such problems. Since these problems belong to the class of *NP*-hard problems, then we often rely on heuristic search methods to deal with them.

A heuristic search method can be seen [35] as a procedure taking advantage of the problem structure in order to identify a good solution within a reasonable amount of computing time. Hence, the efficiency of a heuristic search is specified in terms of the quality of the solution generated and of the computer time required.

In several situations, it is more appropriate to use heuristic methods rather than “exact methods” generating optimal solutions. Indeed, looking for an “optimal” solution might be totally unrealistic given the size of the mathematical model associated with a real world application and given the lack of precision of the data. In some cases, the speed of exact methods is slower than that of a good heuristic method. Furthermore the end users may feel uncomfortable with exact methods because they may look like black boxes over which they do not have any control, and because the presence of specialized personnel is required to adjust properly several parameters.

Heuristic methods are in general more intuitive and more accessible to the end user. Even if parameters have to be adjusted, the end users can in general foresee the effects of modifying the parameters. Furthermore, these methods are closer to the approach that the end users have in mind.

A heuristic method is easier to adapt to a specific problem, and to be combined with other methods to generate efficient ad hoc procedures.

In the following sections we present the methods in the general context of solving the problem

$$\text{Min}_{x \in X} f(x).$$

We also use the graph coloring (or the ξ -coloring) [26] problem to illustrate the basic notions included in these methods. Let $G = (V, E)$ denotes a graph. One formulation of the graph coloring problem is as follows:

Using a specified number ξ of colors, determine a color for each vertex $i \in V$ in order to minimize the number of pairs of adjacent vertices having the same color.

For all $i \in V$ and $j, j = 1, 2, \dots, \xi$, let

$$x_{ij} = \begin{cases} 1, & \text{if vertex } i \in V \text{ has color } j \\ 0, & \text{otherwise.} \end{cases}$$

Then the problem can be formulated as follows:

$$\begin{aligned} & \text{Min } \sum_{j=1}^{\xi} \sum_{(i,\ell) \in E} x_{ij} x_{\ell j} \\ & \text{Subject to } \sum_{j=1}^{\xi} x_{ij} = 1, \quad i \in V \\ & \quad \quad \quad x_{ij} = 0 \text{ or } 1, \quad i \in V, \quad j = 1, 2, \dots, \xi. \end{aligned}$$

Note that the graph coloring problem be seen as an assignment-type problem [9]:

Given $|V|$ items (vertices) and ξ resources (colors), the problem is to determine an assignment of the items to the resources optimizing an objective function.

In general, the objective function includes penalty terms associated with additional constraints of the original problem.

In Section 2, we start with two constructive methods : the greedy method and the GRASP (Greedy Randomized Adaptive Search Procedure). Section 3 includes five neighborhood (local) search techniques (NST) : descent method, tabu search method, exchange procedure, simulated annealing, and threshold accepting method. The last four methods include strategy to move out of local minima in order to search more extensively the feasible domain. Three different strategies (variable neighborhood search approach, intensification, and diversification) to improve NST are described in Section 4. In Section 5, we summarize three different population based techniques : genetic algorithm, scatter search, and ant algorithm. These techniques include an evolutionary mechanism based on a collective process and an individual process to transform populations of solutions. Section 6 includes examples of strategy to combine different heuristic methods in order to generate new hybrid methods more efficient for some problems. This paper ends with some concluding remarks in Section 7.

2 Constructive Approach

In these methods, the values of the variables are determined sequentially. At each iteration, a variable is selected, and its value is determined. The value of a variable is never modified

afterward.

In the *greedy* method, the next variable to be fixed and its value are selected to optimize the objective function given the values of the variables already fixed, and without any looking ahead perspective. For instance, in the graph coloring problem the vertices are ordered in decreasing order of their degree. (Note that the degree of a vertex is equal to the number of adjacent vertices.) The vertices are selected sequentially in that order starting with a vertex with the largest degree. For each vertex, we select a color in order to reduce the number of pairs of adjacent vertices already colored with the same color.

In the GRASP (*Greedy Randomized Adaptive Search Procedure*) [13] method, the next variable to be fixed is selected randomly among those inducing the smallest increase. To be more explicit, suppose that

$$\bar{J} = \{j : x_j \text{ is not fixed yet}\}$$

and let δ_j denote the increase induced by the best value that x_j can take, $j \in \bar{J}$. If we denote

$$\delta^* = \min_{j \in \bar{J}} \{\delta_j\},$$

and $\alpha \in [0, 1]$, then select randomly

$$\bar{j} \in \{j \in \bar{J} : \delta_j \leq \frac{1}{\alpha} \delta^*\},$$

and fix the value of $x_{\bar{j}}$. For instance, in the graph coloring problem, the next vertex to be colored and its color are selected as follows:

- let $\bar{I} = \{i : \text{index of a vertex that is not colored yet}\}$
- let δ_i the number of additional pairs of adjacent vertices having the same color if the best color for x_i is selected
- let $i^* \in \bar{I}$ such that $\delta_{i^*} = \min_{i \in \bar{I}} \{\delta_i\}$, and let $\alpha \in [0, 1]$
- the next vertex \bar{i} to be colored is selected randomly in the set $\{i \in \bar{I} : \delta_i \leq \frac{1}{\alpha} \delta^*\}$.

Note that if $\alpha = 0$, then the selection is totally random, and if $\alpha = 1$, then the selection is greedy.

Constructive methods are fast and very simple. Unfortunately, their myopic behavior may reduce the quality of the solution generated.

3 Neighborhood (Local) Search Techniques

At each iteration of an iterative procedure we move from a current solution $x \in X$ to a new solution $x' \in X$ until some solution $x^* \in X$ is reached where x^* is acceptable according to some criterion. Neighborhood Search Techniques (NST) are iterative procedures where the next solution $x' \in N(x)$, and where $N(x)$ is the *neighborhood* of the current solution x .

In general, the neighboring solutions $x' \in N(x)$ are generated by applying modifications $m \in M$ to the solution x . The set of modifications M is specific for each type of problems. Referring to [41] we denote

$$x' = x \oplus m, \quad m \in M.$$

Hence the neighborhood $N(x)$ of x is defined as follows:

$$N(x) = \{x' \in X : x' = x \oplus m \text{ for some } m \in M\}$$

To illustrate the notion of neighborhood, consider the graph coloring problem. Recall that a feasible solution x for this problem is such that a color $j(i)$ is determined for each vertex i . Hence, for each $i \in V$,

$$\begin{aligned} x_{ij(i)} &= 1 \\ x_{ij} &= 0, \quad j = 1, 2, \dots, \xi, \quad j \neq j(i). \end{aligned}$$

A modification $m \in M$ is determined by selecting a vertex $i \in V$ and by modifying its color from $j(i)$ to a different color j . The corresponding neighboring solution $x' = x \oplus m$ is specified as follows:

$$\begin{aligned} x'_{pq} &= x_{pq} \quad p \in V, p \neq i; j = 1, 2, \dots, \xi \\ x'_{iq} &= x_{iq} \quad q = 1, 2, \dots, \xi, q \neq j(i), j \\ x'_{ij(i)} &= 0 \\ x'_{ij} &= 1. \end{aligned}$$

Thus, if we denote m by $[i, j]$, then it follows that

$$M = \{[i, j] : i \in V; j = 1, 2, \dots, \xi, j \neq j(i)\}.$$

Each different technique has its specific way of selecting the next current solution $x' \in N(x)$. The simplest one is the *Descent Method*. The main drawback of this method is the fact that,

in general, it stops at a local minimum. The *Tabu Search Method* allows to move out of a local minimum to search more extensively the feasible domain X . The *Exchange Procedure* allows to use an enumeration tree search method at a local minimum in order to generate a new initial solution for the Descent Method.

Another group of methods select randomly a solution in $N(x)$ and accept it as the new current solution according to different criteria. *Simulated Annealing* and *Threshold Accepting techniques* belong to this group.

3.1 Descent Method

At each iteration, the best $x' \in N(x)$ (i.e., $x' = \arg \min_{x \in N(x)} f(x)$) is selected and the procedure stops when $f(x) = f(x')$. The procedure is summarized in Figure 3.1. Clearly, the descent method stops at the first local minimum reached from the starting solution. More elaborate search techniques have been introduced to avoid being trapped in a local minimum.

Initialization :

Select an initial solution $x^o \in X$

Let $x := x^o$ and stop := false

While not stop

Determine $x' \in N(x)$ such that $x' := \arg \min_{z \in N(x)} \{f(z)\}$

If $f(x') \geq f(x)$

then stop := true

else $x := x'$

x is the solution generated

Figure 3.1 : Descent Method.

3.2 Tabu Search Method

This solution procedure was developed independently by Glover [20] and Hansen [25]. At each iteration, the best solution (different from the current one) x' in a subset $V \subset N(x)$ is selected. The subset V can be selected randomly from $N(x)$, or it may include more promising elements from $N(x)$ to improve the objective function (interesting neighborhood) when the structure of the problem allows to identify such elements.

As long as x' is better than x , the behavior of the procedure is the same as the descent method. Otherwise, moving to x' as the next current solution induces no improvement or a deterioration of the objective function (the least one among the points in V), but it allows to move out of the local minimum, and hence to search more extensively the feasible domain X .

Now, since the value of f is not monotone from one iteration to the next, a safeguard against cycling is required. This is done by including some kind of memory into the process in order to forbid returning to a solution already visited. Hence *Tabu lists* are used to remember attributes or characteristics $t_i(m)$ of moves $m \in M$ that were recently used to generate new current solutions. For instance, since we assume that each move in M is reversible (i.e., if $m \in M$, then there exists another move $m^{-1} \in M$ such that $(x \oplus m) \oplus m^{-1} = x$), then one of the Tabu lists may include the reverse moves \bar{m}^{-1} of the moves \bar{m} used recently. Thus, when p lists T_i ($i = 1, 2, \dots, p$) are used, moving from x to $x' = x \oplus m$ is forbidden whenever $t_i(m) \in T_i$ for some $i = 1, 2, \dots, p$.

The Tabu lists T_i are cyclic inasmuch as the move m remains Tabu with respect to attribute $t_i(m)$ for a fixed number n_i of iterations. Hence at each iteration the Tabu list T_i is updated by including $t_i(m)$ in T_i , and if $|T_i| \geq n_i$, then the oldest element of T_i is eliminated from it, $i = 1, 2, \dots, p$.

The sizes of the tabu lists T_i are important parameters for the efficiency of the procedure, but it is not easy to determine the best sizes. Let $|T_i|$ denotes a base value for the size of T_i . Sometimes we use a variable Tabu list size [40] $|\tilde{T}_i|$ at each iteration where

$$t_{min} \leq |\tilde{T}_i| \leq |T_i|,$$

and $t_{min} = \lfloor 0.8|T_i| \rfloor$. (Note that $\lfloor a \rfloor$, the floor value of a , is equal to the largest integer smaller than or equal to a .) Then \tilde{T}_i includes the last $|\tilde{T}_i|$ elements of T_i .

Since in most cases the attributes used to specify the Tabu lists refer to characteristics of the move in M , they may forbid to move to a solution that was never visited and that might be very good now or in a near future. Hence we need an *aspiration criterion* to override the Tabu status of some solutions during the process. Several aspiration criteria have been proposed in the literature, but one of the most commonly used is specified in terms of the best solution x^* encountered so far:

z is included in the subset $V \subset N(x)$ of the candidate solutions to replace the current solution even if z is tabu, whenever $f(z) < \frac{1}{\alpha}f(x^*)$ (where the parameter $\alpha \in (0, 1]$).

The stopping criteria for the method are usually specified in terms of a maximum number of iterations and in terms of a maximum number of consecutive iterations where the objective function does not improve. The Tabu search procedure is summarized in Figure 3.2.

Initialization :

Select an initial solution $x^o \in X$

Let $T_i = \phi, i = 1, 2, \dots, p$ {Tabu lists initially empty}

Let iter:= 0; niter := 0

Let $x := x^o; x^* := x^o; \text{stop} := \text{false}$

While not stop

iter := iter +1 ; niter := niter +1

Determine a subset $V \subset N(x)$ of solutions $z := x \oplus m$ satisfying at least one of the two following conditions:

- $t_i(m) \notin T_i, i = 1, 2, \dots, p$
- $f(z) < \frac{1}{\alpha}f(x^*)$

Determine $x' \in V$ such that $x' := \arg \min_{z \in V} \{f(z)\}$

$x := x'$

If $f(x) < f(x^*),$ **then** $x^* := x,$ and niter := 0

If iter = itermax **or** niter = nitermax **then** stop := true

Update Tabu List $T_i, i = 1, 2, \dots, p$

x^* is the best solution generated

Figure 3.2 : Tabu Search Procedure.

3.3 Exchange Procedure

The Exchange Procedure [12] is in the spirit of Tabu Search to reach out of a local minimum. In order to do this, the procedure allows to use several non descent moves but these are monitored to limit their number and the deterioration of the objective function before reaching solution x' such that $f(x') < f(x)$. Hence the technique can be seen as a descent method where the neighborhood is enlarged to a more complex one $EN(x)$ whenever x is a local minimum. The procedure is summarized in Figure 3.3.

Note that one way to determine $x' \in EN(x)$ is to apply a truncated depth-first enumeration tree search where x is the root of the tree. The search is truncated to monitor the number of modifications used (i.e. the depth of the tree or the size of the neighborhood) and the level of deterioration allowed along the way before reaching a new solution x' such that $f(x') < f(x)$. In this variant x' is the first element z encountered in the enumeration such that $f(z) < f(x)$.

3.4 Simulated Annealing

Simulated Annealing is another iterative procedure allowing non descent moves. Kirkpatrick et al. [32] and Cerny [2] were the first to solve combinatorial optimization problems with this approach already used to simulate the evolution of an unstable physical system toward a thermodynamic stable equilibrium point at a fixed temperature (see [34]). At each iteration of this probabilistic technique, a solution x' is selected randomly in a subset $V \subset N(x)$. Then, x' replaces x as the current solution if $f(x') < f(x)$. But x' can also replace x as the current solution even if $\Delta f = f(x') - f(x) \geq 0$ according to a probability decreasing with Δf and the number of iterations already completed. More specifically, x' replaces x with probability $e^{-\Delta f/TP}$ where the parameter TP (referred to as the temperature factor) decreases with the number of iterations completed. The procedure summarized in Figure 3.4 is a version of the approach proposed by Johnson et al. in [29, 30].

In the variant, we complete several iterations using the same temperature TP . This temperature is modified when the number of trial solutions (*trials*) or when the number of times that the current solution is changed (*changes*) reaches threshold values SF or $coff$, respectively. The parameter $a \in (0, 1)$ is used to modified the temperature ($TP := aTP$). Two stopping criteria are used. The first is specified in terms of the number of iterations (*itermax*). To apply the second criterion, we keep track of the number of consecutive temperature values (*fcount*) where the ratio of the number of *changes* over the number of *trials* is smaller than a threshold value *mpc*. When *fcount* reaches a specified value *flimit*, the procedure stops.

Initialization :

Select an initial solution $x^o \in X$

Let $x := x^o$; stop := false; change := false

While not stop

While not change

Determine $x' \in N(x)$ such that $x' := \arg \min_{z \in N(x)} \{f(z)\}$

If $f(x') \geq f(x)$

then change := true

else $x := x'$

Determine $x' \in EN(x)$ such that $x' := \arg \min_{z \in EN(x)} \{f(z)\}$

If $f(x') \geq f(x)$

then stop := true

else $x := x'$, and change := false

x is the solution generated

Figure 3.3: Exchange Procedure.

Initialization :

Select an initial solution $x^o \in X$, and an initial temperature TP^o

Let $iter := 0$; $TP := TP^o$; $fcount := 0$

Let $x := x^o$; $x^* := x^o$; $stop := false$

While not stop

$Changes := 0$; $trials := 0$;

While $trials < SF$ **and** $changes < cof f$

Generate randomly $x' \in V \subset N(x)$

$\Delta f = f(x') - f(x)$

If $\Delta f < 0$

then $x := x'$, and $changes := changes + 1$

else Generate a random number $r \in (0, 1)$.

If $r < e^{-\Delta f/TP}$ **then** $x := x'$, and $changes := changes + 1$

If $f(x') < f(x^*)$ **then** $x^* := x'$, and $fcount := 0$

$trials := trials + 1$;

$TP := a \cdot TP$

$iter := iter + 1$

If $\frac{changes}{trials} < mpc$ **then** $fcount := fcount + 1$

If $iter \geq itermax$ **or** $fcount = flimit$ **then** $stop := true$

x^* is the best solution generated

Figure 3.4 : Simulated Annealing.

The efficiency of Simulated Annealing is closely related to the strategy used to modify (i.e. lower) the temperature. Simulated Annealing can be studied in the context of non homogeneous Markov chains, and interesting asymptotic convergence results can be derived under specific assumptions related to the strategies used to modify the temperature [1, 19, 24, 33]. Since these strategies are difficult to implement and are very time consuming, in practice we often prefer to use a strategy like the one in the variant of Figure 3.4.

3.5 Threshold Accepting

Threshold Accepting methods [9, 10] are iterative procedures that can be seen as deterministic variants of Simulated Annealing. At each iteration, a solution $x' \in V \subset N(x)$ is generated randomly. The decision to replace x by x' depends on the value of an auxiliary function $\gamma(x, x')$ and on a threshold value dr . Several variants can be specified according to different ways of specifying $\gamma(x, x')$ and dr . Figure 3.5 includes a general variant.

The stopping criteria are specified in terms of a maximum number of iterations and of a maximum number of consecutive iterations where the objective function does not improve.

The following sections include three different variants of the threshold accepting approach.

3.5.1 Standard Threshold Accepting Method

In this variant, the threshold valued dr is updated like the temperature TP is modified in Simulated Annealing; i.e., a fixed number of iterations is completed with each value of dr , and the value of the dr is updated using a parameter $a \in (0, 1)$ ($dr := a \cdot dr$). The function $\gamma(x, x')$ is

$$\gamma(x, x') = f(x') - f(x).$$

Hence x' replaces x if it is not deteriorating the objective function by more than a threshold value that is decreasing with the number of iterations.

3.5.2 The Great Deluge Method

The threshold value dr correspond to the level of water. The solution $x' \in V \subset N(x)$ replaces x as the current solution if and only if its value $f(x')$ is below the level of water, independently of the value $f(x)$. Hence

$$\gamma(x, x') = f(x'),$$

and the threshold value dr (level of water) decreases linearly according to a parameter δ

$$dr := dr - \delta$$

Initialization :

Select an initial solution $x^o \in X$, and an initial threshold value dr^o

Let $iter := 0$; $niter := 0$

Let $x := x^o$; $x^* := x^o$; $dr := dr^o$; $stop := false$

While not stop

$iter := iter + 1$; $niter := niter + 1$

Generate randomly $x' \in V \subset N(x)$

If $\gamma(x, x') < dr$ **then**:

$x := x'$, and

If $f(x) < f(x^*)$ **then** $x^* := x$, and $niter := 0$

If $iter = itermax$ **or** $niter = nitermax$ **then** $stop := true$

Update the value of dr

x^* is the best solution generated

Figure 3.5: Threshold Accepting.

3.5.3 Maximal Deterioration Method

In this variant, the solution $x' \in V \subset N(x)$ replaces x as the current solution if and only if its value $f(x')$ does not deteriorate the value $f(x^*)$ of the best solution generated so far by an amount larger than a threshold parameter μ . Hence

$$\gamma(x, x') = f(x'),$$

and the threshold value dr is updated according to $f(x^*)$ as follows:

$$dr := f(x^*) + \mu$$

where the threshold parameter μ takes usually a fixed value throughout the application of the method.

4 Improving Strategies

Several strategies have been proposed to improve the efficiency of neighborhood search techniques to generate better solutions. Even if most of them have been introduced in connection with the *Tabu search method*, these strategies can be applied with any of the procedures described in Section 3.

4.1 Intensification

The intensification strategy is used to search more extensively a promising region. One way to implementing such a strategy is to temporarily enlarge the neighborhood whenever the current solution induces a substantial improvement over the previous best known solution.

Another alternative is to return to the best known solution to restart the *NST* using a temporarily enlarged neighborhood, or using temporarily shorter Tabu lists.

4.2 Diversification

The diversification principle is complementary to the intensification. Its objective is to search more extensively the feasible domain by leading the *NST* to unexplored regions of the feasible domain. Numerical experiences indicate that it seems better to apply a “short” NST (with smaller values for *itermax* and *nitermax*) several times using different initial solutions rather than a “long” NST (with larger values for *itermax* and *nitermax*) only once.

One way of implementing a diversification is to embed the strategy within the *NST* itself by using a list of relatively good solutions whose neighborhoods have not been searched. At each iteration of the *NST*, the second best solution generated is compared with the worst one in the list. If it is better, then it is included in the list and the worst is eliminated from it. Whenever the rate of improvement of the best known solution decreases, then we move to the best solution in the list that becomes the current solution.

Other diversification strategies are used to generate new initial solutions in order to restart the *NST*. The most straightforward strategy of this category is the multistart approach where a *NST* technique is applied several times with different randomly generated initial solutions in order to search more extensively the feasible domain for the best local optimum. The multistart approach search in many valleys of the feasible domain without focusing on promising regions. But since the local minima are often concentrated in few small regions, it is worthwhile to modify the multistart approach to explore increasingly large neighborhoods of the best known solution x^* . The *Variable Neighborhood Search Approach* [26] summarized in Figure 4.1 is such a modification.

A set of neighborhood structures N_k , $k = 1, 2, \dots, \bar{k}$, has to be pre-specified. (For instance, the neighborhoods can be nested such that $N_k(x) \subset N_{k+1}(x)$.) At each iteration, a local minimum x'' is generated with a *NST* technique where the initial solution $x' \in N_k(x^*)$, and where the neighborhood structure N_k is used. If the local minimum x'' is better than x^* (i.e. $f(x'') < f(x^*)$), then x^* is replaced by x'' and the next iteration is completed with the neighborhood structure N_1 . Otherwise, the neighborhood structure N_{k+1} is used at the next iteration. Note that we return to structure N_1 whenever x'' is better than x^* because, in general, the complexity of the neighborhood structures increases with k . Furthermore, the approach can be implemented using different *NST* at different iterations.

The stopping criteria are usually specified in terms of a maximum number of iterations and in terms of a maximum number of consecutive iterations where the objective function does not improve.

A first order diversification strategy [31] can be used after completion of a *NST* in order to move away from the current local minimum by modifying the values of the variables in such a way as to deteriorate the value of the objective function f as little as possible and even to improve it. In general, during this phase, the value of each variable is modified at most once. The solution generated is used as the initial solution for the *NST*.

A second order diversification strategy [31] relies on some mechanism (long term tabu lists, for instance) to keep track of the values taken by the variables in the different local minima generated so far by applying the *NST*. Then a new initial solution for the *NST* is generated where the value of each variable is selected in order to avoid the values most frequently observed in the different local minima generated so far.

Initialization :

Select an initial solution $x^o \in X$

Select the set of neighborhood structures $N_k, k = 1, 2, \dots, \bar{k}$

Let iter := 0; niter := 0

Let $x := x^o; x^* := x^o; \text{stop} := \text{false}$

While not stop

$k := 1$

While $k \leq \bar{k}$

iter := iter + 1; niter := niter + 1

Generate randomly $x' \in N_k(x^*)$

Apply a *NST* technique with x' as initial solution, using the neighborhood structure N_k . Denote x'' the local minimum obtained.

If $f(x'') < f(x^*)$,

then $x^* := x'', k := 1$, and niter := 0

else $k := k + 1$

If iter = itermax **or** niter = nitermax **then** stop := true

x^* is the best solution generated

Figure 4.1: Variable Neighborhood Search Approach.

5 Population Based Techniques

In these techniques, an evolutionary mechanism is used to transform a population (set) of solutions from one generation to the next in order to lead the search into promising regions of the feasible domain. At each generation (iteration) two different processes are completed : a *collective process* and an *individual process*.

During the collective process, the solutions of the current population are compared and combined to generate new solutions inheriting the prevailing characteristics included in the parent solutions. Then the new solutions evolve individually according to the individual process. Hence a new population of solutions is generated (i.e., a new generation).

Different collective and individual processes give rise to different techniques. Three techniques are summarized in this section : the genetic algorithm, the scatter search, and the ant colonies algorithm.

5.1 Genetic Algorithm

Genetic Algorithms [6, 23, 28] mimic biological mechanisms based on selection and natural evolution principles. The *collective process* includes a *selection operator* and a *crossover* (reproduction) *operator*. A *mutation operator* is used in the *individual process*.

At each iteration (or generation) the three operators are applied to generate a set of new (offspring) solutions. Then a fourth operator (*culling operator*) is applied to determine a new population by selecting solutions in the set of parent-solutions and offspring-solutions. In general, the size of the population is maintained to a fixed value (N), and the new population includes the best fitted solutions (i.e., the solutions x having the smallest values $f(x)$) among the solutions in the current population and the offspring-solutions.

Note that in more classical genetic algorithm variants, the complete population is usually replaced at each generation. In order to do that, N parent-solutions are selected and paired. Then a crossover operator is applied according to some probability to generate two offspring-solutions. Otherwise, the two parent-solutions become their own offspring-solutions. The population of the next iteration includes the offspring-solutions.

In a steady-state population variant, only few individuals are changed in the population at each generation according to the strategy described above. The merits of the steady-state approach are discussed in [6, 38].

An essential feature of this solution approach is the *encoding* of the solutions (or the representation of the solutions). Each solution $x \in X$ is encoded (or represented) as a vector $z \in \mathbb{R}^m$. (Note that the value of a solution x encoded as z is denoted $f(x)$ or $f(z)$.) In general $X \in \mathbb{R}^n$, and $m \neq n$. The encoding is problem dependent. The collective and individual process are applied using the encoded representation z of the solution x .

A feasible solution x of the graph coloring problem of the form

$$\begin{aligned} x_{ij(i)} &= 1 & i \in V \\ x_{ij} &= 0 & i \in V; j = 1, 2, \dots, \xi; j \neq j(i) \end{aligned}$$

is encoded as a vector $z \in \mathbb{R}^{|V|}$ where

$$z_i = j(i) \quad i \in V.$$

Hence the i^{th} component of the vector z is equal to the color number of vertex i .

A general formulation of the Genetic Algorithm is summarized in Figure 5.1. The different operators are specified in the following sections.

5.1.1 Selection operator

This operator is used to select an even number of (parent-) solutions from the current population. Each parent-solution is selected according to one of the strategies described below. Note that the same solution can be selected more than once. The parent-solutions are paired two by two to reproduce themselves according to a crossover operator in the next phase of the algorithm.

The most straightforward operator amounts to selecting randomly each solution from the current population P . This approach has the advantage of promoting diversity in the population generated. But several other selection operators are driven by the basic principle that the best fitted individuals (solutions) of a population should survive and should be used for reproduction. Hence these operators have a degree of elitism proportional to the extent to which this basic principle is followed.

Initialization :

Generate an initial population of N solutions x^i , $i = 1, 2, \dots, N$; i.e., $P_0 := \{z^1, z^2, \dots, z^N\}$ where x^i is encoded as z^i , $i = 1, 2, \dots, N$

Let $\text{iter} := 0$; $\text{niter} := 0$

Let $P := P_0$; $\text{stop} := \text{false}$

Let z^* be the best solution in P_0 ; i.e. $z^* = \arg \min_{z \in P_0} \{f(z)\}$

While not stop

$\text{iter} := \text{iter} + 1$; $\text{niter} := \text{niter} + 1$

Apply the selection operator

Apply the crossover operator

Apply the mutation operator

Apply the culling operator to obtain a new population P' of solutions

$P := P'$

$z' = \arg \min_{z \in P'} \{f(z)\}$

If $f(z') < f(z^*)$ **then** $z^* := z'$, and $\text{niter} := 0$

If $\text{iter} = \text{itermax}$ **or** $\text{niter} = \text{nitermax}$ **then** $\text{stop} := \text{true}$

z^* is the best solution generated

Figure 5.1: Genetic Algorithm.

The *proportional* (or *roulette wheel*) selection operator promotes the selection of the best fitted solutions as follows:

- i) Consider any ordering z^1, z^2, \dots, z^N of the solutions in P .
- ii) Select a random number α in the interval

$$\left[0, \sum_{i=1}^N \frac{1}{f(z^i)}\right].$$

- iii) Let τ be the smallest index such that

$$\sum_{i=1}^{\tau} \frac{1}{f(z^i)} \geq \alpha.$$

- iv) Then solution z^τ is selected as a parent-solution.

Clearly, the chance of selecting a solution z increases with its fitness $\frac{1}{f(z)}$ since the length of the subinterval of $[0, \sum_{i=1}^N \frac{1}{f(z^i)}]$ associated with z is larger if $f(z)$ is smaller.

The *tournament* selection operator consists in choosing the best solution among a subset of solutions of P chosen randomly. More specifically, let \bar{N} be an integer such that $1 \leq \bar{N} \leq N$

- i) Select randomly \bar{N} solutions from P one at a time (i.e., the same solution can be selected more than once) to generate the subset \bar{P} .
- ii) Let \bar{z} be the best solution of this subset:

$$\bar{z} = \arg \min_{z \in \bar{P}} \{f(z)\}.$$

- iii) Then solution \bar{z} is selected as a parent-solution.

Note that if $\bar{N} = N$, then the best solution of the population is selected (pure elitism selection operator), and if $\bar{N} = 1$, then the parent-solution is selected randomly (random selection operator).

The main drawback in using elitist selection is a premature convergence of the algorithm to a population of (almost) identical solutions corresponding to a local minimum that can be far from the global minimum. Hence other selection operators have been proposed where the degree of elitism used is in some sense proportional to the diversity of the population. The following selection operator [15] belongs to this category:

- i) Order the solutions in P in increasing order of their value; i.e., z^1, z^2, \dots, z^N is the order if $f(z^1) \leq f(z^2) \leq \dots \leq f(z^N)$.
- ii) Let $r = 1 + D$, where $D \in [0, 1]$ is a population diversity measure (i.e., $D \in [0, 1]$ increases with the diversity of the population). Initialize $r = 2$.
- iii) Select a random number $\mu \in [0, N^{1/r}]$. Let $\tau = \lceil \mu^r \rceil$.
- iv) Then solution z^τ is selected as a parent-solution.

This parent selection approach dynamically adjusts itself to be more elitist when the population's diversity is high and less aggressive as the population includes solutions that are increasingly similar.

This list of selection operators is not exhaustive but it includes several operators commonly used.

5.1.2 Crossover operator

The crossover operator is used to generate new feasible solutions including interesting components contained in different solutions of the current population. The objective is to guide the search toward promising regions of the feasible domain X while maintaining some level of diversity in the population. In order to do that, pairs of parent-solutions are combined to generate offspring-solutions. Different crossover (or recombination) operators can be used.

The *one point crossover* generates two offspring-solutions from two parent-solutions $[z_1^1, z_2^1, \dots, z_m^1]$ and $[z_1^2, z_2^2, \dots, z_m^2]$ as follows:

- i) Select randomly a position (or index) ρ , $1 \leq \rho < m$.
- ii) Then the offspring-solutions are defined by

$$\begin{aligned} \text{offspring-solution 1} &: [z_1^1, \dots, z_\rho^1, z_{\rho+1}^2, \dots, z_m^2] \\ \text{offspring-solution 2} &: [z_1^2, \dots, z_\rho^2, z_{\rho+1}^1, \dots, z_m^1]. \end{aligned}$$

Hence the first ρ components of the offspring-solution 1 (offspring-solution 2) are the corresponding components of the parent-solution1 (parent-solution 2), and the other components are the corresponding components of the parent-solution 2 (parent-solution 1).

In the *two-point crossover*, two different positions (or indices) are used to generate two offspring-solutions from the two parent solutions $[z_1^1, z_2^1, \dots, z_m^1]$ and $[z_1^2, z_2^2, \dots, z_m^2]$:

- i) Select randomly two positions (indices) μ and ν , $1 \leq \mu < \nu \leq m$.
- ii) The offspring-solutions are defined by:

$$\begin{aligned} \text{offspring-solution 1 : } & [z_1^1, \dots, z_{\mu-1}^1, z_{\mu}^2, \dots, z_{\nu}^2, z_{\nu+1}^1, \dots, z_m^1] \\ \text{offspring-solution 2 : } & [z_1^2, \dots, z_{\mu-1}^2, z_{\mu}^1, \dots, z_{\nu}^1, z_{\nu+1}^2, \dots, z_m^2] \end{aligned}$$

Hence the offspring-solution 1 (offspring-solution 2) has components $\mu, \mu + 1, \dots, \nu$ identical with those of the parent-solution 2 (parent-solution 1), and the other components are identical to the corresponding ones in the parent-solution 1 (parent-solution 2).

The *uniform crossover* requires a vector of bits (0 or 1) of dimension m to generate two offspring-solutions from two parents solutions $[z_1^1, z_2^1, \dots, z_m^1]$ and $[z_1^2, z_2^2, \dots, z_m^2]$:

- i) Generate randomly a vector of bits, for example

$$[0, 1, 1, 0, \dots, 1, 0].$$

- ii) Then the offspring-solutions are defined as follows:

$$\begin{aligned} \text{parent-solution 1 : } & [z_1^1, z_2^1, z_3^1, z_4^1, \dots, z_{m-1}^1, z_m^1] \\ \text{parent-solution 2 : } & [z_1^2, z_2^2, z_3^2, z_4^2, \dots, z_{m-1}^2, z_m^2] \\ \text{vector of bits : } & [0, 1, 1, 0, \dots, 1, 0] \\ \text{offspring solution 1 : } & [z_1^1, z_2^2, z_3^2, z_4^1, \dots, z_{m-1}^2, z_m^1] \\ \text{offspring solution 2 : } & [z_1^2, z_2^1, z_3^1, z_4^2, \dots, z_{m-1}^1, z_m^2]. \end{aligned}$$

Hence, for all $i = 1, 2, \dots, m$, the i^{th} component of the offspring-solution 1 (offspring-solution 2) is identical to the corresponding component of the parent-solution 1 (parent-solution 2) if the i^{th} component of the vector of bits is equal to 0, otherwise, it is identical to the corresponding component of the parent-solution 2 (parent-solution 1). Syswerda [37] introduces theoretical and numerical results showing that the uniform crossover generates better results than the one-point and two-points crossover in general.

These crossover operators are sometimes too general to be efficient. Hence, whenever it is possible, we should rely on the structure of the problem to specify an ad hoc problem dependent crossover operator in order to improve the efficiency of the algorithm. Such operators are presented in the following section for the graph coloring problem.

Furthermore, whenever the structure of the problem is such that the offspring-solutions generated are not necessarily feasible, then an auxiliary procedure is required to recover feasibility. Such a procedure is used to transform the offspring-solution into a feasible solution in its neighborhood.

5.1.3 Advanced adapted crossover

First we summarize two advanced crossover operators, and we illustrate their implementation for the graph coloring problem. Then we conclude this section with an operator specifically adapted to the graph coloring problem.

5.1.3.1 Path relinking crossover operator

This operator [17] generates an offspring-solution os from two parent-solutions $z^1 = [z_1^1, z_2^1, \dots, z_m^1]$ and $z^2 = [z_1^2, z_2^2, \dots, z_m^2]$. The offspring-solution is on a “path” linking z^1 and z^2 .

The operator is summarized in Figure 5.2. Given any pair of solutions ps^1 and ps^2 , $d(ps^1, ps^2)$ is equal to number of components that are different in the vectors ps^1 and ps^2 . (Note that $d(ps^1, ps^2)$ is the Hamming Distance between ps^1 and ps^2). Hence $d(ps^1, ps^2) = 0$ if $ps^1 = ps^2$.

The subset $N(ps^1, ps^2) \subset N(ps^1)$ are the elements of $N(ps^1)$ that are closer to ps^2 than ps^1 ; i.e.,

$$N(ps^1, ps^2) = \{z \in N(ps^1) : d(z, ps^2) < d(ps^1, ps^2)\}.$$

Hence a solution $z \in N(ps^1, ps^2)$ has at least one more component in common with ps^2 .

Referring to Figure 5.2, at each iteration we select a solution $ps \in N(ps^1, ps^2)$ according to the proportional selection operator. If this solution is better than the initial parent-solutions z^1 and z^2 , then ps becomes the offspring solution os . Otherwise, ps^1 is replaced by ps , and then ps^2 plays the role of ps^1 . The procedure is repeated until $ps^1 = ps^2$. Then $os = ps^1 = ps^2$.

It is easy to see that for the graph coloring problem,

$$N(ps^1, ps^2) = \{z : z = ps^1 \oplus m \text{ for some } m \in M(ps^1, ps^2)\}$$

where

$$M(ps^1, ps^2) = \{[i, j] : i \in V \text{ such that } ps_i^1 \neq ps_i^2; j = ps_i^2\}.$$

5.1.3.2 Adaptive structured combination operator

This operator [17] also generates only one offspring solution os from two parent-solutions z^1 and z^2 . The principle of this operator relies on three properties as introduced by Glover :

- i) Representation property : A set of votes (Votes z^i) is associated with each parent-solution z^i , $i = 1, 2$, and decisions in constructing an offspring-solution os are based on these votes.

- ii) Trial solution property : Votes translate into a sequence of partial solutions, eventually generating an offspring-solution os .
- iii) Update property: The sets of votes associated with the parent-solutions are updated according to the partial solution generated so far.

Initialization :

Let z^1 and z^2 be two parent-solutions

Let $ps^1 := z^1$; $ps^2 := z^2$

Let stop := false

While not stop

Determine the subset $N(ps^1, ps^2) \subset N(ps^1)$

Use the proportional selection operator to choose $ps \in N(ps^1, ps^2)$

If $f(ps) < f(z^1)$ **and** $f(ps) < f(z^2)$ **then** stop := true

$ps^1 := ps^2$

$ps^2 := ps$

If $d(ps^1, ps^2) = 0$ **then** stop := true

$os := ps$

Figure 5.2: Path relinking crossover operator.

The operator is summarized in Figure 5.3. The procedure is initialized with a set of votes $V^1 \cup V^2$ where $V^1 = \text{Votes}(z^1)$ and $V^2 = \text{Votes}(z^2)$. At each iteration, given the current partial solution pos , the function $eval(v)$ evaluates the cost induced if the next decision is based on vote v . Then, decision $\bar{v} \in V^1 \cup V^2$ is selected according to the proportional selection operator relatively to the function $eval(v)$. The current partial solution pos is modified according to vote \bar{v} . (This operation is denoted $mod(pos, \bar{v})$ in Figure 5.3). The iteration is completed by updating

Initialization :

Let z^1 and z^2 be two parent-solutions

Let $V^1 = \text{Votes}(z^1)$; $V^2 = \text{Votes}(z^2)$

Let $\text{stop} := \text{false}$

While not stop

For $v \in V^1 \cup V^2$

Determine $\text{eval}(v)$

Use the proportional selection operator relatively to $\text{eval}(v)$ to choose $\bar{v} \in V^1 \cup V^2$

$\text{mod}(\text{pos}, \bar{v})$

Update V^1 according to \bar{v}

Update V^2 according to \bar{v}

If $V_1 \cup V_2 = \phi$ **then** $\text{stop} := \text{true}$

$os := \text{pos}$

Figure 5.3: Adaptive structured combination operator.

the sets of votes V^1 and V^2 according to the decision based on vote \bar{v} used to modify the current partial solution. When $V_1 \cup V_2 = \phi$, then the offspring-solution os is pos .

For the graph coloring problem, the set of votes $\text{Votes}(z)$ associated with a solution z is specified as follows:

$$\text{Votes}(z) = \{(1, j(1)), (2, j(2)), \dots, (i, j(i)), \dots, (|V|, j(|V|))\};$$

i.e., $\text{Votes}(z)$ is the set of pairs of vertices and their colors. The function $\text{eval}(i, j)$ is evaluated as follows:

$eval(i, j)$ = The number of additional pairs of adjacent vertices having the same color in the partially colored graph if vertex i has color j .

If $\bar{v} = (i, j)$, then the operation $mod(pos, \bar{v})$ colors vertex i with color j in the partial solution pos ; i.e., $pos(i) = j$. Finally, the sets of votes V^1 and V^2 are updated as follows : (assuming that $\bar{v} = (i, j)$)

$$\begin{aligned} V^1 &:= V^1 - \{(i, z_i^1)\} \\ V^2 &:= V^2 - \{(i, z_i^2)\}. \end{aligned}$$

5.1.3.3 Knowledge-augmented crossover for graph coloring

The following crossover operator takes advantage of the specific structure of the graph coloring problem [16]. The procedure is summarized in Figure 5.4. It generates an offspring-solution os from two parent-solutions z^1 and z^2 .

For any solution z , let $CN(z)$ denotes the set of color conflicting vertices in z (i.e., the set of vertices having an adjacent vertex with the same color):

$$CN(z) = \{i \in V : \exists(i, \ell) \in E \text{ such that } j(i) = j(\ell)\}.$$

An offspring-solution os is generated by coloring a conflicting vertex in one parent-solution using the color supplied by the other parent-solution, if it is conflict-free in the latter. When a vertex is color conflicting in both parent-solutions, we select the color that is used the least often for adjacent vertices in either parent. Each of the remaining vertices is colored with the color in one of the parent-solution, selected randomly, as in the uniform crossover.

Note that in Figure 5.4, $use(i, j)$ is equal to the number of times that color j is used to color an adjacent vertex of i in z^1 or z^2 . Hence, if $N(i)$ denotes the set of adjacent vertices to i (i.e., $N(i) = \{\ell \in V : (i, \ell) \in E\}$), then

$$use(i, j) = \sum_{\ell \in N(i)} col(j, i, \ell)$$

where

$$col(j, i, \ell) = \begin{cases} 1 & \text{if } z_\ell^1 = j \text{ or } z_\ell^2 = j \\ 0 & \text{otherwise.} \end{cases}$$

Initialization :

Let z^1 and z^2 be two parent-solutions

For $i \in CN(z^1)$

If $i \notin CN(z^2)$ **then** $os_i := z_i^2$

For $i \in CN(z^2)$

If $i \notin CN(z^1)$ **then** $os_i := z_i^1$

For $i \in CN(z^1) \cap CN(z^2)$

$os_i := \arg \min_{i \leq j \leq \xi} \{use(i, j)\}$

For $i \in (V - (CN(z^1) \cup CN(z^2)))$

Select randomly $\alpha \in \{0, 1\}$

If $\alpha = 0$ **then** $os_i := z_i^1$

else $os_i := z_i^2$

os is the offspring-solution generated

Figure 5.4: Knowledge-augmented crossover

5.1.4 Mutation operator

Once the offspring-solutions are generated, an *individual process* can be used to modify each of these by applying a mutation operator. In more traditional variants of the genetic algorithm, the mutation operator is used to modify arbitrarily each component z_i of the solution z with a small probability:

For $i = 1$ **to** N

 Generate a random number $\alpha \in [0, 1]$

If $\alpha < \bar{\alpha}$, **then** select randomly a new value for z_i .

Here, the value of $\bar{\alpha}$ is small enough in order to modify z_i with a small probability. Hence, this operator simulates random events perturbing the natural evolution process.

This mutation operator is not as essential as the preceding ones since a genetic algorithm can be specified using only selection and crossover operators without any mutation operator. But an advantage of using a mutation operator of the type described above is to introduce some randomness in the approach in order to promote diversity in the current population. This may prevent from premature convergence to a bad local minimum.

5.2 Scatter Search

The Scatter Search method was introduced by F. Glover in [21] for integer programming problems. *The collective process* consists in combining (for instance, making a linear combination) more than two solutions of the population in order to generate a new solution. The role of the *individual process* is to recover feasibility for the solution generated.

This method has not been widely used until now. Furthermore this method is often seen as a generalization of a genetic algorithm where more than two parent-solutions are used to generate offspring-solutions.

5.3 Ant Algorithm

This type of method mimics the behavior of insect colonies completing their activities. It looks like the collective behavior is the result of an invisible agent that coordinates all individual activities. Hence the behavior of each individual member is guided by this invisible agent, but each action has an impact on the behavior of the individual member executing it and also on the behavior of all other members of the colony.

To illustrate the principle, consider an ant colony including na individual members looking for food locations. Initially, all ants leave the nest, and they move randomly on the ground. Whenever an individual finds a food location, it informs the other colony members by leaving a trace of pheromone (a volatile substance) on its way back to the nest in order to guide them to the location. When the other members go to the location, they also leave traces of pheromone on their way back to the nest. Hence the food collection is optimized through the reinforcement of the

traces of pheromone on the most frequently used routes. On the long run, the colony will exploit only the closest location because the traces of pheromone leading to farther locations evaporate.

In [7, 8, 18] Dorigo et al. propose an evolutionary method that mimics the collective behavior of an ant colony. During the *collective process*, the solutions of the current population are used to update a global memory. Then, in the *individual process*, a new feasible solution is generated by means of a constructive method that uses the information in the global memory. Hence, each application of the constructive method corresponds to a trip of an individual ant, and the global memory corresponds to the traces of pheromone.

This type of method is well suited for assignment-type problems. Consider the construction of a new solution. At each iteration of the constructive procedure, we first have to select an item, and then, a resource to which it is assigned. In a traditional constructive method these decisions are made to optimize the objective function given the decisions taken in earlier iterations. In a sense, we select the currently best desirability pair of item and resource. In the ant algorithm the decisions are made according to a probability depending not only on the current desirability of the items and of the resources, but also on past history included in the global memory. More specifically, at iteration k we select an item i (not yet selected) with probability $p_{it}(k, i)$ and an admissible resource j for item i with probability $p_{re}(k, i, j)$:

$$p_{it}(k, i) = \frac{\tau_{it}(x[k-1], i) \cdot \eta_{it}(x[k-1], i)}{\sum_{\nu \in I - \{i_1, i_2, \dots, i_{k-1}\}} \tau_{it}(x[k-1], \nu) \cdot \eta_{re}(x[k-1], \nu)}$$

$$p_{re}(k, i, j) = \frac{\tau_{re}(x[k-1], i, j) \cdot \eta_{re}(x[k-1], i, j)}{\sum_{\ell \in J_i} \tau_{re}(x[k-1], i, \ell) \cdot \eta_{re}(x[k-1], i, \ell)}$$

where I denotes the set of items, J_i the set of admissible resources for item i , and $x[k-1]$ the partial solution where items i_1, i_2, \dots, i_{k-1} are assigned to j_1, j_2, \dots, j_{k-1} respectively. Given $x[k-1]$, $\eta_{it}(x[k-1], i)$ and $\eta_{re}(x[k-1], i, j)$ denote the current desirability of item i and of admissible resource j for i , respectively. Also, $\tau_{it}(x[k-1], i)$ and $\tau_{re}(x[k-1], i, j)$ denote the traces (of global memory) for item i and admissible resource j for i , respectively. The ant algorithm is summarized in Figure 5.5.

Initialization :

Let x^o be any feasible solution

Let iter := 0; niter := 0

Let $x^* := x^o$

Let $\tau_{it}(\dots, i) := 1, i \in I; \tau_{re}(\dots, i, j) := 1, i \in I, j \in J_i$

Let stop := false

While not stop

iter := iter + 1; niter := niter + 1

For $a := 1$ **to** na

For $k := 0$ **to** $|I|$

 Select an item $i \in I - \{i_1, i_2, \dots, i_{k-1}\}$ with probability $p_{it}(k, i)$

 Select a resource $j \in J_i$ with probability $p_{re}(k, i, j)$

 Assign item i to resource j

 Evaluate $f(x^a)$ (value of solution x^a)

$P := \{x^1, x^2, \dots, x^{na}\}$

$\bar{x} := \arg \min_{x \in P} \{f(x)\}$

If $f(\bar{x}) < f(x^*)$ **then** $x^* := \bar{x}$ and niter := 0

If iter = itermax **or** niter := nitermax **then** stop := true

Update traces τ_{it} and τ_{re}

x^* is the best solution generated

Figure 5.5: Ant Algorithm

To illustrate how to specify the desirability and the trace in evaluating the probabilities, consider the graph coloring problem. In this case, $f(x[k-1])$ denotes the number of pairs of adjacent vertices having the same color when vertices i_1, i_2, \dots, i_{k-1} have colors j_1, j_2, \dots, j_{k-1} , respectively.

Past history included in the global memory does not really influence the selection of the item (vertex) at any given iteration. Hence

$$\tau_{it}(x[k-1], i) \equiv 1.$$

The current desirability of a vertex can be specified as its degree; i.e.,

$$\eta_{it}(x[k-1], i) = \text{degree of } i.$$

Indeed, it becomes more difficult to select a color for a vertex having a larger number of adjacent vertices when a larger number of vertices are already colored.

The current desirability of a color j (resource) for a vertex (item) i depends on the colors of the adjacent vertices of i . Hence we may define

$$\eta_{re}(x[k-1], i, j) = \begin{cases} 1 & \text{if an adjacent vertex of } i \text{ has already color } j \\ 10 & \text{otherwise} \end{cases}$$

The importance of the trace (global memory) of using color j for vertex i is directly proportional to the number of solutions where vertex i has color j , and to their quality. But this trace is independent of the current partial solution $x[k-1]$ (i.e., $\tau_{re}(x[k-1], i, j) = \tau_{re}(i, j)$). At each iteration of the ant algorithm, the trace is updated as follows:

$$\tau_{re}(i, j) := \rho\tau_{re}(i, j) + \Delta\tau_{re}(i, j)$$

where

$$\Delta\tau_{re}(i, j) = \sum_{a=1}^{na} \Delta\tau_{re}^a(i, j)$$

and

$$\Delta\tau_{re}^a(i, j) = \begin{cases} \frac{Q}{f(x^a)}, & \text{if vertex } i \text{ has color } j \text{ in solution } x^a \\ 0, & \text{otherwise.} \end{cases}$$

The value $(1 - \rho)$ corresponds to an evaporating coefficient, and Q is a constant.

Note that other ways to specify the current desirability and the trace can be used to generate other variants of the algorithm [5]. This approach has also been used to solve quadratic assignment problems [18] and job shop problems [3].

6 Hybrid Methods

Hybrid methods can be specified by combining two or more heuristic methods in order to increase their efficiency. Different methods are obtained by combining a population based algorithm with a neighborhood search technique in the *individual process* to improve the offspring-solutions.

It is well-known that in general, population based algorithms are very time consuming. Furthermore, the solutions generated are generally worse than those generated with a neighborhood search technique like Tabu search.

The strength of hybrid method comes from combining two complementary search strategy. Indeed, the purpose of using a neighborhood search technique is to intensify the search in a given region of the feasible domain while a population based algorithm allows to search more extensively the whole feasible domain. Hence the *collective process* induces some long term diversification.

In [14, 15] we use hybrid methods combining a genetic algorithm with the descent method or the tabu search as a mutation operator to solve several combinatorial problems (quadratic assignment, graph coloring, maximum clique, and satisfiability). The numerical results indicate that these methods are very time consuming, but that they allow to generate very good solutions for instances of these problems which are difficult to solve.

Recently we have been using a new crossover operator for the genetic algorithm that may be seen as an hybrid of any other crossover operator and a constructive method. The underlying principle is an adaptation of the so called “mirror method” [39] to solve the following problem:

$$\text{Min}_{x \in \mathbb{R}^n} g(x).$$

This iterative procedure can be summarized as follows: At each iteration,

- i) Let $x^1, x^2, \dots, x^n, x^{n+1} \in \mathbb{R}^n$ be $(n + 1)$ solutions ordered such that

$$f(x^1) \leq f(x^2) \leq \dots \leq f(x^n).$$

- ii) Determine the centroid solution c of the n better solutions x^1, x^2, \dots, x^n ; i.e.

$$c_j = \frac{1}{n} \sum_{i=1}^n x_j^i, \quad j = 1, 2, \dots, n.$$

- iii) Replace the worst solution x^{n+1} by its mirror image m with respect to the centroid c :

$$m_j = c_j + \alpha(c_j - x_j^{n+1}).$$

In Figure 6.1 we illustrate the procedure in \mathbb{R}^2 with $\alpha = 1$.

Figure 6.1 Mirror image.

This procedure can be adapted to obtain the mirror crossover operator. Consider any offspring-solution os generated by applying any other crossover operator with parent-solutions z^1 and z^2 . Hence os can be seen as “some kind of centroid” of z^1 and z^2 . Furthermore, let z^w denotes the worst solution in the current population. Then we can characterize “some kind of mirror image” osm of the worst solution z^w with respect to os using the following principle:

$$os_i = z_i^w \quad \text{or} \quad os_i = osm_i, \quad i = 1, 2, \dots, m.$$

Hence, for the indices i such that $os_i \neq z_i^w$, then $osm_i = os_i$. But for the other indices i (i.e., for i such that $os(i) = z_i^w$) osm_i can be selected using a constructive method to complete the partial solution.

In some way, the constructive method leads away from the worst solution z^w through the centroid os to improve the objective function. This operator can be seen to serve a complementary purpose to the usual crossover operators. Indeed, these operators combine parent-solutions to generate new solutions inheriting their prevailing characteristics. This new operator induces to move away from the prevailing characteristics of the worst solution in the population.

7 Conclusion

In recent years the development of heuristic search methods has attracted the attention of a growing number of researchers. This ever-increasing interest is due to the complexity of a wide range of real-world problems that cannot be tackled by means of exact methods. Significant advances were achieved and good solutions can now be found to problems that were not tractable only a few years ago.

This paper provides an overview of the most popular heuristic search methods. Most of them have proved effective in solving hard optimization problems. The underlying search principles were

illustrated with the well-known graph coloring problem. However, the extent of these mechanisms goes much beyond this specific frame. Examples of applications are in the fields of finance, chemistry, supply chain, scheduling, engineering and telecommunications. For applications of search methods to specific areas, the interested reader is referred to the bibliography of Osman and Laporte [36] and to the book of Glover and Laguna [22].

Choosing a heuristic search method adapted to a given optimization problem is not an easy task. Depending on the problem's specificity, more than one search mechanism is applicable. Before implementing a search mechanism, decision-makers must have a good knowledge of the problem under study and must make an effort to build a proper model. The selection of the most appropriate heuristic search method must then be done with great care while taking into account the following criteria [26] :

simplicity : simple and clear principles should govern the method;

coherence : the various step of the method should follow the general principle of the method;

efficiency : the method should provide optimal or near-optimal solutions to realistic problem instances;

effectiveness : good solutions should be achieved within a reasonable amount of time;

robustness : efficiency and robustness should prevail for a variety of problem instances;

user-friendliness : the method should be easy to understand and to use;

flexibility : the method could be easily adapted to deal with new problem variants.

References

- [1] Aarts E.H.L. Aarts, Korst J., “*Simulated Annealing and Boltzmann Machines*”, Wiley, New York, 1989.
- [2] Cerny V., “Thermodynamical Approach to the Traveling Salesman Problem : an Efficient Simulation Algorithm”, *Journal of Optimization Theory and Applications* 45(1985), 41–51.
- [3] Colorni A., Dorigo M., Maniezzo V., Trubian M., “Ant System for Job Shop Scheduling”, *Belgian Journal of Operations Research, Statistics and Computer Science* 34 (1994), 39–53.
- [4] Costa D., “Méthodes de résolution constructives, séquentielles et évolutives pour des problèmes d’affectation sous contraintes”, Doctoral dissertation, Department of Mathematics, École Polytechnique Fédérale de Lausanne, Switzerland (1995).
- [5] Costa D., Hertz A., “Ants Can Colour Graphs”, *Journal of Operational Research Society* 48(1997),295–305.
- [6] Davis L. (Ed.), “*Handbook of Genetic Algorithms*”, Van Nostrand Reinhold, New York, 1991.
- [7] Dorigo M., Maniezzo V., Colorni A., “The Ant System: Optimization by a Colony of Cooperating Agents”, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: cybernetics* 26(1996), 29–41.
- [8] Dorigo M., Di Caro G., Gambardella L.M., “Ant Algorithms for Discrete Optimization”, *Artificial life* 5(2000), 851–871.
- [9] Dueck G., Scheuer T., “Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing”, *Journal of Computational Physics* 90(1990), 161–175.
- [10] Dueck G., “New Optimization Heuristics : The Great Deluge Algorithm and the Record-to-Record Travel”, *Journal of Computational Physics* 104(1993), 86–92.
- [11] Ferland J.A., Hertz A., Lavoie A., “An Object Oriented Methodology for Solving Assignment Type Problems with Neighborhood Search Techniques”, *Operations Research* 44(1996), 347–359.
- [12] Ferland J.A., Lavoie A., “Exchange Procedures for Timetabling Problems”, *Disc. Appl. Math.* 35 (1992), 237–253.
- [13] Feo T., Resende M.G.C., “Greedy Randomized Adaptive Search Procedures”, *Journal of Global Optimization* 2(1995), 1–27.

- [14] Fleurent C., Ferland J.A. “Genetic Hybrids for the Quadratic Assignment Problem”, *Dimacs Series in Discrete Mathematics and Theoretical Computer Science* 16(1994), 173–187.
- [15] Fleurent C., Ferland J.A., “Object-Oriented Implementation of Heuristic Search Methods for Graph Coloring, Maximum Clique, and Satisfiability”, *Dimacs Series in Discrete Mathematics and Theoretical Computer Science* 26(1996), 619–652.
- [16] Fleurent C., Ferland J.A., “Genetic and Hybrid Algorithms for Graph Coloring”, *Annals of Operations Research* 63(1996), 437–461.
- [17] Fleurent C., Glover F., “Advanced Recombination Operators for Heuristic Search Methods”, Report from Graduate School of Business, University of Colorado, Boulder (1995).
- [18] Gambardella L.M. Taillard E., Dorigo M., “Ant Colonies for the Quadratic Assignment Problem”, *Journal of the Operational Research Society* 50(1999), 167–176.
- [19] Geman S., Geman D., “Stochastic Relaxation, Gibbs Distributions, and Bayesian Restoration of Image”, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6(1984), 721–741.
- [20] Glover F., “Future Paths for Integer Programming and Links to Artificial Intelligence”, *Computers and Operations Research* 13(1986), 533–549.
- [21] Glover F., “Genetic Algorithms and Scatter Search: Unsuspected Potentials”, *Statistics and Computing* 4(1994), 131–140.
- [22] Glover F., Laguna M., “*Tabu Search*”, Kluwer Academic Publishers, Boston (1997).
- [23] Goldberg D.E., “*Genetic Algorithms in Search, Optimization, and Machine Learning*”, Addison Wesley, 1989.
- [24] Hajek B., “Cooling Schedules for Optimal Annealing” *Mathematics of Operations Research* 13(1988), 311–329.
- [25] Hansen P., “The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming”, presented at *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy (1986).
- [26] Hansen P., Mladenovic N., “ Variable Neighborhood Search : Principles and Applications”, *European Journal of Operational Reserach* 130(2001), 449–467.
- [27] Hertz A., De Werra D., “Using Tabu Search Techniques for graph Coloring”, *Computing* 39(1987), 345–351.

- [28] Holland J.H., “*Adaptation in Natural and Artificial Systems*”, Ann Arbor, University of Michigan Press, 1975.
- [29] Johnson D.S., Aragon A., McGeoch L.A., Shevon C., “Optimization by Simulated Annealing : An Experimental Evaluation; Part 1, Graph Partitioning”, *Operations Research* 37(1989), 865–892.
- [30] Johnson D.S., Aragon A., McGeoch L.A., Shevon C., “Optimization by Simulated Annealing : An experimental Evaluation; Part II, Graph Coloring and Number Partitioning”, *Operations Research* 39(1991), 378–406.
- [31] Kelly J.P., Laguna M., Glover F., “A Study of Diversification Strategies for the Quadratic Assignment problem”, *Computers and Operations Research* 21(1994), 885–893.
- [32] Kirkpatrick S., Gelatt C.D. Jr., Vecchi M.P., “Optimization by Simulated Annealing”, *Science* 220(1983), 671–680.
- [33] van Laarhoven P.J.M., Aarts E.H.L., “*Simulated Annealing : Theory and Applications*”, D. Reidel Publishing Company, Dordrecht, 1987.
- [34] Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., Teller E., “Equation of State Calculations by Fast Computing Machine”, *Journal of Chemical Physics* 21(1953), 1087–1091.
- [35] Nicholson T., “*Optimization in Industry*”, Vol. 1, Optimization Techniques, Chapter 10, Longmann Press, London, 1971.
- [36] Osman J.H., Laporte G., “Metaheuristics : A Bibliography”, *Annals of Operations Research* 63(1996), 513–628.
- [37] Syswerda G., “Uniform Crossover in Genetic Algorithms”, *Proceedings of the Third International Conference on Genetic Algorithms*, Schaffer J.D. (Ed.), Morgan Kaufmann Publishers, San Mateo, California, 1989, 2–9.
- [38] Syswerda G., “A Study of Reproduction in Generational and Steady-State Algorithm”, in “*Foundations of Genetic Algorithms*”, Rawlings G.J.E. (Ed.), Morgan Kaufmann, San Mateo (1992), 94–101.
- [39] Swann W.H., “Direct Search Methods”, in Murray W. (ed), “*Numerical Methods for Unconstrained Optimization*, Academic Press, London and New York (1972), 13–28.
- [40] Taillard E., “Robust Tabu Search for the Quadratic Assignment Problem”, *Parallel Computing* 17(1991), 443–455.

- [41] de Werra D., Hertz A., “Tabu Search Techniques : A Tutorial and an Application to Neural Networks”, *OR Spektrum* 11(1989), 131–141.