Local Search Techniques for Scheduling Problems — A Tutorial —

Pablo Moscato
Departamento de Engenharia de Sistemas,
Faculdade de Engenharia Elétrica,
Universidade Estadual de Campinas,
C.P. 6101, Campinas, SP, CEP 13081-970, BRAZIL
e-mail: moscato@densis.fee.unicamp.br
http://www.densis.fee.unicamp.br/~moscato

and

Andrea Schaerf
Dipartimento di Informatica e Sistemistica,
Università di Roma "La Sapienza",
Via Salaria 113, 00198 Roma, ITALY,
e-mail: aschaerf@dis.uniroma1.it
http://www.dis.uniroma1.it/~aschaerf

Abstract

Although it is not a newcomer in the combinatorial optimization literature, *Local Search* is an emerging paradigm for combinatorial search, which has been recently shown to be very effective for a large number of scheduling problems. A number of metaheuristics based on local search have been also proposed to address with success a variety of scheduling problems.

In this tutorial, we survey the basic local search techniques proposed in the literature and implemented in many industrial scheduling systems: Simulated Annealing, Tabu Search, and various forms of Hill Climbing. We also illustrate some of the most promising improvements and variations of such basic techniques which are currently investigated. Finally, we propose the combination of local search with other solution paradigms, such as genetic algorithms and constructive heuristics.

Throughout the tutorial we illustrate three case studies of successful applications of these techniques to real life problems: school timetabling, frequency assignment in mobile radio systems, and sport scheduling.

Contents

1 Introduction

Local search is an emerging paradigm for combinatorial search, which has been recently shown to be very effective for a large number of combinatorial problems. In particular, many industrial systems based on local search have been design for a variety of real-world instances of scheduling problems.

1.1 Local Search Techniques

Local search is based on the simple idea of *navigating* the search space by iteratively stepping from one solution to one of its *neighbours*. The neighbourhood of a solution is the set of solutions that can be obtained by applying an "atomic local change" to it. The definition of the neighbourhood relation is dependent on the specific problem under consideration. The mechanism upon which the neighbour is selected at each step is the main issue that differenciates among different local search techniques. In any cases, however, it is related to the definition of the *objective function*, which assesses the quality of each neighbour, and is also problem dependent.

Local search techniques belong to the larger class of the so-called *selective* methods which are based on the exploration of the search space composed by the instance solutions. For example, another type of selective methods are the *genetic algorithms*, which are based on the evolution of a population of solutions.

On the contrary, *constructive* methods build the solution in a step-by-step manner by iteratively adding a new piece to the partial (up to now infeasible) solution constructed so far. Each step generally consists of a careful assignment decision such that at each step we get closer to feasibility while taking in consideration small changes in the objective function. Constructive techniques can be either exhaustive, leading to some branch-and-bound or backtracking search implementations, or non-exhaustive ones like heuristic greedy methods.

1.2 Scheduling Problems

The term *scheduling* refers to the task of associating one or several resources to activities over a certain time period. This is a very general definition of scheduling, which has been refined and specialized by various authors. In this tutorial, however, we stick to the general definition, and we discuss the application of local search techniques to problems that fall under the term 'scheduling' in this wide sense. In particular, our case study are problems on subjects such as *timetabling*, *frequency assignment*, and *sport tournament scheduling*.

The common groud of these diverse problems is to be all combinatorial problems (search or optimization ones), a large number of them being NP-hard, many decision problems are NP-Complete, and to show similar types of constraints (precedence, overlap, availability, capacity, ...). But most of all, they share the fact of having been recognized to be genuinely "difficult on the average" in practical real-life cases. For this reason, such problems have not been addressed in a complete and

satisfactory way yet, and are still matter of theoretical research and experimental investigation.

1.3 Objective and Scope of the Paper

In this paper we aim at reconstructing a unified view of the countless different techniques and variations of local search proposed in the literature, specifically in the field of scheduling.

To this aim, we first provide a precise definition of local search. This corresponds to the most common way of seeing local search, although different definitions have been provided in the past. For example, *genetic algorithms* do not fall under local search in our view, whereas the are included in local search by other authors (see e.g., [113]). However, we do discuss memetic algorithms, which is another population approach which in most implementations use local search techniques and have been shown to be more successful than GAs in practice. Our definition also serves to delimit clearly the scope of the paper.

Second, we identify three techniques which have been investigated in the academic framework and successfully employed in many industrial applications. These techniques we illustrate are *Hill Climbing*, *Simulated Annealing*, and *Tabu Search*.

Such techniques are not only widely accepted as the basic ones, but they also correspond to three different philosophies inside the local search framework (see Section 3). Therefore they represent an ideal starting point for a comprehensive introduction to the local search paradigm.

We then try to classify all various other approaches, on the basis on the above-mentioned three. Therefore, in the subsequent discussion we focus on the difference of the proposals with the basic techniques. More specifically, we identify the part of the basic technique that is improved or replaced, this can be for example the neighborhood relation, or the cost function. For the sakeness of clarity, the classification is organized around of the part of the given model that is improved, rather than on the techniques themselves or on the applications.

We also address the issue of combining local search with different solution technique, either selective or constructive one. In particular, we illustrate a number of ways to combine local search with other search paradigms so as to obtain hybrid techniques that merge the advantages of different approaches. As we said before, we will discuss *Memetic Algorithms* which rely on hybridization of a population-based approach (like a GA for instance) with local search techniques or other mechanisms that help to evolve the solutions (like a truncated branch-and-bound method, for instance).

The paper is organized as follows. In Section 2 we provide some general definitions related to the local search paradigm. In Section 3 we introduce the basic techniques proposed in the literature. In Section 4 we discuss a number of variations of the basic techniques, and possible combinations of them. In Section 5 we present hybrid techniques. In Section 6 we group a set of side issues related to the solution of scheduling problems by local search methods. Finally, conclusions are drawn in Section 7.

2 Local Search

In this section we provide some introductory notions on local search. In particular, we first briefly define in Section 2.1 the notions of *search* and *optimization problems*, to which local search techniques can be applied. We then discuss in Section 2.2 the three notions of *search space*, *neighbourhood relation*, and *cost function* that are the key elements of the local search paradigm, and are independent of the specific local search technique choosen to solve the problem.

2.1 Computational Problems

A *search problem* P has a set I_P of instances and for each instance $x \in I_P$ there exists a set $ans_P(x)$ of corresponding answers. A subset of $ans_P(x)$, called $sol_P(x)$, identifies the feasible (acceptable) solutions to problem P. We will say that an algorithm *solves* problem P if on input $x \in I_P$ it outputs any member $y \in sol_P(x)$, or in case $sol_P(x)$ is empty it reports that no such y exists.

An *optimization problem* is a special kind of search problem where every instance $x \in I_P$ has a set $sol_P(x)$ of *finite* cardinality and every solution $y \in sol_P(x)$ has associated an *objective function* $m_P(y, x)$. The problem is to find which feasible solution $y^* \in sol_P(x)$ minimizes (or maximizes) the value of the objective function.

In this paper we focus (without loss of generality) on problems in which we seek the y^* that *minimizes* the value of the objective function over all $y \in sol_P(x)$. Therefore, we view the objective function $m_P(y^*, x)$ as the *penalty* (or *cost*) of solution y^* given the instance x for problem P.

As an example, we will consider the MIN GRAPH COLOURING problem. Given a graph $G(V, E)$, the problem P is to find the minimum number of colours needed to colour each node such that there is no edge $e \in E$ connecting two nodes which have been assigned the same colour. In this case $x = G(V, E)$, $sol_P(x)$ represents *valid* colourings of G, $m_P(y, x)$ is the number of colours used by colouring y.

2.2 Local Search Problems

In order to apply a local search algorithm to a computational problem, either a search problem or an optimization one, we need to define three entities, namely, the *search space*, the *neighborhood relation*, and the *cost function*.

A computational problem upon which these three entities are defined is called a *local search problem*. A given computational problem P can give rise to different local search problems for different definitions of these entities.

A slightly different and tighter definition of local search problems and the definition of complexity classes related to such notion is given in [64]. Here we are trying to follow the same guidelines but having in mind what is generally know as local search in practice.

2.2.1 Search Space

Given a computational problem P, we associate to each instance of $x \in I_P$ a search space $S_P(x)$, with the following properties.

- Each element $s \in S_P(x)$ represents an element $y \in ans_P(x)$.
- For search problems: at least one element of $sol_P(x)$ is represented in $S_P(x)$.
- For optimization problems: at least one optimal element of $sol_P(x)$ is represented in $S_P(x)$.

We may say that if the previous requirements have been achieved, we have a *valid representation* or *valid formulation* of the problem. For simplicity, we will write just S for $S(x)_P$ when x and P are clear from the content. Furthermore, we will refer to elements of S as *solutions*.

2.2.2 Neighborhood Relation

Given a problem P, an instance $x \in I_P$ and a search space S for it, we assign to each element $s \in S$ a set $\mathcal{N}(s) \subseteq S$ of neighboring solutions of s. The set $\mathcal{N}(s)$ is called the *neighborhood* of s and each member $s' \in \mathcal{N}(s)$ is called a *neighbor* of s.

For each s the set $\mathcal{N}(s)$ needs not be listed explicitly, in general it is implicitly defined by referring to a set of possible *moves*, which define transitions between solutions. Moves are usually defined as local modifications of some part of s. The "locality" of moves (under a correspondingly appropriate definition of distance between solutions) is one of the key ingredients of local search, and actually it has also given the name of the whole search paradigm. Nevertheless, from the definition above there is no implication that there exist "closeness" in some sense among neighbors, and actually complex neighborhood definitions can be used as well (a reader familiar with local search techniques for the Traveling Salesman Problem might recall the λ-change upon which the Lin-Kernighan algorithm relies).

2.2.3 Cost Function

The selection of the move to perform at each step of the search is based on the *cost function*. The cost function F associates to each element $s \in S$ a value $F(s)$ that assesses the quality of the solution. For the sake of simplicity, we assume that the value of F is always a positive and integer, or in other words, that the codomain of F is the set of natural numbers.

For search problems, the cost function F is generally based on the so-called *distance to feasibility*, which accounts for the number of constraints that are violated.

For optimization problems, the cost function F must also take into account the objective function of the problem. Therefore the cost function is typically defined as a weighted sum of the value of the objective function and the distance to feasibility (which accounts for the constraints). Normally, the highest weight is assigned to the constraints, so as to give preference to feasibility over optimality.

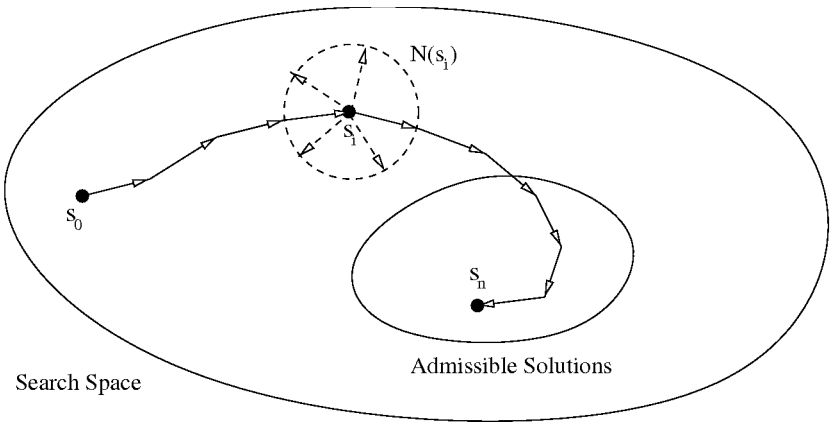

Figure 1: A pictorial representation of local search

In some optimization problems, the search space can be defined in such a way that it represents only the feasible solutions. In this case, the cost function generally coincides with the objective function of the problem.

2.3 Local Search Algorithms

A *local search algorithm*, starting from an initial solution $s_0 \in S$, iterates using the moves associated with the definition of neighborhood, such that it navigates the search space. At each step it makes a transition between one solution s to one of its neighbors s'. When the algorithm makes the transition from s to s', we say that the corresponding move t_m has been *accepted*, and we also write that s' is equal to $s \oplus t_m$.

The selection of moves is based on the cost function, which as explained above accounts for the number of violated constraints, and, for optimization problems, also on the objective function of the problem. The precise way such selection takes place depends on the specific local search technique, as explained in Section 3

The initial solution s_0 can be constructed by some other algorithm or generated at random [82, 121].

3 Basic Local Search Techniques

In the section we illustrate the three most popular local search techniques proposed in the literature. In details, in Section 3.1 we discuss a family of techniques that are grouped under the name of hill climbing. In Sections 3.2 and 3.3, respectively, we introduce two other techniques, namely Simulated Annealing and Tabu Search.

Simulated Annealing and Tabu Search represent two essentially different approaches for improving the simple idea behind hill climbing. Specifically, the first relies on probabilistic, memoryless decisions, whereas the latter is based on the use of memory of previously visited solutions. Therefore, the methods correspond to three different philosophies: simple, probabilistic-driven, and memory-based.

3.1 Hill Climbing

Among local search techniques, the simplest ones are based in some form of *hill climbing*. The term comes from the goal of maximizing a certain function via an iterative improvement scheme of selecting at each step a move that improves the value of the objective function or reduce the distance to feasibility.

Although people tend to identify as 'hill climbing' as large variety of techniques, in this paper we will refer to hill climbing as all the techniques based on the idea of making only moves that either improve the cost function or leave its value unchanged. This means that they never perform worsening moves, although they can perform *"sideways"* moves.

We now list a set of common hill climbing techniques:

- The most well-known form of hill climbing is the so-called *steepest hill climbing* (SHC) technique. At each iteration SHC selects, from the whole neighborhood $\mathcal{N}(s)$ of the current solution s, the element $s' = s \oplus t_m$ which has the minimum value of the cost function F. The procedure accepts the move t_m only is it is an improving move. Consequently, it stops as soon as it reaches a local minimum.

  As an example, the GSAT procedure proposed in [106] is a variant of SHC. The difference stems from the fact that GSAT accepts also sideways move and it breaks ties arbitrarily. Therefore GSAT has the capability of navigating *plateaus*, whereas standard SHC is trapped by strict local minima.

- Another popular hill climbing technique is *random hill climbing* (RHC). This technique selects at random (with or without a uniform probability distribution) one element $s' = s \oplus t_m$ of the set $\mathcal{N}(s)$ of neighbors of the current solution s. The move t_m is accepted, and thus s' becomes the current solution for the next iteration if t_m improves or let equal the cost function, otherwise s remains the current one for the next iteration.

- A recently proposed hill climbing technique is the *min-conflict hill climbing* (MCHC) of Minton *et al* [80], which combines features of SHC and RHC. MCHC is specifically designed for problems in which the elements of the search space are composed of a set of assignments of integer values to variables, and moves are defined as changes of the value of one variable (Constraint Satisfaction Problems).

  In MCHC, the selection of the move is divided into two steps, which are done using the two techniques. Specifically, MCHC first looks randomly for one variable of the current solution s that is involved in at least one constraint violation. Subsequently, for the variable v, it selects among only the moves of $\mathcal{N}(s)$ that change the value of v, the one that creates the minimum number of violations (arbitrarily breaking ties).

Both RHC and MCHC accept the selected move if only if the objective function value is improved or is left at the same value. Therefore, like GSAT, they navigate *plateaus*, but are trapped by strict local minima.

Their stop criterion is based on the number of iterations without improving the value of the best solution. Other *ad-hoc* early termination procedures are generally used such as stopping the iteration process if the objective function has crossed a certain value. In principle, they could also stop when they reach a strict local minimum. Unfortunately, though, in general they do not recognize such a situation.

Other forms of hill climbing, like for example the *fast local search* technique of Tsang and Voudouris [112], can be seen as improvements of these basic ones, and are discussed in Section 4.

3.2 Simulated Annealing

Simulated Annealing (SA) was proposed by Kirkpatrick *et al*[68] and V. Cerny [21] and extensively studied by Aarts *et al* [1, 115] and many other researchers. The method got that name after an analogy with a simulated controlled cooling of a collection of hot vibrating atoms.

The process starts by creating a random initial solution s_0. The main procedure consists of a loop that generates *at random* at each iteration a neighbor of the current solution.

Given a move t_m, let's call Δ the difference in the cost function between the new solution and the current one, i.e., $\Delta = f(s \oplus t_m) - f(s)$. If $\Delta \leq 0$ the new solution is accepted and it becomes the current one. If $\Delta > 0$ the new solution is accepted with probability $e^{-\Delta/T}$, where T is a parameter, called the *temperature*.

The temperature T is initially set to an appropriately high value T_0. After a fixed number of iterations, the temperature is decreased by the *cooling rate* α, so that $T_n = \alpha \times T_{n-1}$. In general α is in the range $0.8 < \alpha < 1$.

The procedure stops when the temperature reaches a *low-temperature region*, that is when no solution that increases the objective function is accepted anymore. In this case we say that the system is *frozen*. Note that for "low temperatures" the procedure reduces to a RHC when T tends to zero.

There are few control parameters of the procedure: the cooling rate α, the number of iterations at each temperature, and the starting temperature T_0. Obviously, there are many other decisions which affect the performance, namely the choice of a search space, the neighborhood and the cost function (see e.g., [83, 84, 100, 40]).

The one described above is the most common form of SA. Nevertheless, many variants of the above scheme have been proposed for SA. For example, the way the temperature is decreased, i.e. the *cooling schedule*, can be different from the one mentioned above, which is called *geometric* and is based on the parameter α.

For example, two other cooling schemes, called *polynomial* and "*efficient*" respectively, are described in []. We do not discuss them in details, but we just mention that they are both based on monitoring the quality solutions visited at the given temperature T_n, and choosing the new temperature accordingly. Specifically, in the polynomial scheme, the new temperature T_{n+1} is chosen on the basis of the standard deviation of the cost function in all solutions at temperature T_n. Similarly, in the efficient scheme, T_{n+1} is based on the number of accepted moves at temperature T_n.

Furthermore, there are some deterministic variants of the basic probabilistic

acceptance rules of SA given in [21, 68]. Moscato and Norman [87] use the scheme below, which is a variant of a deterministic update proposed in [86].

- if $\Delta \geq 0$, accept the move only if $\Delta \leq T$
- if $\Delta < 0$, accept the move only if $\Delta < -T$

As for the traditional SA, for low temperatures the procedure reduces to RHC. However, for any intermediate regime it will only select improving moves if they are at least better than a certain threshold. This scheme was designed due to the previous conclusion that it is the gradual definition of local optima, and not the stochastic jumping over minima as generally believed, what is the main mechanism behind successful implementations of SA [86]. This deterministic update procedures are generally known under the denomination of *threshold accepting* [8, 34, 38, 50, 78].

3.3 Tabu Search

As mentioned in the beginning of Section 3, Tabu Search (TS) is a method in which keeping memory of features of previously visited solutions has a fundamental role. We discuss in this section only a basic version of TS, which makes use of memory in a limited way. Some more complex features will be briefly discussed in Section 4, but for an exhaustive treatment of TS we refer to the recent, comprehensive book on the subject [52].

The basic mechanism of TS is quite simple: At each iteration a subset $Q \subseteq \mathcal{N}(s)$ of the neighborhood of the current solution s is explored. The member of Q that gives the minimum value of the cost function becomes the new current solution independently of the fact that its value is better or worse than the value in s.

To prevent cycling, there is a so-called *tabu list*, which is the list of moves which it is forbidden to execute. The tabu list comprises the last k moves, where k is a parameter of the method, and it is run as a queue; that is, whenever a new move is accepted as the new current solution, the oldest one is discarded.

Notice that moves, not solutions, are asserted to be tabu. Therefore, a move t_m can be tabu even if when applied to the current solution s it leads to an unvisited solution. In other words, the basic TS scheme avoids visiting not only previous solutions, but also solutions having features presented in solutions already visited.

For this reason, there is also a mechanism that overrides the tabu status of a move: If in a solution s a move t_m gives a *large* improvement of the cost function, then its tabu status is dropped, and the solution $s \oplus t_m$ is accepted as the new current one.

More precisely, we define an *aspiration function* Af that, for each value of the objective function, returns another value for it, which represents the value that the algorithm aspires to reach from the given value. Other aspiration functions not generally related to the objective function are also considered in some implementations. Given a current solution s, the cost function F, and the best neighbor solution $s' \in Q$, if $F(s') \leq Af(F(s))$ then s' becomes the new current solution, even if the move t_m that leads to s' has a tabu status.

In some proposals, it is not the move itself but rather some *attributes* of it to be considered tabu. The attributes that make a move tabu are generally known as *tabu-active* ones. The choice of the tabu-active attributes is obviously problem dependent.

The TS procedure stops either when the number of iterations reaches a given value or when the value of the objective function in the current solution reaches a given lower bound.

The main control parameters of the basic TS procedure are: the length of the tabu list, the aspiration function Af, and the cardinality and definition of the set Q of neighbor solutions tested at each iteration.

Notice that the tabu list needs not to be physically implemented as a list. More sophisticated data structures can be used to improve the efficiency of checking the tabu status of a move. For example, some hashing mechanisms are used in [122].

One of the key issues of TS is the *tabu tenure* mechanism, i.e., the way we fix the number of iterations that a move has to be tabu. The basic mechanism described above, which is based on the fixed-length tabu list, has been refined and improved by several authors.

A first improvement, proposed in [110] and commonly accepted, is the employment of a tabu list of variable size. Specifically, the size of the tabu list is kept at a given value for a fixed number of iterations, and then it is changed. For setting the new length, there is a set of candidate lengths, and they are used circularly.

A further improvement of this idea is the one proposed by Gendreau *et al* [49]: Each performed move is inserted in the tabu list together with the number of iterations *tab_iter* it is going to be in the list. The number *tab_iter* is randomly selected between two given parameters $tabu_{min}$ and $tabu_{max}$ (with $tabu_{min} \leq tabu_{max}$). Each time a new move is inserted in the list, the value *tabu_iter* of all the moves in the list is updated (i.e. decremented), and when it gets to 0, the move is removed.

Regarding the aspiration function, we have mentioned that several criteria have been proposed in the literature. The most common one is the following: If a move leads to a solution that is better than the current best solution found so far, it is accepted even if it is tabu. In other words, assuming s^* is the current best solution, the aspiration function Af is such that $Af(f(s)) = f(s^*) - 1$ for all s (assuming the codomain of f are the integers)q.

In some cases, the aspiration mechanism is used to protect the search form the possibility that in a given state all moves are tabu. In such cases, the aspiration function is set in such a way that at least one moves fulfils its criterion, and its tabu status is removed.

In other cases, it is set in such a way that, if a move that has a big impact on the solution is made, the tabu status of other *lower-influence* moves is dropped. The underlying idea is that after a big change, the effect of a move has changed completely, therefore there is no reason to keep it tabu.

Other types of aspiration functions are defined in [52].

3.4 Case Study No. 1: Radio Frequency Assignment

The first case study we discuss is the application of local search to the *Frequency Assignment* problem in the context of mobile phone systems.

In Section 3.4.1 we state the definition of the problem. In Section 3.4.2 we describe the search space we choose for the solution and the definition of the neighborhood relation and the cost function. The application of the local search basic techniques to the problem is discussed in Section 3.4.3.

3.4.1 Definition of the Problem

It is given a set of n *cells*, c_1, c_2, ..., c_n and a set of m frequencies (or channels) 1, 2, ..., m. For each cell c_i it is given a positive integer r_i, called the *request* of c_i, which denotes the number of frequencies requested by the cell. An assignment is a function which associates to each cell c_i a set of frequencies $F_i = \{f_{i1}, \ldots, f_{ir_i}\}$ of cardinality r_i. The solution of the problem is subject to the following constraints.

Separations: We call distance between two channels i and j the value $d(i,j) = |i-j|$. It is given a non-negative integer matrix $S_{n\times n}$, called *separation matrix* such that s_{ij} represent the minimum distance between frequencies assigned to cell c_i and cell c_j. The following relations must hold for all i, j, h, k such that $1 \leq i \leq n,\ i \leq j \leq n,\ 1 \leq h \leq r_i,\ 1 \leq k \leq r_j$

$$d(f_{ih}, f_{jk}) \leq S_{ij}$$

The diagonal elements S_{ii} of the separation matrix determine the minimum distance between frequencies assigned to the same cell (the elements below the main diagonal are ignored).

Illegal channels and preassignments: Is it given a matrix $P_{n\times m}$ whose elements belong to the set $\{-1, 0, 1\}$, which represents limitations on the use of channels. In particular, for all i, j such that $1 \leq i \leq n, 1 \leq j \leq m$ we have necessarily that

$$\begin{array}{llll} \text{if} & P_{ij} = -1 & \text{then} & j \notin F_i \\ \text{if} & P_{ij} = 1 & \text{then} & j \in F_i \end{array}$$

The problem consists in finding an assignment that satisfies all constraints. The problem is a search problem and no objective function is defined (more complex versions of the problem can be found in the cited papers).

3.4.2 Solution by Local Search

The search space for the problem is defined by representing each solution by means of an $n \times m$ binary matrix X, such that $x_{ij} = 1$ if j is one of the channels assigned to the cell c_i, $x_{ij} = 0$ otherwise.

The initial solution is generated at random, in such a way that satisfies the requirements, the preassignments and the illegal channels, but neglects the separations. It is easy to see that a random solution of this kind can be obtained in linear time.

The neighborhood relation is defined by the changes of one assignment for one cell to a different frequency. More formally, a move is identified by a triple (i, j, k) such that c_i is a cell, j is one of the frequencies assigned to c_i (i.e. $x_{ij} = 1$) and k is one of the frequencies not assigned to i (i.e. $x_{ik} = 0$).

We impose the limitation in the neighborhood definition that a move must maintain the satisfaction of the preassignments and illegal channels constraints. That is (i, j, k) is a possible move if $P_{ij} \neq 1$ and $P_{ik} \neq -1$. This way we implicitly define the search space as the set of matrices X that satisfy both the requirements and the preassignment and illegal-channel constraints.

Since the problem is a search problem and there is no objective function to take into account, the cost fuction to be used is solely related to the violated constraints. Moreover, given that the preassignment and the illegal-channel constraints are satisfied by all elements of the search space, only the separation constraints need to be dealt with in the cost function. Therefore, we simply define the cost function as the number of separation violations associated with the solution.

3.4.3 Application of Local Search Techniques to Frequency Assignment

Local search techniques have been applied to this problem by several authors. For example, TS has been applied in [10, 20, 29], while a simulated annealing algorithm has been developed in [29, 39].

(* ADD TSANG AND VOUDOURIS *)

We do not discuss here in details the various choices and the parameter setting performed in the cited papers. We just mention the tabu mechanism used for running.

Costa [29] proposes the following tabu mechanism. When a move (i, j, k) is performed, the pair (i, j) is inserted in the tabu list, with the meaning that for all h, a move of the form (i, h, j) is not allowed for a number of iterations equal to the tabu list length. That is, the pair (i, j) is the (only) tabu-active attribute of a move (i, j, k), which is stored into the tabu list.

In [20] there is also another tabu mechanism based on the frequency a move is performed in the full run. That is, if a move has been performed too many times since the beginning of the run, it is made tabu for a given number of iterations. This is a form of long term memory, which will be discussed in Section 4.2.3.

All the cited papers report successful results for real instances of the problem. In addition, Costa [29] claims that TS is clearly superior to SA for this specific problem.

3.5 Applications of basic local search techniques

There are many papers which report on the application of SA to scheduling and timetabling problems.

Wills and Terrill present a SA approach for the Australian State Cricket Season game scheduling problem in [119], Teodorovic, Krcmar-Nozic, and G. Stojkovic applied SA to the airline-seat inventory control [111], Brusco and Jacobs applied it to the flexible labour scheduling problem [12]. Sridhar and Rajendran present their SA approach to scheduling in cellular manufacturing systems in [109] while Mittenthal, Raghavachari, and Rana deal with the single machine scheduling problem with non-regular penalty functions [62]. Jeffcoat and Bulfin study the resource-constrained scheduling problem [63] and Wong and Wong study short-term hydrothermal scheduling problems in [120].

Starting from the original papers which studied threshold accepting applied to the TSP [86, 38], the deterministic update was applied to several problems in [8, 91, 50, 78, 42, 34].

There are many other applications of the basic SA mechanism available on the literature for other OR problems. We refer to [26] for applications before 1988. Koulamas, Davis and Turner present a survey of other applications of SA to OR problems in [70]. We also refer to [85] for other references on applications of SA to combinatorial optimization problems.

There are also many applications of TS to timetabling and sequencing problems available on the literature. Hertz [58] has applied Tabu Search to timetabling problems. Carmusciano and De Luca Cardillo, have presented an algorithm with features of both SA and basic TS in [19]. Chan and Sheung have applied TS to a roster scheduling problem which comes from an air cargo terminal [22]. Porto and Ribeiro applied TS to the task-scheduling on heterogeneous multiprocessor systems. The execution order must be defined taking into account the precedence constraints of the tasks [95]. França, Gendreau, Laporte and Muller have used TS for the multiprocessor scheduling problem with sequence dependent setup times [46]. Kim and Park have conducted a comparison study of TS and SA for the problem of scheduling orders on identical parallel machines. Orders are split into multiple jobs and a job is processed on one of the (parallel) machines while the objective is to minimize the holding costs of orders [94]. Agnetis *et al.* have applied it to the joint part/tool scheduling problem in a flexible manufacturing cell [3]. Dorn, Girsch, Skele and Slany [36] have implemented a comparative study of their implementations of Tabu Search and genetic algorithms (GA) for some scheduling problems from a steel making plant which had many, sometimes overly contradictory, constraints. Dorn, Kerr, and Thalhammer [37] have also applied TS for improving the robustness of schedules using a perturbation and repair mechanism based on local search.

Crauwels, Potts, and VanWassenhove [34] have performed a computational study on a large set of test instances using RHC, a GA, a SA with a stochastic and a deterministic update, and a TS. The problem was the single machine scheduling with batch set-up times where the objective was to minimize total weighted completion time The latter is reported as being superior when in some cases while the GA is

superior in others.

4 Improvements on the Basic Techniques

In this section we propose variants and improvements of the basic local search techniques discussed in Section 3. We start providing in Section 4.1 general issues and motivations, and in the following sections we present some of the most prominent ideas.

4.1 Motivations

We discuss here two general issues that inspire the research on improving the basic local search techniques. We will see later that the proposals are indeed related to these issues, namely "diversification vs. intensification" (also related to the "exploration vs. exploitation" dilemma) that we introduce in Section 4.1.1 and the "learning problem" mentioned in Section 4.1.2, in addition to efficiency considerations.

4.1.1 Diversification and Intensification

One of the main issues of local search techniques is the way they deal with the local minima of the cost function. Even if the search procedure employs some specific mechanism for escaping them (like TS), a local minimum still behaves as a sort of *attractor*. Intuitively, when a trajectory moves away from a local minimum and steps through a solution "near" to it, even though it is not allowed to go back to the minimum itself, it still tends to move "forward" it instead of moving in an "opposite" direction.

For the above reason, the search procedure needs to use some form of *diversification* strategy that allows the search trajectories not only to escape a local minimum but to move "far" from it thus avoiding this sort of *chaotic trapping* (as called in [5]) around the local minimum.

Notice that here the terms near, far, forward, and opposite are used in an intuitive way, without referring to any metric of distance. A more formal discussion is given for example in [5, 6].

On the other hand, for practical problems the landscape of the objective function is usually such that the objective function is correlated in neighbors (and near) solutions. Therefore, once a good solution is found, it is reasonable to search in the proximity of it for a better one. For this reason, when a local minimum is reached the search should be in some way *intensified* around it. Moscato [83, 84] discusses the relevance of *correlation of local optima* of the objective function m_P and its relation with the chosen representation.

In conclusion, the search algorithm should be able to balance the two sometimes conflicting objectives; it should diversify and intensify by moving outside the attraction area of already visited local minima, but not to far from it.

Several strategies have been proposed in the literature to solve this issue by giving the appropriate quantity of each in different phases of the search (see [54, 52]).

4.1.2 Learning while Searching

One characteristic of some local search techniques, namely Hill Climbing and Simulated Annealing, is the fact that they are completely *oblivious*. That is, they do not maintain any information about the past history of the search. Only a limited form of memory is used in Tabu Search through the tabu list mechanism, which preserves track of the last performed moves.

On the contrary, it is a common intuition that during search there is a lot of other information gathered from the trajectory followed and the evaluations made that could be learnt. Such information could then be used at a latter stage of the search. For example, as advocated by Battiti in [5], it should be possible to learn, with a reasonable overhead of computational time, some part of the landscape of the objective function so as to improve search effectiveness.

Such information can be used either to tune online the search parameters (e.g. the tabu list length) or to model the cost function and/or the neighborhood relation. The two proposals of Section 4.2.3 and 4.2.3 to improve the tabu mechanism of Tabu Search, and the cost function modification scheme described in Section 4.3.2 are a limited form of such "learn and react" approach.

4.2 Modeling the Neighborhood Relation

In this section, we discuss some improvements related to the notion of neighborhood. Specifically, in Section 4.2.1 we discuss how the candidate neighbors can be selected in order to improve the efficiency of the search process. In Section 4.2.2 we show how the interleaving of different neighborhood structures can help in effectively navigating the search space. Finally, in Section 4.2.3 we show how the neighborhood relation can be modeled by modifying the tabu mechanism of TS.

4.2.1 Neighborhood Sampling

Some local search techniques, e.g. SA and RHC, provide for drawing one random move at each iteration. Conversely, some others, e.g. Steepest Hill Climbing, require the exploration of the full neighborhood of each solution.

Regarding the latter, in many applications the neighborhood of a solution might be so large that its exhaustive exploration could be impractical, or at least inefficient. This consideration highlights the existence of an efficiency tradeoff between exhaustiveness of exploration versus speed of performing the selection. Therefore, even the algorithms that in principle should explore the full neighborhood, in practice they usually sample at least a part of it.

The simplest strategy for sampling the neighborhood is to select randomly a fixed share of it at each iteration. Such strategy thus involves a new parameter, which we call *neighborhood share*, that determines the share of the neighborhood examined, and which needs to be adjusted experimentally for the problem under consideration.

In order to improve this idea, several attempts have been done to identify the specific moves that are more promising, without fully evaluate all of them. We now

describe two of these approaches.

Hierarchical Cost Function. The first idea is to define the cost function in a *hierarchical* way, so that the components are considered strictly more important then all those belonging to lower levels of the cost function. The evaluation of the cost function in a candidate neighbor is done incrementally starting from the most valuable components. Whenever, based on the evaluation of some levels of the cost function, the neighbor is recognized as "not promising" the evaluation is interrupted, and the neighbor is discarded. This way, for many solutions, the procedure is not forced to make the full evaluation of the cost function but only a part of it, thus saving computational time. This idea is used in [27] for the high-school timetabling problem. The relevance of hierarchical objective functions had been previously discussed in [84].

Recognition of "natural hierarchies" in the objective function might improve the performance of the metaheuristic. For instance, a so-called *Morph-based* variant of SA has been applied to a variant of the bin-packing problem by Brusco *et al.* [15]. The so-called "morphs" are basically lists of similarly-sized items which attempt, for each item, to help guide the SA. In essence, they play the same role of "candidate lists" for Basic Tabu Search implementations.

Candidate List Strategies. A second mechanism is based on the idea of reusing the evaluation made in the previous explorations of the neighborhood. The intuition is that if a move m was good (bad) in a given solution s, very likely it will be still good (bad) in the neighbor $s' = s \oplus m'$ obtained by performing another move m'. Based on this intuition, during each neighborhood evaluation not only the best move is selected, but also the set of the other promising moves. In the subsequent iterations the moves previously selected, but not executed yet, are re-evaluated and possibly executed without redoing the full exploration (see e.g. [52, Section 3.2]).

This idea is further improved in the procedure called *fast local search* in [112]. Such procedure is a steepest hill climbing applied to a problem whose search space is composed by the permutations of the set $\{1, \ldots, n\}$ (where n is the number of jobs), and a move (i, j) is a swap between two positions of the permutation. Specifically, during the exploration of the neighborhood, if for a given i none of the swaps (i, k) (for $k = 1, \ldots, n$) is a descending move, than the position i is marked as *off*. A position gets back *on* when it is part of a move that is executed. All positions are *on* at the beginning of the search. At each iteration, the exploration of the neighborhood does not include the moves made of two *off* positions. This is analogous to the concept of the *don't look bit* for iterative improvement schemes introduced by Bentley for the TSP [7].

The above mechanism allows to explore only a small fraction of the neighborhood without a substantial loss on good moves, and therefore it gives a large speed-up (16 times in [112]) to the procedure without deteriorating the performances of the search.

4.2.2 Interleaving Different Neighborhood Relations

For a given problem more than one neighborhood relation could be suitable for solving it by using local search techniques. For example, as reported in [50], there are two interesting neighborhood structures for the *flow shop scheduling* problem. Assuming that an element of the search space is represented as a permutation of jobs, the two choices are the following:

Insert neighborhood Remove a job from one position in the permutation and insert it in a new position. For example (4,**2**,3,1,5) and (4,3,1,**2**,5) are neighbors under this definition.

Swap neighborhood Swap the jobs from two positions in the permutation. For example (4,**2**,3,1,**5**) and (4,**5**,3,1,**2**) are neighbors under this definition.

In the experiment presented in [50], the swap neighborhood turned out to be more effective than the insert neighborhood, even though the dominance is not so clear. However, as proposed by Glover *et al* [54], it might be generally profitable to alternate in distinct phases of the search different neighborhood structures.

Glover and Laguna propose in [52, Section 4.3.1] to use two types of moves, T_1 and T_2, simultaneously, so that the neighborhood is composed by the moves of both types. They also propose to add an auxiliary penalty factor d to moves of one type, say T_2, so as to bias the search procedure toward choosing either all moves of type T_1 or to mixed ones. Assuming that moves of type T_2 provide a more radical change of the solution, a low value of d would bias the choice toward a larger diversification, whereas a large value would allow only moves of type T_1. In conclusion, the value d can be used as a control knob for diversification and intensification, and good results can be obtained by letting it oscillate during search between two opposite values.

Another example of using two neighborhood relation will be shown in Section 4.5 for the case study on the high school timetabling problem.

4.2.3 Improving the Tabu Mechanism

In this section we illustrate two proposals that use a learning approach to model the neighborhood by improving the tabu mechanism of Tabu Search.

Long Term Memory and Frequency Based Tabu Status The tabu mechanism as explained in Section 3.3 is based on the set of the most recent moves. As illustrated in details by Glover and Laguna in [52, Chapter 4], a form of longer term memory can also be included to improve the tabu mechanism.

For example, assume that moves are swaps and they are then characterized by a pair of indices (i, j). We can maintain in memory a matrix that stores how many times a certain move (i, j) has been performed and a vector that stores how many times a specific component i has been involved in a move. Using such data structures certain moves can be declared tabu based on the number of times they have been executed.

This form of tabu mechanism is called *residence-based.* Similarly it is ...

Glover and Laguna reviewed several techniques for exploiting the information stored in these arrays. We do not go in details of such techniques, and refer to the cited book for their detailed discussion.

Reactive Adjustments of Tabu List Length Battiti and Tecchioli [5, 6] propose, in what they call *Reactive* Tabu Search, a scheme of adjusting the tabu list length during search.

The underlying idea of their work is that the list length must be long when the search is near to a local minimum and short when far from all of them. This way the search, according to the diversification needs, is pushed away a "local attractor" when it has been running around it for a while, and it is free to move (because the tabu list is short) before getting to a new attractor.

We do not explain the scheme in details. Intuitively, the algorithm starts with tabu list of length 1 and it is increased by 1, whenever a solution is visited too often. Conversely, if no repetition of solutions occur for a given number of iterations, the length is decreased by one. In addition, when the number of repetitions of a given set of solutions gets above a given threshold, a more extreme mechanism, called *escape*, is triggered: A sequence of random moves, possibly biased away the repeated solutions, is performed.

The cost of this idea is that the algorithm needs to keep tracks of all the solutions visited during search in order to compute the repetitions, which requires a lot of momery space. To this regard, the cited authors propose an efficient way to store solutions, which is based on the use of a *hashing* mechanism.

4.3 Modeling the Cost Function

As explained in Section 2.1, the cost function of a typical local search procedure is composed by a weighted sum of a number of components. Some of the components come from the embedding of the constraints, while others are related to the objective function of the problem. Such weights reflect the relative importance of the various components of the objective function and the constraints. For example, the weights of the constraints in the cost function are generally fixed to the highest value, so as to privilege the search for feasibility w.r.t. the improvement of the objective function.

Even though the weights reflect the "real" importance of the corresponding components, to improve effectiveness their values might be modified during the search procedure so as to vary the landscape of the cost function. The temporary modification of the landscape of the cost function, known as *shifting penalty*, can help to improve search effectiveness [49].

Ishibuchi, Yamamoto, Misaki, and Tanaka [61] presented results for a flow shop scheduling problem where the due-date of each job is given as a fuzzy set. They also propose an approach based on a change of the objective function and they study several local search algorithms.

4.3.1 Basic Shifting Penalty

One form of shifting penalty proposed in the literature is based on the following idea (see e.g. [27]): The weight of some of the components (typically the constraints) is dramatically decreased at a certain stage of the search. After a fixed number of iterations it is raised back to its original value. This process is repeated for several times during the search.

An improvement upon this scheme is provided by Gendreau *et al* [49] in their tabu search procedure for the vehicle routing problem. In the problem they consider there are two types of constraints. They associated to them two (independent) weights, that we call α_1 and α_2 respectively, which vary adaptively during search according to the following scheme (for $i = 1, 2$):

1. At the beginning of the search we set $\alpha_i = 1$.
2. Every h moves (with h a positive integer-valued parameter):
   - if all the h solutions visited are feasible w.r.t. the infeasibility of the type associated to α_i then $\alpha_i := \alpha_i/2$;
   - if all the h solutions visited are infeasible w.r.t. the infeasibility of the type associated to α_i then $\alpha_1 := \alpha_i \cdot 2$;
   - if some solutions are feasible and some others are infeasible then α_i is left unchanged.

This scheme allows the search trajectory to step in and out the feasible region of the search space, thus allowing for a better navigation through local minima than using a fixed value for α_i.

This idea is further developed in the case study of Section 4.5.

4.3.2 Learning the Weights

In the work of Selman and Kautz [105] for the propositional satisfiability decision problem (SAT), the shifting penalty idea is pushed further: An independent dynamic weight is given not to the constraint types but to each single constraint (a clause in the formula, in their case). Such finer grain weighting turned out to be useful for dealing with problems with strong asymmetries, i.e. problems in which the number of constraints in which the variables are involved can be different by a large factor from variable to variable.

Kautz and Selman in [105] changed the weights after each local search run. In the work of Frank [47] the weights are updated after every single move. To be more precise, the weights of all the clauses that are not satisfied in a given solution are increased. The quantity by which they are increased, and their initial values, are the parameters of his learning scheme. We refer to the cited paper for the outcomes of the approach.

Similar work has been carried out by Morris [81] and Yugami *et al* [124] that employ some form of relaxation of constraints when a local minimum is reached.

The *guided local search* procedure of Tsang and Voudouris [112] is based on the idea of adding to the cost function some additional components which are specifically designed to escape from local minima. That extra components are chosen among the attributes of the elements of the search space in such a way that they get a high value on the local minimum under consideration.

4.4 Combining Local Search Techniques

In this section we report two possible ways of improving the effectivenss of local search based on combining among themself different local search techniques.

4.4.1 Combining Features of Different Techniques

Chiang and Russell present in [24] a hybrid of TS and SA for a vehicle routing problem with time windows. Specifically, they implement a SA algorithm and they study the possible enhancement obtained adding the short-term memory effect given by the tabu list. Such combination is claimed to compare favorably with previously reported results. Two different neighborhood relations are also considered and compared in their work.

4.4.2 Interleaving Different Techniques

In [102] it is shown an algorithm that interleaves TS with a RHC procedure. Specifically, each procedure, starting with the hill climbing, is let run as long as it meets its own stop criterion, after which the other one takes over starting from the best solution found by the previous one. The full process is repeated iteratively, and it stops when both procedures give no improvements for a given number of cycles.

The two procedures make use of two different neighborhood structures: The neighborhood for TS is constituted by *atomic moves* obtained by one simple local change. Conversely, the neighborhood for RHC is composed by a sequence of two moves called *double moves* (see Section 4.5.2 for their precise description).

The most time-consuming procedure is TS, and indeed as reported in [102] it is the one that gives the most significant improvement of the objective function. The RHC is inserted in the cycle for two different purposes: First, it generates the initial solution for the TS. In fact, the use of the TS starting from the random solution is too time consuming, and RHC instead represents a fast method to generate a reasonably good initial solution (which might still contain infeasibilities).

Second, after the TS has given no improvements for a given number of iterations, it proved to be useful to run the RHC on the best solution found. The reason for it is twofold: On the one hand, the RHC (using double move) might find improvements that the TS is not able to find at that stage. On the other hand, the RHC with double moves, even if it does not improve the solution (which is often the case), it makes substantial sideways modifications. Therefore, it "shakes up" the solution before the TS starts again to try to improve it. The idea underlying such a procedure is that after the TS has worked unsuccessfully for a given number of iterations, it is

useful to modify the solution so that, according to the diversification issue discussed in Section 4.1.1, the TS can start in a different direction.

This approach is further discussed in the case study on high-school timetabling given below.

4.4.3 Iterating Local Search

4.5 Case Study No. 2: High School Timetabling

The high-school timetabling problem regards the weekly scheduling for all the lectures of a high school. The problem consists in assigning lectures to periods in such a way that no teacher (or class) is involved in more than one lecture at a time, other side constraints are satisfied, and a given optimization function is minimized.

The problem has a large number of variants, depending on the country, on the type of school, and even on the specific school involved (see [101] for an overview). Many local search procedures have been applied to some variants of the problem: hill-climbing [123], simulated annealing [2], *tabu search* [30]. The specific optimization problem, and the corresponding local search procedure, we describe here is a simplification of the one provided in [102].

4.5.1 Definition

There are m classes $c_1, \ldots, c_m$, n teachers $t_1, \ldots, t_n$, and p periods $1, \ldots, p$. It is given a non-negative integer matrix $R_{m \times n}$, called *Requirements matrix*, where r_{ij} is the number of lectures that teacher t_j must give to class c_i. The unavailabilities of teachers are taken into account by introducing a binary matrix $T_{n \times p}$ such that $t_{jk} = 1$ if teacher t_j is available at period k, and $t_{jk} = 0$ otherwise. The mathematical zero-one formulation of the problem is the following

$$
\begin{aligned}
& find \quad x_{ijk} && (i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, p) && \\
s.t. \quad & \sum_{k=1}^{p} x_{ijk} = r_{ij} && (i = 1, \ldots, m; j = 1, \ldots, n) && (1) \\
& \sum_{i=1}^{m} x_{ijk} \leq t_{jk} && (j = 1, \ldots, n; k = 1, \ldots, p) && (2) \\
& x_{ijk} = 0 \text{ or } 1 && (i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, p) && (3)
\end{aligned}
$$

The meaning of the binary matrix $X_{n \times m \times p}$ is the teacher t_i teaches to class c_j in period k if and only if x_{ijk} is assigned to 1.

The above one is the theoretical formulation of the problem, which has been shown NP-complete by Even *et al* [41] and appears in [48, SS19, p. 243]. However, the practical problem includes two other types of constraints. First, each class for a given set of periods, must necessarily be involved in one lecture (*class covering* constraints). Second, some pairs of lectures must be scheduled simultaneously (*simultaneity* constraints). We refer to [102] for their precise formulation.

The objective function (to minimize) is a weighted sum of a set of eight components. They range from "windows" in teachers' schedules, to minimum and maximum teaching loads, to teachers preferences. We do not list them here and refer again to [102] for their definition.

4.5.2 Representation of the Problem

The definition of the search space is such that a timetable is represented as an integer-valued matrix $M_{m \times p}$ such that each row j of M represents the weekly assignment for teacher t_j. In particular, each entry m_{jk} contains the name of the class that teacher t_j is meeting at period k. The value $m_{jk} = 0$ represents the fact that t_j is not teaching at period k.

This representation (used also in [27]) allows for the definition of simple and natural types of moves, which permit to navigate effectively the search space. In addition, this is the representation upon which generally the persons that do manual timetabling reason, and therefore, it is also suitable for interactive timetabling optimization with manual corrections.

Infeasible timetables are also included in the search space. The cost function is thus the objective function augmented with three components coming from the measure of the number of infeasibilities. In particular, we count: (1) the number of times that either two teachers teach to the same class in one period or a class is uncovered (2) the number of times a simultaneous assignment is missed, and (3) the number of times a teacher teaches when she/he is not available.

The possibility that a teacher teaches simultaneously to two (or more) classes is ruled out automatically in the representation chosen.

The weight given to the infeasibilities is set to a value higher than the weight of all other quantities. However, as shown below, such weight is allowed to vary during the search phase, in the way explained in Section 4.3.1.

The first type of move that we consider is the one that naturally fits in our representation. It is obtained by simply swapping two distinct values on a given row. That is, the lectures of a teacher t in two different periods p_1 and p_2 are exchanged between them, or, in case that one value is 0, one lecture is moved to a different period. We call a move of such type an *atomic move*.

Atomic moves applied to feasible timetables generally create infeasibilities assigning two teachers to the same class. For this reason, we consider more complex move types. In particular, we consider *double moves*, which are moves made by a pair of atomic moves, so that the second one "repairs" the infeasibility (or one of the infeasibilities) created by the first one by exchanging the lecture that conflicts with the one just inserted. If the first atomic move creates no infeasibilities, then there is no second move and the double move reduces to an atomic one.

4.5.3 Solution by Local Search

As already explained in Section 4.4.2, the core of the algorithm's search engine is a TS with the neighborhood constituted by atomic moves interleaved with a phase of RHC using double moves. The initial solution is obtained by scheduling the lectures

for each teacher randomly, respecting the requirement matrix (or it can be obtained from previous runs, in case of interactive timetabling). Then, the RHC starts to work on the random timetable until it makes no improvements for a given number of iterations. At this point, the TS starts and goes on until it makes a given number of iterations without improving. The whole process (RHC + TS) is repeated on the best solution found, and it stops when it gives no improvements for a given number of times.

4.5.4 Adaptive Relaxation

During the RHC phase the weight of the infeasibilities is set to the value W, which is higher than all the other weights involved in the objective function. Conversely, during the TS stage, such weight is dynamically adjusted in the following way: For each of the three sources of infeasibilities (see Section 2.1), namely (1) clash of teachers or uncovered classed, (2) simultaneity of lectures, and (3) unavailability, we multiply W by a real-valued factor α_i (for $i = 1, 2, 3$) which varies according to the scheme propose in Section 4.3.1.

The real-valued parameter γ by which α_i is multiplied or divided is randomly selected (each time) in the interval $[1.8, 2.2]$, whereas in [49] (as explained in Section 4.3.1), γ is deterministically set to 2. This randomization improves the performances because it creates during search a full variety of ratios between different components of the cost function, instead of using the deterministic ratios created by multiplying and dividing only by 2, that could bias the search.

Differently from [49], we bound the value of each α_i by two constants $\alpha_{i,min}$ and $\alpha_{i,max}$. Thus, if α_i gets a value higher than $\alpha_{i,max}$, then it is set to $\alpha_{i,max}$. Similarly, when it goes below $\alpha_{i,min}$, it is set to $\alpha_{i,min}$. The value of α_i is limited to prevent it from getting too high (resp. too low) after a long sequence of infeasible (resp. feasible) solutions. In fact, in that case, too many iterations would be wasted in order to get back to values comparable to the other values in the objective function, when such a sequence is interrupted. In addition, the three constants $\alpha_{i,max}$ are set to different values (in the experiments $\alpha_{1,max} = 1$, $\alpha_{2,max} = 10$, $\alpha_{3,max} = 3$, and $\alpha_{i,min} = 0.01$, for $i = 1, 2, 3$), so that there is no stable situation in which different sources of infeasibilities coexist.

4.5.5 Tabu List Management

We employ a tabu list of variable size, as explained in Section 3.3. In addition, we enforce two different tabu mechanisms, although we make use of a single tabu list (similarly to [30]). The first mechanism states that a move $\mathcal{M} = \langle t_i, p_j, p_k \rangle$ is declared as 'tabu' if the exact triple $\langle t_i, p_j, p_k \rangle$ appears in the tabu list. The second mechanism, which has a shorter term effect, states that $\mathcal{M}$ is tabu if either (t_i, p_j) or (t_i, p_k) appear in the last-inserted I_{st} elements of the tabu list, where I_{st} is another user-specified parameter such that $I_{st} < I_{min}$. In other words, the assignments of teacher t_i at times p_j and p_k cannot be swapped for I iterations and they cannot be singularly exchanged with any other assignment for the shorter period of I_{st} iterations.

4.5.6 Neighborhood Sampling

The size of the neighborhood of any solution s, $\mathcal{N}(s)$ is given by

$$| \mathcal{N}(s) | \ = \frac{mp(p-1)}{2}$$

which is the number of teachers times the number of unordered pairs of distinct periods. However, moves that swap two identical lectures do not actually change the timetable; therefore they are considered *illegal* and they are not included in the neighborhood. This fact decreases the number of *legal* moves by a factor that depends on the number of classes n and on the requirement matrix R (in some practical cases we have studied it has the value of approximately 30%).

Due to the adaptive relaxation it is very difficult to find a way to predict the most promising moves. For this reason we choose to analyze the entire neighborhood of legal moves at each iteration.

However, there are certain moves that do not really affect the structure of the solution. The existence of a large number of such moves gives the possibility of finding a move with minimal effect that gives a zero-cost variation. Therefore, the TS does not choose worsening moves (unless we use a very long tabu list), and this fact prevents the TS from effectively exploring the search space. On the other hand, such moves might produce an improvement in the objective function (e.g., by filling holes). For the above reason, we define the concept of *semi-illegal* moves which intuitively are those moves that have a small effect on the structure of the timetable. The constraint we impose on semi-illegal moves is that they can be made only if they *strictly* improve the (current) objective function. We refer to [102] for the definition of semi-illegal.

As claimed in [102], the algorithm has given good results for schools of various types, and for different settings of the weights of the objective functions. For all cases, the timetable produced turned out to be better than the hand-made ones.

5 Combining Local Search with Other Techniques

In this section we describe search techniques that arise from the combination of local search with other search paradigm. Since local search techniques require the presence of an initial solution, the most natural combination is the use of a *constructive* heuristics for finding a solution, and the subsequent employment of local search for improving it.

Such combination, together with different way of "blending" constructive heuristics and local search, is discussed in Section 5.1. In Section 5.2, we discuss the combination of local search with genetic algorithms, which originated a new line of research, called *memetic algorithms*, that has become very active in the recent years. Finally, other combinations are listed in Section 5.3.

5.1 Local Search and Constructive Heuristics

There is a variety of techniques for combinatorial optimization problems which are based on the use *in tandem* of a constructive method followed by a local search step.

For example, Yoshikawa *et al* [123] combines a sophisticated greedy algorithm, called *Really-Fully-Lookahead*, for finding the initial solution and a local search procedure for the optimization phase. The local search algorithm employed is a the *Min-Conflict Hill-Climbing* (MCHC), defined by Minton *et al* [80].

Others, like GRASP [55], propose an iterative scheme which uses this tandem as an inner loop. Starting with a constructive method, a local search scheme is later applied. Some kind of adaptation will guide the constructive phase for a new attempt which again will be followed by a local-search step.

Solotorevsky *et al*[107] employ a propose-and-revise rule-based approach to the course timetabling problem. The solution is built by means of the *Assignment Rules*. When the construction reaches a dead-end, the so-called *Local_Change Rules* come into play so as to find a possible assignment for the unscheduled activity. However, they only perform a single step before restarting the construction, and their aim is only to accommodate the pending activity, without any look-ahead mechanism.

5.1.1 Local Search on Partial Solutions

In [103] the author proposes a more complex way of combining (backtracking-free) constructive methods and local search techniques. The technique incrementally constructs the solution, performing a local search on partial solutions each time the construction reaches a dead-end. Local search on the space of partial solutions is guided by a cost function based on three components: the distance to feasibility of the partial solution, a look-ahead factor, and (for optimization problems) a lower bound of the objective function. In order to improve search effectiveness, it makes use of an adaptive relaxation of constraints and an interleaving of different look-ahead factors. The technique has been successfully experimented on two real-life problems: university course scheduling and sport tournament scheduling.

Another approach was employed in [125] where the authors combine constructive and local search methods in a different way. Their method finds a partial solution using local search, and then explores all its possible completions using a backtracking-based algorithm. Therefore, they also make use of local search on partial solutions, but they have no notion similar to the look-ahead factor in [103].

5.1.2 Local Search and "Matrioshka Instances"

To mention an example of a different strategy which can be extended to scheduling and timetabling, we will discuss a new hybrid methodology, which also tries to *blend* a constructive and local search step. It is being tested using instances of the Euclidean Traveling Salesman Problem (ETSP) [71].

Given a set of points in the euclidean plane, the algorithm first computes the *Delaunay Triangulation* (DT) and a subgraph of it, the *convex-hull*. There are many fast methods for doing this (the reader can refer to any textbook in Computational

Geometry), and the algorithm proceeds by creating an initial subtour using only the cities of the convex hull. This is a good starting point, since it is proved that the cities of the convex hull appear in the same order in the optimal tour.

We proceed iteratively selecting city (from the remaining ones not yet in the subtour) follow by the addition of the city to the subtour. The selection is based on the sum of length of the edges (incident to that city) of the DT. The city is inserted where it minimizes the cost (lenght) of the current subtour. A local search step is started using this city and its two adjacent cities in the subtour as candidates for further local search improvements. The approach is very promising due to an empirical low-complexity. This is a consequence that, at all steps involved, low-complexity algorithms from the field of computational geometry are being used.

We can try to formalize this algorithm to understand its generality. Given a certain ETSP instance $x \in I_P$, where the problem P is the ETSP, the instance having $N = |x|$ cities, we start with a instance x_0 which only contains the cities which are vertices of the convex-hull of the whole set x, a sequence of instances $x_0, x_1, \ldots, x_{N-|x_0|} = x$ is created such that each $x_i \subset x_{i+1}$ and $|x_i| + 1 = |x_{i+1}|$, for all $i = 0, \ldots, i = (N - |x_0| - 1)$. The subset symbol here is indicating the fact that all points of instance x_i are present in instance x_{i+1} thus we gave the informal name of *"matrioshka instances"* to approaches based on such a sequence.

For each instance x_i we are addressing the *same* problem P and under the assumption that a good heuristic for obtaining an approximating sequence of instances was used, we hope a local minimum for instance x_i is a good starting solution for x_{i+1} after a new city is added. It is important to remark that the convex-hull is the *optimal* tour for the instance x_0.

Which are then the general rules to be taken into consideration to construct an appropriate sequence of instances ? Unfortunately we are still unable to give a general answer for this question and further research is necessary. In the strategy above, the initial instance x_0 was selected to be the cities of the convex-hull based on the fact that every Euclidean TSP has an optimal solution that visits the cities on the boundary of the convex hull in the same order as if the boundary of the convex hull itself were traced [45], thus properties of the optimal solution to be exploited are certainly useful. For the ETSP, it may be convenient that the instance x_1 is defined by identifying a city c which maximizes the value of some kind of lower bound to the optimum tour length of instances x_1. There are several ways this can be achieved, one possible candidate function is the computation of the minimum length spanning tree of x_1, $MST(x_1)$, since we know that $MST(x_1) \leq opt(x_1)$. That will guarantee a "coarse" to "fine" sequence of instances. Other approaches, such as those based on *farthest-selection, cheapest-insertion* seem to be motivated on similar grounds. It seems reasonable that in scheduling or timetabling problems we can look at how much constrained are the individual elements of the instance in order to build such a sequence. A good rule-of-thumb might be that the most constrained elements should be early introduced.

5.2 Local Search and Genetic Algorithms: Memetic Algorithms

The first use of the term *'Memetic Algorithms'* (MAs) to denote hybrids of iterative improvement schemes with recombination operators and population based strategies appeared in [83]. In that paper, several issues concerning, at that time recent, hybrids of population-based strategies which use crossover operators and local search techniques are discussed. Since then, many other papers have been published in this area. A World Wide Web page[1] provides access to many available on-line references on this research field. The MAs approach has been applied to problems arising in sports timetabling, exam timetabling, course scheduling, parallel machine scheduling, job-shop scheduling, and single machine scheduling with setup times, due dates and early/tardy penalities. Other applications in combinatorial optimization include the graph bi-partitioning problem, the traveling salesman problem, quadratic assingment, set covering, graph coloring, set partitioning, generalized assignment, etc. Most of these references can be accessed from the MAs WWW page.

The main idea behind MAs is to use a population-based strategy which uses all available knowledge about the problem. Most of the implementations, up to now, have relied on local search and fast heuristics interspersed with the use of one or several recombination operators. When we iterate between these two processes, and when we have a local search procedure as our optimizing heuristic, we have a method that combines good features of local optima to create new points in the search space. These new "child" solutions would require further optimization before being recombined. In this case, we have a *Local-Search-Based Memetic Algorithm* which, in some cases, might be seen as a GA-like process but in the space of local optima of the underlying local search scheme.

In a MA, each individual of the population may be handling more than one solution for the problem[89]. The example given in Fig. 2 is only one type of memetic algorithm, others might use different neighbourhoods for each of the optimizing agents. However, the pseudocode includes a wide variety of them.

In the pseudocode, the population *Pop* is initialized with a FirstPop() procedure (which can rely on constructive heuristics, for instance) and then optimized with some local search algorithm (here called Local-Search-Engine()), which can also rely on some other metaheuristic to guide the process (SA, TS, Guided Local Search, etc). At each step of the "generations loop", some set of agents are selected (the set of "parents" S_par), and recombined using the information given by them and the instance. This process can be computationally high, and in those cases the word "crossover" does not generally apply. Some authors even consider solving the associated subproblems to optimality at this step. The resulting solution is reoptimized, evaluated according to the objective function, and incorporated to the population. The Mutate() function is generally some random perturbation of the current solution and it is optimized with the available local search algorithm (called 'Local-Search-Engine()' in the pseudocode).

Some authors are favourable to include a *population structure*. Sometimes agents

[1] See `http://www.densis.fee.unicamp.br/~moscato/memetic_home.html`

```
procedure Local-Search-based Memetic Algorithm;
  begin
    initializePopulation Pop using FirstPop();
    foreach individual i ∈ Pop do i := Local-Search-Engine(i);
    foreach individual i ∈ Pop do evaluator(i);
    repeat /* generations loop */
      for i := 1 to #recombinations do
        selectToMerge a set S_par ⊆ Pop;
        offspring := Recombine(S_par, x);
        offspring := Local-Search-Engine(offspring);
        Evaluate(offspring);
        addInPopulation individual offspring to Pop;
      endfor;
      for i := 1 to #mutations do
        selectToMutate an individual i ∈ Pop;
        i_m := Mutate(i);
        i_m := Local-Search-Engine(i_m);
        Evaluate(i_m);
        addInPopulation individual i_m to Pop;
      endfor;
      Pop := SelectPop(Pop);
      if (Pop has converged) then Pop := RestartPop(Pop);
    until termination-condition=True;
  end;
```

Figure 2: The Local-Search-Based Memetic Algorithm

are organized using a particular interaction topology, this affects the way the "reserved word" **selectToMerge** is going to be interpreted in this case. For instance, Moscato and Tinetti have proposed a hierarchical structure (a tertiary tree) of optimizing agents, each one of them performing local search using one of three different neighborhoods. Each agent (with the exception of those in the leaves of the tree and the root node) is both a *leader* and a *supporter* of another agent. A very simple set of interactions is established between individuals. For instance, if a solution found by a follower after a period of local search is better than the solution that currently is being considered by its leader, then the follower and leader interchange their solutions. Simple rules like this, which also organize the recombination operations, lead to a collective behavior which was shown to be better than multiple runs of the basic local search technique [89].

We remark that other MAs might use agents using heuristic methods interacting with exact procedures. Researchers in the MAs field claim that we can constructively use many existing algorithms to produce a much more robust optimization strategy. It is important to note that the limitations of some basic TS and SA schemes for

some problem domains also directed the researchers to create new types of hybrids [51, 84].

Today these methods are gaining a wide acceptance [43]. Cheng and Gen [23] have recently applied MAs to parallel machine scheduling problems. In this case a GA is used to evolve the job partition among machines and a local optimizer adjusts the job permutation. They concluded that it had outperformed both the GA approach and the conventional heuristics.

5.2.1 Recombining solutions

The Recombine(S_{par}, x) procedure is generally a well-crafted mechanism in most MAs implementations. For instance, in [87], the authors present a recombination operator for the TSP which tries to maximize the edges present in both parents, given the highest priority to edges which are simultaneously present in both of them.

A main issue MAs is to use *all* available knowledge about the problem to help the search process. If appropiately implemented, it may be possible to introduce a human interaction as another agent (or as part of the selection processes) to improve the search. This may be of particular interest in problems which may also involve perceptual issues related to design, esthetics, etc., all knowledge which can be difficult to represent in mathematical terms. This can be of relevance also in scheduling or timetabling situations where it is desirable that the FirstPop() population might have *handmade* solutions by some human expert. In addition, the expert might interact with the population in a variety of ways, via the Mutate() function for example. The human expert can also rule the SelectPop() procedure, for instance by eliminating some solutions according to her/his criteria.

5.2.2 Design and Implementation Issues

How can be sure that we have a good memetic algorithm implemented for the problem of interest ? This is a question with many facets. First of all, we consider as 'good' MAs those which have a small number of parameters that need to be tuned (or none, which is certainly better). The number of individuals used might be an essencially unavoidable parameter, but common sense dictates that in a good implementation the quality of the solutions found should monotonically improve for larger populations sizes. This is something which can be easily tested.

In a MA we have a matching of one (or several) heuristic and one (or several) recombination operators. If the computer code is well-structured, it is relatively easy to test the contributions of each by running experiments on the same set of instances, switching on and off the different procedures. For instance, a MA composed on some sort of selection, recombination, mutation and local search (in that order) sometimes reduces to a standard GA if the local search process is switched off. Radcliffe and Surry have used this type of analysis to show that a MA they implemented for the TSP is orders of magnitude faster than a GA. In that paper [98], leaving invariant all other algorithmic decisions, they also tested the benefits of a set of different recombinations.

There are other ways of understanding the performance of recombination operators. For instance, Hofmann studied the performance of the Strategic Edge Crossover of [87], as well as other "general purpose" recombination mechanisms previously introduced by Radcliffe using the TSP as test-bed of this general methodology. He proposed *correlation within Formae experiments* [59], and statistical tests then help to get a better understanding of what is necessary to preserve when recombining the information from the parents. Radcliffe and Surry have extended this type of analysis in[97].

Colorni, Dorigo and Maniezzo [27] also comment on the good results they had applying these techniques to high school timetabling problems. The model included a hierarchical structure for the objective function. The relevance of hierarchical objective function for the success of implementations of MAs was also discussed in [84].

5.2.3 Hybridization with other metaheuristics

It is also possible to build MAs using other metaheuristics for the optimization agents. In [87] the authors present a hybrid technique which uses a deterministic update procedure for a SA-like optimization step. In a similar line, Varanelli and Cohoon have proposed another hybrid technique which also tries to blend the benefits from GAs and the so-called "thermodynamic" approach [116]. Costa, particularly in Ref. [31], used a scheme similar to [87] although he makes use of Tabu Search steps. In Ref. [32] the authors rely only in hill-climbing without Tabu Search steps. This has some analogies with the approach used for the Binary Perceptron learning problem of [84]. For that problem, it was clear that only hill-climbing was not enough to find the optimal solutions. Hybridization of the MA using Tabu Search to diversify the population was necessary in this case.

Burke, Elliman and Weare developed MAs for highly constrained timetabling problems arising from university needs [18] as well as in a more recent collaboration with J. Newell [16]. They have used two types of mutation which they named as *light* and *heavy* (we note that a 'heavy' mutation can be described in terms of the RestartPop() procedure of the LS-based MA pseudocode described before). The former is implemented by choosing a number of events at random from any point in the timetable and rescheduled at some other legal period followed by an application of a hill-climbing algorithm. The so-called *heavy* mutation deterministically selects and disrupts one or more whole periods of a timetable. These mechanisms depart from the standard GAs approach to mutation which tends to be unbiased. Paechter, Cumming, Norman and Luchian have also presented results on a memetic timetabling system [93]. Their approach also makes use of various specialized forms of directed mutations which have a synergistic behaviour when used together. The structured population they use is based in the hierarchical type previously used for the TSP [89]. Cumming maintains several web pages that allow to edit and create timetables interactively with this system[2].

[2]See: `http://www.dcs.napier.ac.uk/~andrew/dneeps/about.html`

5.2.4 Other applications

A recent comparison on scheduling problems can be found in the paper by S. Rana *et al.* [99]. Crauwels, Potts, and VanWassenhove, have studied several local search based schemes for the single-machine scheduling with batching problem where the objective is to minimize the number of late jobs [35]. Glass and Potts have recently shown good results of an approach that can be classified as a MA in comparison against other metaheuristics for the flow shop scheduling problem [50]. Levine presents a steady-state algorithm for the airline crew scheduling problem and compares his results against exact algorithms based in branch-and-cut and branch-and-bound on a set of 40 real-world instances [75]. The MA has found the optimum in half of them and good results for 9 others.

In a preprint entitled *"Characterizing search spaces for Tabu Search"*[3], by C.R. Houck, J. A. Joines, and M. G. Kay, the authors presented a combination of both Tabu Search and Genetic Algorithms for a location-allocation problem, where regions around genetically determined sample points are marked as 'tabu'. They conclude that the combination compares favorably to the genetic algorithm in terms of increased computational efficiency. Kim, Hayashi and Nara presented a heuristic for the long term thermal unit maintenance scheduling problem where they also use Tabu Search and SA with recombination mechanisms and a population strategy [67]. Burke and Smith have presented a Memetic Algorithm for the Maintenance Scheduling Problem in [17]. Murata, Ishibuchi and Tanaka have presented other hybrids of GA and local search techniques which had high performance for flowshop scheduling problems [90].

Other applications might be outside the borderline of what can be considered as a MA. For instance, Hou, Ansari and Ren [60] applied a GA for multiprocessor scheduling, while Conta, Ferreira and Rebreyend have integrated list heuristics into a GA [28]. Kadaba and Nygard have used a GA for the automated discovery of parameters for mathematical heuristics, in the domain of computer-aided vehicle routing and scheduling problems. [66]. Haase [57] studies a single-stage capacitated lot-sizing problem with sequence dependent setup costs. In his model, continues lot-sizes are allowed and the setup state can be preserved over idle time. A priority rule based heuristic is used. Since the priority values are affected by paramenters a local search is performed to obtain low cost solutions. Padman and Roehrig have used genetic programming for selection of heuristics in constrained project scheduling [92]. These methods might be considered under the denomination of Genetic Programming.

5.2.5 Hybridization with systematic methods

Regarding the use of population-based hybrids which rely on systematic methdos for the local search, Médioni, Durand and Alliot proposed an approach based on a GA and a Linear Programming code for the generation of optimal trayectories for avoidance of conflicts in air traffic control problems [4]. Though exponential in the

[3]See: `http://www.fmmcenter.ncsu.edu/fac_staff/joines/papers/papers/tabula.ps`

worst-case scenario, the simplex algorithm for Linear Programming can be viewed as another useful local search algorithm for many problems appearing in practice. Bowen and Dozier have also investigated the use of hybrids of GAs with systematic methods in [11]. Morgan and Williams [77] apply a GA to search at a base level for the economic scheduling of electric power generation while they use a gradient local search tecnique to find local optima.

We believe that there are many other opportunities for the development of MAs based on exact, systematic procedures like *Branch-and-Bound*, *Branch-and-Cut* [65] or comercial codes for the resolution of mixed integer programming problems. For instance, S.E. Ling proposed a hybrid approach based in the integration of a GA with an assignment program written in Prolog for a timetabling application. Psarras *et al.* have combined the forces of Constraint Logic Programming with local search in a vehicle-fleet scheduling problem [96]. With the addition of a recombination step we believe an interesting MA would be implemented.

5.3 Other Combinations

Valenzuela and Williams [114] propose an approach different from MA to combine a heuristics with a population based strategy in. A GA is used to perturb the position of the cities (that is they are perturbing $x \in I_P$). Perturbation of TSP instances has been also proposed by B. Codenotti *et al.* (see [25] and references therein).

In [53] the authors use a similar approach to solve the graph coloring problem. Their method starts with a complete solution, found with a high-quality heuristics. Thereafter, it alternates a destructive phase, in which some nodes are *uncolored*, and a constructive one, in which nodes are *colored*. The uncoloring is guided by an estimation of the depth and the width of the current local minimum.

Blazewicz *et al.* [9] presented a heuristic approach that combines TS and linear programming for the deadline scheduling of multiprocessor tasks, that is where tasks may require more than one processor at a time to be excecuted.

Koulamas [69] has presented a *beam search* technique that is compared agains the results of a local search technique for the problem of scheduling two parallel semiautomatic machines which share the same server. The objective is to minimize the machine idle time resulting from the unavailability of the server. Sotskov, Tautenhahn, and Werner described algorithms based on local search and reinsertion techniques for permutation flow shop scheduling problems with batch setup times [108] (see also [118]).

5.4 Case Study No. 3: The NHL Scheduling problem

Daniel Costa applied a memetic algorithm to the National Hockey League scheduling problem [31]. His approach is based in a combination of Tabu Search and a population-based strategy that uses a recombination operator. The problem is similar to a highly constrained multi-person TSP where each sales-representative should revisit cities a given number of times. Costa presented results of this approach to

the 89-90, 91-92 and 93-94 seasons. For each season, more than one thousand games must be scheduled.

It is very difficult to find feasible schedules for sport tournament scheduling problems. The National Hockey League (NHL) game scheduling problem does not present an exception to general rule. In general, we should face a variety of very different types of constraints. We have selected this problem as a case-study on memetic algorithms since sometimes the existence of a large number of constraints seems to discourage some researchers to adopt such an approach. However, it is true that the recombination mechanism must be selected appropiately. Costa's approach exemplifies well how to handle these difficulties.

In this case, the NHL game scheduling problem includes teams located in North America (i.e. USA and Canada) and is divided into two *Conferences* (Western - Eastern) each of which is split into two *Divisions*. The number of teams involved is growing, it was 21 for the 89-90 season and it will be probably more than 28 by the end of the Century. For a season where 1066 games between 26 teams must be scheduled, each team would play 41 games at home (H) and 41 away (A). In the Pacific and Central divisions, which compose the Western Conference, the teams play other teams within their own division 6 times (3H, 3A). In the Northeast and Atlantic Divisions of the Eastern Conference, teams play other teams within their own division only 5 times. This means that some of them play (3H,2A) and others (2H,3A). All teams play 4 games (2H, 2A) against each of the teams in the other Division of their Conference and 2 games (1H, 1A) against each of the teams in the other Conference. The season has a duration which varies between 26 and 28 weeks.

Informally speaking, the NHL game scheduling problem is basically the problem of assigning each game a date while taking the following constraints into account:

1. Any team can not play more than one game a day.

2. Every team manager provides a set of 56 or slightly more prefered home dates from which 41 are to be chosen by the scheduler.

3. The total distance travelled by the teams should be minimized.

4. Any team should not play games on three straight days, and any team should not play more than three games in five consecutive days.

5. A team should play regurlarly while it is not at home. Breaks of more than three days should be avoided on the road.

6. Games can not be scheduled on certain days, either because of a major event (All Star Game break) or a holiday (Christmas Eve, Christmas) or the unavailability of a team (e.g. a special event in which the team is involved).

7. Some arenas can not be used on certain days because of other major activities (circus, music concerts, other sporting events, etc.)

8. There should be an *even* distribution of games through the season. Identical games (i.e. the same pair of teams at the same location) involving teams of the

same division should be scheduled at least 14 days apart. When teams come from different divisions the minimum allowed lapse of time between revisits is 30 days.

9. A team can not play more than 7 consecutive road games and the duration of a road trip can not exceed 14 days.

10. One idle day must be provided between games involving lengthy travel. Two games can not be scheduled on two consecutive days if one of the teams involved has to travel more than 900 miles. By convention, this applies only to teams coming from different divisions.

As we mentioned above, we would like to stress how a recombination operator can be designed for a problem with so many, rather restrictive, constraints. It was essential to the success of his implementation the identification of two diferent types of constraints, C_e and C_r. Constraints in class C_e are called *essential constraints* and those of class C_r are named *relaxed.* A schedule S is named *acceptable* if it satisfies all the constraints in $C_e \bigcup C_r$. A schedule S is named *feasible* if it satisfies at least all the essential constraints.

As it was seen from the list of constraints, there is no objective function aside from the requirement of the minimization of the total length travelled by all teams. Costa defined an *ad hoc* objective function m for a given feasible schedule S:

$$m(S) = \sum_{i \in C_r} w_i f_i(S) \tag{4}$$

where a set of weights $\{w_i\}$ indicates the relative importance given to each relaxed constraint. The $f_i(S)$ indicates the *degree of violation* of the i^{th} relaxed constraint of S. The problem is then to find the schedule S^* that minimizes the value of m over the set of feasible schedules.

Costa made some strategic decisions, that of including constraints 6 to 8 in C_e and to relax constraints 4 and 5. He also decided to *relax* constraint 1. He decided to split constraint 1 in three constraints:

- Any team can not play more than one game per day.
- Any team can not play more than two games per day.
- Any team can not play at home and on the road during the same day.

and in order to produce *feasible* schedules the first one is a relaxed constraint and the second two belong to C_e. Since one contains the others, possibly the denomination of *"matrioshka constraints"* would help to give a name to this algorithmic strategy to define a search space. Constraint 2 is probably impossible to be satisfied, so it is better to considered it as a relaxed constraint. Constraint 3 is obviously a relaxed constraint. Constraints 9 and 10 have been included in C_e. Then we have

- $f_1(S)$ indicates how many times a team plays *two games a day.*

- $f_2(S)$ indicates how many times the unavailability of an arena is not respected.
- $f_3(S)$ indicated the total distance (in hundreds of miles) traveled by the teams during the whole season.
- $f_4(S)$ indicated how many times a team plays more than two games on three straight days or more than three games on five straight days.
- $f_5(S)$ indicates how many breaks of more than three days occur while a team is on the road.

With this necessary definitions, we are now ready to discuss how Costa created a recombination mechanism which enables an effective search process. He comments that it was not an easy task to came up with a good mechanism, this is true in general. However, when such a mechanism is well designed it generally delivers good approximate solutions with a low computational complexity cost.

Given a schedule S he defined the *frame of a team t in schedule S* (noted $F_{S,t}$) as a binary string of length *ndays*. If $F_{S,t}[d] = 0$, this means that team t is *at home* on day d in the schedule S, otherwise the team is *on the road.* A convention is used: a team is at home between the last game on the road (respectively at home) and the next game at home (respectively on the road). In addition, a team is assumed to be at home before its first game and after its last game of the season. The *cost of a game g* in schedule S is a measure based on the degree of violation of relaxed constraints due to game g (which relates to the function m defined above). For a given visitor-home team pair, if the number of such games is greater than the number specified by the league, then all the games scheduled are named *redundant.*

Given two "parent" schedules S_a and S_b, we will suppose, without losing generality, that a *cooperation request* has been sent by S_a to S_b. A necessary, but not suficient, condition for S_b to accept the mating proposal is that S_a must be fitter than S_b (the relevance of introducing a mating proposal acceptance rule had been anticipated in [83] and [88]). Then a single schedule S_c^* is created putting together as many games as possible without modifying the frame of schedule S_b. This, of course, makes S_c^* unfeasible most of the times. Feasibility is regained by a greedy algorithm that sequentially removes the most costly redundant game in S_c^*.

This said, the relevant fact here is that Costa has found (after some previous attempts) that a good structure to be preserved in a highly optimized schedule (as it is one of the parents which came from a a Tabu Search period) is the *frame* of the schedule. This is not evident from the description of the set of constraints. In essence, it is an underlying structure, that in a fuzzy way characterizes a measure of similarity between two schedules. Its preservation, as opposed to strict preservation of particular dates of games, seems to be linked to an efficient implementation. The reason is that the frame of the schedule codes, in a certain way, information about the underlying basic structure of the schedule. We refer to Costa's paper for the other components of his approach and his computational results

6 Additional Issues

In this section we discuss three diverse issues related to the design and implementation of local search algorithms. In Section 6.1 we briefly discuss the behavior of local search for NP-complete problems. In Section 6.3 we review the available software tools that support the implementation of local search algorithms. In Section 6.4 we address the issue of parallelization.

6.1 Coping with NP-complete scheduling problems

How do computational complexity and NP-completeness relate with the empirical behaviour of local search-based metaheuristics ? Unfortunately, we are still far from a general theory and what we know does not amount much. However, they happen to probably be the method of choice to cope with hard computational problems. The last paragraph of Lewis and Papadimitriou's second edition of their introductory book to the theory of computation [76] clearly depicts the current situation:

> *"From the point of view of the formal criteria we have developed in this book, the local improvement algorithms and their many variants are totally unattractive: they do not in general return the optimum solution, they tend to have exponential worst-case complexity, and they are not even guaranteed to return solutions that are in any well-defined sense "close" to the optimum. Still, for many NP-complete problems, in practice they often turn out to be the ones that perform best ! Explaining and predicting the impressive empirical success of some of these algorithms is one of the most challenging frontiers of the theory of computation today."*

We have already mentioned that important contributions to establish worst-case complexity results for many scheduling problems were studied by Lenstra, Rinnooy Kan and Brucker [74, 73]. Some of the authors cited above had surveyed work in this area until 1993 in [72]. In addition, we recall that there is available information on the Internet which may help in the formulation of a given problem. Brucker and Knust maintain a set of WWW pages which contain useful information about the complexity results on scheduling problems [4]. Their pages include an interactive program (CLASS) that classifies scheduling problems according to the $\alpha|\beta|\gamma$ notation first used by Graham *et al.* [56]. They also have a computer program (MSPCLASS) which based in the $\alpha|\beta|\gamma$-classification scheme helps to retrieve several results about their computational complexity.

Several instances of timetabling problems arising in practice have been proven NP-Complete by Cooper and Kingston in [**?**]. NP-Completeness of a problem is generally viewed as "bad news" since it is conjectured that probably no efficient (i.e. polynomial) algorithm exists for the general case. However, Cooper and Kingston gave some hope to practitioners when they say: *"Against these negative results we can set the limited size of timetable construction instances. High schools with more*

[4] See `http://scarlett.mathematik.uni-osnabrueck.de/research/OR/class`

than 100 teachers are rare; a week of more than 40 times is also rare. University problems are larger but seem to be easier."

In contrast, other timetabling problems arising in practice can be larger than the challenges put forward by high schools. They can also provide potentially more economic rewards. Brusco, Jacobs, Bongiorno, Lyons, and Tang, presented a case study arising from United Airlines' tour scheduling problems which seek to improve the abilities of their personnel. Their work deals with the creation of two modules designed to enhance the tour-scheduling process associated with their planning system. One is a column generation approach which improves the generation of employee shifts. The second module is a local search heuristic based on Simulated Annealing. They report potential annual cost savings of more than 8 millon US dollars [14]. Brusco and Jacobs, in another paper, discuss the cost-analysis of different formulations of personnel scheduling problems [13].

6.2 PLS-Completeness

An interesting characterization of problems in computational complexity classes related with local search algorithms has been established. Using the same mathematical notation used in the introduction, we can present it as follows:

Definition: A *local search problem* $(P/\mathcal{N})$ is in class PLS if there are three polynomial-time algorithms $a1_P$, $a2_P$, and $a3_P$ with the following properties:

- Given an input x (formally a string $x \in \{0,\ 1\}^*$), algorithm $a1_P$ determines whether $x \in I_P$ and in this case produces some solution $y_0 \in sol_P(x)$.
- Given an instance $x \in I_P$ and an input y, algorithm $a2_P$ determines whether $y \in sol_P(x)$ and computes $m_P(y,\ x)$.
- Given an instance $x \in I_P$ and a solution $y \in sol_P(x)$, algorithm $a3_P$ determines whether is a local optimum, and if it is not, $a3_P$ outputs a neighbour $y' \in \mathcal{N}_P(y,\ x)$ with (strictly) better value of m_P, i.e. $m_P(y',\ x) < m_P(y,\ x)$ for a minimization problem, and $m_P(y',\ x) > m_P(y,\ x)$ for a maximization problem.

Note that the definition above also includes the problems of local optimality arising in linear programming, maximum matching, minimum spanning tree, and other problems for which efficient polynomial-time algorithms exist. It is obvious that the definition embodies a hill-climbing procedure by iterating in the application of algorithm a_3.

It is very unlikely that the class PLS contains NP-hard problems. It has been proven that if a PLS problem is NP-hard, then NP=co-NP. The complexity class co-NP is the class of decision problems whose complement is in NP. The Hamiltonian Cycle problem is in NP. Its complement would be the decision problem given by the question: 'Is a given graph non-Hamiltonian' ? which belongs to co-NP.

6.3 Local Search Software Tools

Local search is a general paradigm for the solution of a large number of problems. For this reason, some research efforts have been devoted to the design of generic tools for the solution of scheduling problems based on local search. Unfortunately, though, up to now not many systems have been developed in the research community that can be considered as general purpose tools.

One on such systems is LOCALIZER, a modeling language developed by Michel and Van Hentenrick [79]. LOCALIZER allows the user to describe in a completely declarative way a local search problem and the corresponding algorithm. The specification of the problem is divided in a number of *sections* corrsponding to the various features; among the others, there are the search space (section **Variable**), the neighborhood relation and the move selection (section **Neighborhood**), the cost function (section **Objective Function**), and the initial solution (section **Start**). Starting from the specification, the system performs the corresponding local search run and deliver the output.

Fleurent and Ferland [44] and Schaerf *et al* [104] provide an alternative approach based on the use of object-oriented *frameworks* as general tools for local search algorithms in C++. For example, the framework proposed in [104] comprises a hierarchy of abstract classes, one for each technique taken into account. Each class specifies and implements the invariant part of the algorithm built according to the technique, and is meant to be specialized by a concrete class once a given local search search problem is considered, so as to implement the problem-dependent part of the algorithm.

The main difference of the latter approach with respect to systems *a la* LOCALIZER is that such systems are *black boxes*, in the sense that the program corresponding to the specified algorithm is assembled internally by the system and therefore its execution is performed "behind the scenes". Conversely, the framework approach is completely *glass box*, and the exact C++ statements of the program are "before user's eyes". Furthermore, the LOCALIZER user must learn the syntax of the specification language. In the latter approach instead the algorithm is written in a language (i.e., C++) that a skilled user might already know, or at least be well-disposed to learn. Finally, a framework is fully extensible and customizable by the expert user by means of new derived classes. In contrast, the modification of the LOCALIZER system would require the intervention of the system designers.

6.4 Parallel Local Search

Local search techniques can be parallelized in many different ways. For example, given that multiple runs of the same technique are often necessary, an effective way to exploit parallel computational power, is to have multiple executions of the algorithm on different machines starting from different initial solutions.

In particular, if the initial solution is generated at random, a different starting point is generated by each independent run, and thus this form of parallelization requires no programming effort and results in no computational overhead.

The above mentioned idea is the simplest and most natural way to include parallelism in a local search technique. However, many other parallel schemes have been proposed in the literature. For example, for techniques that explore the full neighborhood at each iteration, the exploration could be done in parallel by splitting the neighborhood among different processors each of them exploring a portion of it and returning the best move upon such portion. A central controller would then choice the overall best among them.

Verhoeven and Aarts [117] provide a comprehensive discussion of parallel local search techniques and provide an exhaustive classification based on a set of general features of the algorithm. For example, they partition the approaches in *single-walk*, when all processors contribute to form the single trajectory, and *multiple-walk*, when they work on different ones. For the sake of brevity, we do not repeat the classification here and we refer to the cited paper for it, and for the relevent literature. Another classification, specialized only on Tabu Search, is provided by Crainic *et al* in [33].

7 Conclusions

Aknowledgements

Pablo Moscato wants to aknowledge Prof. Paulo Morelato França for useful suggestions which helped to improve a preliminary version of this tutorial. He also wants to thank Prof. Marcos Nereu Arenales for letting him use computing facilities at his lab. He also wants to acknowlege finantial support by FAPESP, Brazil.

This work has been carried out by Andrea Schaerf as part of the project *Innovative Control Strategies for Artificial Intelligence Systems* (SCI*SIA) financed by the Italian Research Council (CNR).

(* Citare l'ECAI-98 tutorial *)

References

[1] E. H. L. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, New York, 1989.

[2] D. Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, 37(1):98–113, 1991.

[3] A. Agnetis, A. Alfieri, P. Brandimarte, and P. Prinsecchi. Joint job/tool scheduling in a flexible manufacturing cell with no on-board tool magazine. *Computer Integrated Manufacturing Systems*, 10(1):61–68, Feb. 1997.

[4] J.-M. Alliot, editor. *Artificial evolution: European conference, AE 95, Brest, France, September 4–6, 1995: selected papers*, volume 1063 of *Lecture Notes in Computer Science*, New York, NY, USA, 1996. Springer-Verlag Inc.

[5] R. Battiti. Reactive search: Toward self-tuning heuristics. In V. J. Rayward-Smith, editor, *Modern Heuristic Search Methods*, pages 61–83. John Wiley and Sons Ltd, 1996.

[6] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal of Computing*, 6(2):126–140, 1994.

[7] J. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4:387–411, 1992.

[8] M. Bertocchi and C. D. Odoardo. A stochastic algorithm for global optimization based on threshold accepting technique. *Collection: Optimization 1&2, Singapore*, pages 141–146, 1992.

[9] J. Blazewicz, M. Drozdowski, D. deWerra, and J. Weglarz. Deadline scheduling of multiprocessor tasks. *Discrete Applied Mathematics*, 65(1-3):81–95, 1996.

[10] A. Bouju, J. Boyce, C. Dimitropoulos, G. vom Scheidt, and J. Taylor. Tabu search for the radio links frequency assignment problem. In *Proc. of Applied Decision Technologies (ADT-95)*, 1995.

[11] J. Bowen and G. Dozier. Solving constraint satisfaction problems using A genetic/systematic search hybrid that realizes when to quit. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 122–129, San Francisco, July15–19 1995. Morgan kaufmann Publishers.

[12] M. Brusco and L. Jacobs. A simulated annealing approach to the solution of flexible labour scheduling problems. *OR; the journal of the Operational Research Soc*, 44(12):1191, 1993.

[13] M. Brusco and L. Jacobs. Cost-analysis of alternative formulations for personnel scheduling in continuously operating organizations. *EUROPEAN JOURNAL OF OPERATIONAL RESEARCH*, 86(2):249–261, Oct. 19, 1995.

[14] M. Brusco, L. Jacobs, R. Bongiorno, D. Lyons, and B. Tang. Improving personnel scheduling at airline stations. *Operations Research*, 43(5):741–751, Sep.-Oct. 1995.

[15] M. Brusco, G. Thompson, and L. Jacobs. A morph-based simulated annealing heuristic for a modified bin-packing problem. *JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY*, 48(4):433–439, Apr. 1997.

[16] E. Burke, J. Newall, and R. Weare. A memetic algorithm for university exam timetabling. In E. Burke and P. Ross, editors, *The Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[17] E. Burke and A. Smith. A memetic algorithm for the maintenance scheduling problem. In *Proceedings of the ICONIP/ANZIIS/ANNES '97 Conference, Dunedin, New Zealand*, pages 469–472. Springer, 24-28 November 1997.

[18] E. K. Burke, D. G. Elliman, and R. F. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. In *6th International Conference on Genetic Algorithms (ICGA'95, Pittsburgh, USA, 15th-19th July 1995)*, pages 605–610. Morgan Kaufmann, San Francisco, CA, USA, 1995.

[19] F. Carmusciano and D. D. L. Cardillo. A simulated annealing with tabu list algorithm for the school timetable problem. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 231–243, 1995.

[20] D. Castelino, S. Hurley, and N. Stephens. A tabu search algorithm for frequency assignment. *Annals of Operations Research*, 63:301–319, 1996.

[21] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[22] H. W. Chan and J. Sheung. Roster scheduling at an air cargo terminal: a tabu search approach. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 409–422, 1995.

[23] R. Cheng and M. Gen. Parallel machine scheduling problems using memetic algorithms. *COMPUTERS & INDUSTRIAL ENGINEERING*, 33(3–4):761–764, Dec. 1997.

[24] W. Chiang and R. Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63:3–27, 1996.

[25] B. Codenotti, G. Manzini, L. Margara, and G. Resta. Perturbation: an efficient technique for the solution of very large instances of the euclidean tsp. *INFORMS Journal on Computing*, 8:125–133, 1996.

[26] N. Collins, R. Eglease, and B. Golden. Simulated annealing - an annotated bibliography. *American journal of mathematical and management*, 8(3/4):209, 1988.

[27] A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for high school timetabling. *Computational Optimization and Applications*, 9(3):275–298, 1998.

[28] R. C. Conta, A. Ferreira, and P. Rebreyend. Integrating list heuristics into genetic algorithms for multiprocessor scheduling. In *Eighth IEEE Symposium on Parallel and Distributed Processing (SPDP'96)*, pages 462–469, Washington - Brussels - Tokyo, Oct. 1996. IEEE Computer Society.

[29] D. Costa. On the use of some known methods for t-colorings of graphs. *Annals of Operations Research*, 41:343–358, 1993.

[30] D. Costa. A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research*, 76:98–110, 1994.

[31] D. Costa. An evolutionary Tabu Search algorithm and the NHL scheduling problem. *INFOR*, 33(3):161–178, 1995.

[32] D. Costa, A. Hertz, and O. Dubuis. Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs. ORWP 94-09, Ecole Polytechnique Fédérale de Lausanne, November 1994.

[33] T. G. Crainic, M. Toulouse, and M. Gendreau. Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal of Computing*, 9(1):61–72, 1997.

[34] H. Crauwels, C. Potts, and L. VanWassenhove. Local search heuristics for single machine scheduling with batch set-up times to minimize total weighted completion time. *ANNALS OF OPERATIONS RESEARCH*, 70:261–279, 1997.

[35] H. Crauwels, C. Potts, and L. VanWassenhove. Local search heuristics for single-machine scheduling with batching to minimize the number of late jobs. *European Journal of Operational Research*, 90(2):200–213, Apr. 19, 1996.

[36] J. Dorn, M. Girsch, G. Skele, and W. Slany. Comparison of iterative improvement techniques for schedule optimization. *European Journal of Operational Research*, 94(2):349–361, Oct. 25, 1996.

[37] J. Dorn, R. Kerr, and G. Thalhammer. Reactive scheduling - improving the robustness of schedules and restricting the effects of shop-floor disturbances by fuzzy-reasoning. *International Journal of Human-Computer Studies*, 42(6):687–704, Jun. 1995.

[38] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys. 90, No.1, 161-175*, 1990.

[39] M. Duque-Antón, D. Kunz, and B. Rüber. Channel assignment for cellular radio using simulated annealing. *IEEE Transaction on Vehicular Technology*, 42(1):14–21, 1993.

[40] J. R. *et al.* Easily searched encodings for number partitioning. Technical Report TR-10-94, Harvard University, Cambridge, MA, USA, 1994. available via ftp://das-ftp.harvard.edu/techreports/tr-10-94r.ps.gz, to appear in *Journal of Optimization Theory and Applications*, 1995.

[41] S. Even, A. Itai, and A. Shamir. On the complexity of timetabling and multicommodity flow problems. *SIAM Journal of Computation*, 5(4):691–703, 1976.

[42] B. Fleischmann and H. Meyr. The general lotsizing and scheduling problem. *OR SPEKTRUM*, 19(1):11–21, Jan. 1997.

[43] C. Fleurent and J. Ferland. Hybrid genetic algorithms for combinatory optimization. *RAIRO – Recherche Operationnelle – Operations Research*, 30(4):373–398, 1996.

[44] 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, editors, *Cliques, Coloring, and Satisfiability. Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 619–652. American Mathematical Society, 1996.

[45] M. Flood. The traveling-salesman problem. *Oper. Res.*, 4:61–75, 1956.

[46] P. França, M. Gendreau, G. Laporte, and F. Muller. A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 43(2-3):79–89, Jun. 1, 1996.

[47] J. Frank. Weighting for Godot: Learning heuristics for GSAT. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, pages 338–343, Portland, USA, 1996. AAAI Press/MIT Press.

[48] M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness.* W.H. Freeman and Company, San Francisco, 1979.

[49] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

[50] C. A. Glass and C. N. Potts. A comparison of local search methods for flow shop scheduling. *Annals of Operations Research*, 63:489–509, 1996.

[51] F. Glover. Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Applied Mathematics*, 49(1-3):231–255, Mar. 30, 1994.

[52] F. Glover and M. Laguna. *Tabu search.* Kluwer Academic Publishers, 1997.

[53] F. Glover, M. Parker, and J. Ryan. Coloring by tabu branch and bound. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability. Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1996.

[54] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.

[55] J. L. Gonzalez Velardez. Capitulo 4: GRASP. In B. A. Diaz, editor, *Las Nuevas Tecnicas Heuristicas y las Redes Neuronales*. Ed. Paraninfo, Madrid, 1996.

[56] R. Graham, E. Lawler, J. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.*, 4:287–326, 1979.

[57] K. Haase. Capacitated lot-sizing with sequence dependent setup costs. *OR Spektrum*, 18(1):51–59, Jan. 1996.

[58] A. Hertz. Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 54:39–47, 1991.

[59] R. Hofmann. *Examinations of the Algebra of Genetic Algorithms*. PhD thesis, Technische Universität München (Institut für Informatik), November 1993. (Diplomarbeit).

[60] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, Feb. 1994.

[61] H. Ishibuchi, N. Yamamoto, S. Misaki, and H. Tanaka. Local search algorithms for flow-shop scheduling with fuzzy due-dates. *International Journal of Production Economics*, 33(1-3):53–66, Jan. 1994.

[62] M. R. J. Mittenthal and A. Rana. A hybrid simulated annealing approach for single machine scheduling problems with non-regular penalty functions. *Computers and Operations Research*, 20(2):103, 1993.

[63] D. Jeffcoat and R. Bulfin. Simulated annealing for resource-constrained scheduling. *European Journal of Operational Research*, 70(1):43, 1993.

[64] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37:79–100, 1988.

[65] M. Jünger and S. Thienel. Introduction to ABACUS - A Branch-And-CUt System. ALCOM-IT Technical Report TR-143-97, Cologne, 1997.

[66] N. Kadaba and K. E. Nygard. Improving the performance of genetic algorithms in automated discovery of parameters. Technical Report NDSU-CS-TR-90-28, North Dakota State University. Computer Science and Operations Research, 1990.

[67] H. Kim, Y. Hayashi, and K. Nara. An algorithm for thermal unit maintenance scheduling through combined use of ga, sa and ts. *IEEE Transactions on Power Systems*, 12(1):329–335, Feb. 1997.

[68] S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[69] C. Koulamas. Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers & Operations Research*, 23(10):945–956, Oct. 1996.

[70] C. Koulamas, K. Davis, and F. Turner III. A survey of simulated annealing applications to operations research problems. *Omega*, 22(1):41, 1994.

[71] N. Krasnogor, P. Moscato, and M. Norman. A new hybrid heuristic for large geometric traveling salesman problems based on the delaunay triangulation. In *Anais do XXVII Simposio Brasileiro de Pesquisa Operacional, Vitoria, Brazil*, SOBRAPO, 6-8 Nov. 1995.

[72] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys. Sequencing and scheduling: algorithms and complexity. In S. Graves, A. R. Kan, and P. Zipkin, editors, *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. North Holland, Amsterdam, 1993.

[73] J. Lenstra and A. R. Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[74] J. Lenstra, A. R. Kan, and P. Brucker. Complexity of machine scheduling problems. In P. Hammer, E. Johnson, B. Korte, and G. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. North Holland, Amsterdam, 1977.

[75] D. Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers & Operations Research*, 23(6):547–558, Jun. 1996.

[76] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1998.

[77] F. Li, R. Morgan, and D. Williams. Hybrid genetic approaches to ramping rate constrained dynamic economic dispatch. *Electric Power Systems Research*, 43(2):97–103, Nov. 1997.

[78] C. Lin, K. Haley, and C. Sparks. A comparative-study of both standard and adaptive versions of threshold accepting and simulated annealing algorithms in 3 scheduling problems. *European Journal of Operational Research*, 83(2):330–346, Jun. 8, 1995.

[79] L. Michel and P. Van Hentenrick. Localizer: A modeling language for local search. In *Proc. of the 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP-97)*, number 1330 in Lecture Notes in Computer Science, pages 238–252, Schloss Hagenberg, Austria, 1997.

[80] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[81] P. Morris. The breakout method for escaping from local minima. In *Proc. of the 11th Nat. Conf. on Artificial Intelligence (AAAI-93)*, pages 40–45. AAAI Press/MIT Press, 1993.

[82] R. J. Morris and W. S. Wong. Systematic choice of initial points in local search: Extensions and application to neural networks. *Inf. Process. Lett.*, 39:213–217, 1991.

[83] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. C3P Report 826, California Institute of Technology, Pasadena, CA 91125, 1989.

[84] P. Moscato. An introduction to population approaches for optimization and hierarchical objective functions: The role of tabu search. *Annals of Operations Research*, 41(1-4):85–121, 1993.

[85] P. Moscato and B. A. Diaz. Recocido simulado. In B. A. Diaz, editor, *Optimización heurística y redes neuronales*, pages 37–66. Editorial Paraninfo, Madrid, España, 1996.

[86] P. Moscato and J. Fontanari. Stochastic vs. deterministic update in simulated annealing. *Phys. Lett. A*, 146:204–208, 1990.

[87] P. Moscato and M. Norman. A "memetic" approach for the traveling salesman problem. Implementation of a computational ecology for combinatorial optimization on message-passing systems. In *Proceedings of the International Conference on Parallel Computing and Transputer Applications*, pages 177–186, Amsterdam, 1992. IOS Press.

[88] P. Moscato and M. Norman. A memetic approach for the traveling salesman problem. implementation of a computational ecology for combinatorial optimization on message-passing systems. In M. V. *et al.*, editor, *Parallel Computing and Transputer Applications. Part 1*, Amsterdam, 1992. IOS Press.

[89] P. Moscato and F. Tinetti. Blending heuristics with a population-based approach: A memetic algorithm for the traveling salesman problem. Report 92-12, Universidad Nacional de La Plata, C.C. 75, 1900 La Plata, Argentina, 1992.

[90] T. Murata, H. Ishibuchi, and H. Tanaka. Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering*, 30(4):1061–1071, Sep. 1996.

[91] K. J. Nurmela. Constructing combinatorial designs by local search. Research Report A27, Helsinki University of Technology, Nov. 1993.

[92] R. Padman and S. F. Roehrig. A genetic programming approach for heuristic selection in constrained project scheduling. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Science and Operations*

Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies, chapter 18, pages 405–421. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[93] B. Paechter, A. Cumming, M. Norman, and H. Luchian. Extensions to a Memetic timetabling system. In E. Burke and P. Ross, editors, *The Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 251–265. Springer Verlag, 1996.

[94] M. Park and Y. Kim. Search heuristics for a parallel machine scheduling problem with ready times and due dates. *Computers & Industrial Engineering*, 33(3-4):793–796, Dec. 1997.

[95] S. Porto and C. Ribeiro. A tabu search approach to task-scheduling on heterogeneous processors under precedence constraints. *International Journal of high speed computing*, 7(1):45–71, Mar. 1995.

[96] J. Psarras, E. Stefanitsis, and N. Christodoulou. Combination of local search and clp in the vehicle-fleet scheduling problem. *European Journal of Operational Research*, 98(3):512–521, May. 1, 1997.

[97] N. Radcliffe. Fitness variance of formae and performance prediction. Report TR-94-17, The University of Edinburgh, 1994. submitted to *Foundations of Genetic Algorithms*.

[98] N. Radcliffe and P. Surry. Formal memetic algorithms. *AISB94*, 1994.

[99] S. Rana, A. E. Howe, L. D. Whitley, and K. Mathias. Comparing heuristic, evolutionary and local search approaches to scheduling. In B. Drabble, editor, *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 174–181. AAAI Press, 1996.

[100] J. Ruml. Stochastic approximation algorithms for number partitioning. Technical Report TR-17-93, Harvard University, Cambridge, MA, USA, 1993. Available via `ftp://das-ftp.harvard.edu/techreports/tr-17-93.ps.gz`.

[101] A. Schaerf. A survey of automated timetabling. Technical Report CS-R9567, CWI, Amsterdam, The Netherlands, 1995. To appear in *Artificial Intelligence Review*.

[102] A. Schaerf. Tabu search techniques for large high-school timetabling problems. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, pages 363–368, Portland, USA, 1996. AAAI Press/MIT Press.

[103] A. Schaerf. Combining local search and look-ahead for scheduling and constraint satisfaction problems. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI-97)*, pages 1254–1259, Nagoya, Japan, 1997. Morgan-Kaufmann.

[104] A. Schaerf, M. Lenzerini, and M. Cadoli. A C++ framework for combinatorial search problems, 1998. Submitted for publication.

[105] B. Selman and H. A. Kautz. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, pages 290–295. Morgan Kaufmann, 1993.

[106] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.

[107] G. Solotorevsky, E. Gudes, and A. Meisels. RAPS: A rule-based language specifying resource allocation and time-tabling problems. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):681–697, 1994.

[108] Y. Sotskov, T. Tautenhahn, and F. Werner. Heuristics for permutation flow shop scheduling with batch setup times. *OR Spektrum*, 18(2):67–80, Apr. 1996.

[109] J. Sridhar and C. Rajendran. Scheduling in a cellular manufacturing system: a simulated annealing approach. *International journal of production research*, 31(12):2927, 1993.

[110] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:433–445, 1991.

[111] D. Teodorovic, E. Krcmar-Nozic, and G. Stojkovic. Airline seat inventory control: An application of the simulated annealing method. *Transportation planning and technology*, 17(3):219, 1993.

[112] E. Tsang and C. Voudouris. Fast local search and guided local search and their application to british telecom's workforce scheduling. Technical Report CSM-246, Department of Computer Science, University of Essex, Colchester, UK, 1995.

[113] R. Vaessens, E. Aarts, and J. K. Lenstra. A local search template. Technical Report COSOR 92-11 (revised version), Eindhoven University of Technology, Eindhoven, NL, 1995.

[114] C. L. Valenzuela and L. P. Williams. Improving heuristic algorithms for the Travelling Salesman Problem by using a genetic algorithm to perturb the cities. In T. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 458–464, San Francisco, July19–23 1997. Morgan Kaufmann.

[115] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Kluwer Academic Publishers Group, 1987.

[116] J. M. Varanelli and J. P. Cohoon. Population-oriented simulated annealing: A genetic/thermodynamic hybrid approach to optimization. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 174–183, San Francisco, July15–19 1995. Morgan kaufmann Publishers.

[117] M. G. A. Verhoeven and E. H. L. Aarts. Parallel local search. *Journal of Heuristics*, 1:43–65, 1995.

[118] F. Werner. On the heuristic solution of the permutation flow-shop problem by path algorithms. *Computers & Operations Research*, 20(7):707–722, Sep. 1993.

[119] R. Wills and B. Terrill. Scheduling the australian state cricket season using simulated annealing. *Journal of the Operational Research Society*, 45(3):276, 1994.

[120] K. Wong and Y. Wong. Short-term hydrothermal scheduling. Part II: Parallel simulated annealing approach. *IEE Proceedings. C*, 141(5):502, 1994.

[121] W. S. Wong and R. J. Morris. A new approach to choosing initial points in local search. *Inf. Process. Lett.*, 30:67–72, 1989.

[122] D. Woodruff and E. Zemel. Hashing vectors for tabu search. *Annals of Operations Research*, 41:123–137, 1993.

[123] M. Yoshikawa, K. Kaneko, T. Yamanouchi, and M. Watanabe. A constraint-based high school scheduling system. *IEEE Expert*, 11(1):63–72, 1996.

[124] N. Yugami, Y. Ohta, and H. Hara. Improving repair-based constraint satisfaction methods by value propagation. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pages 344–349, 1994.

[125] J. Zhang and H. Zhang. Combining local search and backtracking techniques for constraint satisfaction. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, pages 369–374, Portland, USA, 1996. AAAI Press/MIT Press.