

# Neighborhood Portfolio Approach for Local Search applied to Timetabling Problems\*

Luca Di Gaspero and Andrea Schaerf  
l.digaspero@uniud.it, schaerf@uniud.it  
Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica  
Università di Udine

January, 8 2006

## Abstract

A recent trend in local search concerns the exploitation of several different neighborhoods so as to increase the ability of the algorithm to navigate the search space.

In this work we investigate a hybridization technique, that we call *Neighborhood Portfolio Approach*, that consists in the interleave of local search techniques based on various combinations of neighborhoods. In particular, we are able to select the most effective search technique through a systematic analysis of all meaningful combinations built upon a set of basic neighborhoods.

The proposed approach is applied to two practical problems belonging to the timetabling family, and systematically tested and compared on real-world instances. The experimental analysis shows that our approach leads to automatic design of new algorithms that provide better results than basic local search techniques.

## 1 Introduction

Local search [1, 8] is a successful meta-heuristic paradigm for the solution of optimization and constraint satisfaction problems that have proven to be very effective in a large number of problems.

One of the most critical features of local search is the definition of the neighborhood. In fact, for most popular problems, many different neighborhoods have been considered and experimented. For example, at least ten different neighborhoods have appeared in the literature for the JOB-SHOP SCHEDULING problem (see [18] for a review).

We believe that the exploitation of different neighborhoods, and/or different meta-heuristics, within a hybrid local search strategy, is one of the key ideas to improve the results of local search algorithms. One of the main motivations is related to the diversification of search needed to escape from local minima. In fact a state that is a local minimum for a given neighborhood, is not necessarily a local minimum for another one. Therefore, an algorithm that uses both has more chances to move toward better solutions. More generally, the use of different neighborhoods may lead to unexplored trajectories that make the overall search more effective.

Several researchers have already proposed, applied, and evaluated algorithms based on combination of neighborhoods (see, e.g., [6]). For example, the well-known *Variable Neighborhood Search* technique proposed by Hansen and Mladenović [7] is based on the idea of using a set of neighborhoods of increasing size.

However, up to our knowledge, little effort has been devoted to a comprehensive investigation of the automatic design of local search algorithms based on the systematic exploitation of a portfolio of neighborhoods.

In this work we attempted to accomplish this task and we formally define and apply principled ways to combine different neighborhoods and algorithms. In details, we define a set of simple operators that

---

\*This paper appeared in Journal of Mathematical Modelling and Algorithms, Volume 5, Number 1, April 2006. DOI: 10.1007/s10852-005-9032-z. The original publication will be available at [www.springerlink.com](http://www.springerlink.com)

compound basic neighborhoods, and a solving strategy that combines several algorithms, possibly based on different neighborhoods. We name this idea *Neighborhood Portfolio* approach.

We provide a formal description of the Neighborhood Portfolio and we describe its use in the solution of two different problems belonging to the timetabling family, namely COURSE TIMETABLING and EVENT TIMETABLING. For each of them, we provide a systematic experimental analysis that highlights the advantages of the Neighborhood Portfolio idea.

## 2 Local Search basics

Given an instance  $\pi$  of a (constraint satisfaction or optimization) problem  $\Pi$ , we associate a *search space*  $S$  with it. Each element  $s \in S$  corresponds to a potential solution of  $\pi$ , and is called a *state* of  $\pi$ . Local search relies on a function  $\mathcal{N} : S \rightarrow \mathcal{P}(S)$ <sup>1</sup>, which assigns to each  $s \in S$  its *neighborhood*  $\mathcal{N}(s) \subseteq S$ . Each state  $s' \in \mathcal{N}(s)$  is called a *neighbor* of  $s$ . A neighbor of a state is usually described in terms of the atomic change (called *move*) that must be applied upon the state to obtain the neighbor.

A local search algorithm starts from an initial state  $s_0$  (which can be obtained with some other technique or generated randomly) and enters a loop that *navigates* the search space, stepping from a state  $s_i$  to one of its neighbors  $s_{i+1}$ .

Several search control strategies (also called meta-heuristics) can be defined upon this framework, according to the criteria used for the selection of the move and for stopping the search. However, in all techniques, the search is driven by a *cost function*  $f$  that estimates the quality of each state and, without loss of generality, it has to be minimized. For constraint satisfaction problems,  $f$  generally accounts for the number of violated constraints, whereas for optimization problems it takes into account also the objective function of the problem.

Two of the most common local search techniques are *hill climbing* (HC) and *tabu search* (TS). These techniques have many variants, and in the following we briefly describe the version that we use in this work. The reader who is interested in a more detailed presentation can refer, for example, to [19] and [6]. It is worth to mention that another prominent local search technique not considered in this work is Simulated Annealing [10].

HC is not a single local search technique, but rather a family of techniques based on the idea of performing only moves that improve (or leave unchanged, i.e. *sideways* moves) the value of the cost function  $f$ . Among HC variants, we make use of the so-called RNA (*Randomized Non Ascending*) that selects a random move at each iteration, and performs it if improving or sideways, and leaves the state unchanged otherwise.

This HC strategy does not stop when it reaches a local minimum. In fact, the search might loop infinitely by cycling among two or more neighbor local minima at equal cost. To provide against this situation, the stop criterion is based on the number of iterations elapsed from the last strict improvement. Specifically, given a fixed value  $n$  the algorithm stops after  $n$  iterations that do not improve the value of the cost function.

TS is a local search strategy based on memory. At each state  $s_i$ , TS explores all members of the current neighborhood  $\mathcal{N}(s_i)$ . Among the elements in  $\mathcal{N}(s_i)$ , the one that gives the minimum value of the cost function becomes the new current state  $s_{i+1}$ , independently of the fact whether  $f(s_{i+1})$  is less or greater than  $f(s_i)$ .

In order to prevent cycling, the so-called *tabu list* is used, which determines forbidden moves. This list stores the most recently accepted moves and the *inverses* of the moves in the list are forbidden (i.e., the moves that are leading again toward the just visited local minimum).

We make use of a *variable length* tabu list: we assign to each move that enters the list a random number that represents the number of iterations for which the move is kept in the tabu list. The random number considered ranges between  $k_{min}$  to  $k_{max}$ , where  $k_{min}$  and  $k_{max}$  are parameters of the method. A move is removed from the list when its tabu period is expired.

There is also a mechanism, called *aspiration*, that overrides the tabu status: if a move leads to a state whose cost function value is better than the best value found so far, then its tabu status is dropped and the resulting state is acceptable as the new current one.

Like HC, the stop criterion is based on the so-called *idle iterations*: the search terminates when a given number of iterations has elapsed from the last (strict) improvement of the cost function.

---

<sup>1</sup> $\mathcal{P}(S)$  denotes the powerset of  $S$ , i.e., the set of subsets of  $S$

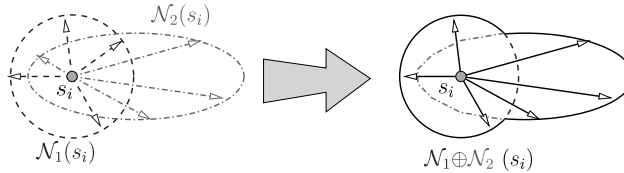


Figure 1: An example of the neighborhood union operator  $\oplus : \mathcal{F}_S \times \mathcal{F}_S \rightarrow \mathcal{F}_S$  for the neighborhood  $\mathcal{N}_1 \oplus \mathcal{N}_2$ .

### 3 Neighborhood Operators

Consider a set  $k$  of neighborhoods  $\mathcal{N}_1, \dots, \mathcal{N}_k$  defined on  $S$ , for a given problem  $\Pi$ ; we denote with  $\mathcal{F}_S = \{F|F : S \rightarrow \mathcal{P}(S)\}$  the functional space of all candidate neighborhood functions for the search space  $S$ , so that  $\mathcal{N}_i \in \mathcal{F}_S$  for each  $i = 1, \dots, k$ . The functions  $\mathcal{N}_i$  are obviously problem-dependent, and are defined by the person who investigates  $\Pi$ . Though there are in principle many ways to combine different neighborhoods, in this work we formally define and investigate the following set of operators:

**Neighborhood union:** we consider the union of many neighborhoods in the set-theoretical sense. The local search algorithms based on this neighborhood select, at each iteration, a move belonging to any of the components.

**Neighborhood sequence:** the neighborhood is composed of chains of moves, each move belonging to a different neighborhood. The type of the moves at each step in the chain is fixed, and is based on the order of the neighborhoods appearing in the sequence.

**Neighborhood composition:** is a more general case of neighborhood union and sequence. As with the sequence operator, the atomic moves are chains of moves. However, in this case, the moves in the chain can belong to any of the neighborhoods employed, regardless of the order of sequence.

In order to define these operators we have to prescribe how the elements of the compound neighborhood are generated, starting from the elements of the basic neighborhoods.

#### 3.1 Neighborhood Union

We start presenting the most straightforward combination operator, i.e., the neighborhood union. A pictorial representation of this operator is reported in Figure 1, whereas its definition is as follows.

**Definition 3.1** (Neighborhood Union). *Given  $k$  neighborhood functions  $\mathcal{N}_1, \dots, \mathcal{N}_k$ , we call neighborhood union, written  $\mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k$ , the neighborhood function such that, for each state  $s$ , the set  $\mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k(s)$  is equal to  $\mathcal{N}_1(s) \cup \dots \cup \mathcal{N}_k(s)$ . A member of this neighborhood is simply a move  $m \in \mathcal{N}_i$  for a suitable index  $i$ .*

*A random element in the neighborhood union  $\mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k$  is selected at random from the set union of all the neighborhoods  $\mathcal{N}_i$ ,  $i = 1, \dots, k$ . The thorough exploration of the  $\mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k$  is performed by exhaustively exploring each  $\mathcal{N}_i$  and selecting the overall best solution.*

*The definition of inverse of a move, related to the prohibition mechanism of TS, is the following. Given a move  $m \in \mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k$ , we have that  $m \in \mathcal{N}_i$  for a suitable index  $i$ . If we denote with  $m_i^{-1}$  the inverse of  $m$  in  $\mathcal{N}_i$ , we consider the move  $m_i^{-1}$  as inverse also in the compound neighborhood  $\mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k$ .*

Notice that, in this case, the order of the neighborhoods in the combination is not relevant and it is not worth to repeat the same neighborhood more than once, since the set union is an associative, commutative and idempotent operator.

Furthermore, according to this definition, there are several possible choices for the selection of a random move in the neighborhood union. In fact, the simplest random distribution for selecting a move in  $\mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k$  from a state  $s$  first selects uniformly a random  $i$  (with  $1 \leq i \leq k$ ), and then selects a random state  $s' \in \mathcal{N}_i(s)$  as to the random distribution associated with  $\mathcal{N}_i$ . In this case, the selection in the neighborhood union is not uniform, because it is not weighted based on the cardinality of the sets  $\mathcal{N}_i(s)$ .

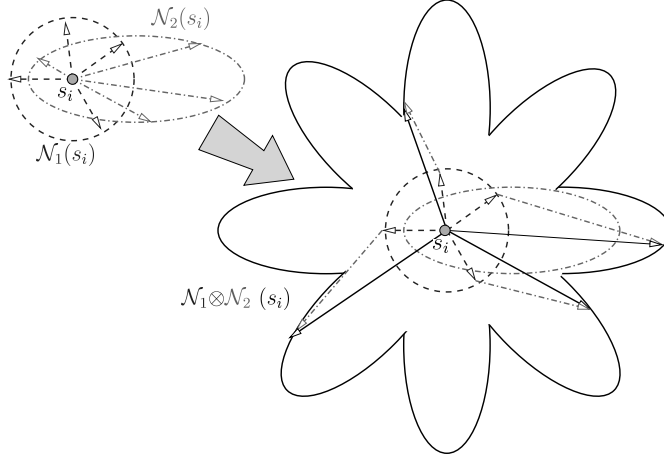


Figure 2: An example of the neighborhood sequence operator  $\otimes : \mathcal{F}_S \times \mathcal{F}_S \rightarrow \mathcal{F}_S$  for the neighborhood  $\mathcal{N}_1 \otimes \mathcal{N}_2$ .

However, this strategy could be unsatisfactory for some applications. In such cases the random distribution for selecting the neighborhood can change by taking into account also the size of each neighborhood that takes part in the union.

### 3.2 Neighborhood Sequence

Now we move to neighborhood sequence, depicted in Figure 2. The formal definition of the operator is as follows.

**Definition 3.2** (Neighborhood Sequence). *Given  $k$  neighborhood functions  $\mathcal{N}_1, \dots, \mathcal{N}_k$ , we call neighborhood sequence, denoted by  $\mathcal{N}_1 \otimes \dots \otimes \mathcal{N}_k$ , the neighborhood function defined as follows. Given two states  $s_a$  and  $s_b$ , then  $s_b$  belongs to  $\mathcal{N}_1 \otimes \dots \otimes \mathcal{N}_k(s_a)$  if there exist  $k - 1$  states  $s_1, \dots, s_{k-1}$  such that  $s_1 \in \mathcal{N}_1(s_a)$ ,  $s_2 \in \mathcal{N}_2(s_1)$ ,  $\dots$ , and  $s_b \in \mathcal{N}_k(s_{k-1})$ . Intuitively, a move is an ordered sequence of moves  $m = m_1 m_2 \dots m_k$ , where each move  $m_i$  in the sequence belongs to the neighborhood  $\mathcal{N}_i$ .*

*The strategy for selecting a random move in the compound neighborhood  $\mathcal{N}_1 \otimes \dots \otimes \mathcal{N}_k$  consists in drawing a random move  $m_i$  for each component neighborhood  $\mathcal{N}_i$  ( $1 \leq i \leq k$ ).*

*The complete exploration of the neighborhood is obtained by exploring in inverse lexicographic ordering each component neighborhood  $\mathcal{N}_i$ . In other words, given a move  $m = m_1 m_2 \dots m_j \dots m_k$  its successor is  $m' = m_1 m_2 \dots m'_j \dots m_k$ , where  $m'_j$  is the successor of  $m_j$  in the exploration of the neighborhood  $\mathcal{N}_j$  (for all  $1 \leq j \leq k$ ).*

*Concerning the definition of inverse, given a move  $m = m_1 m_2 \dots m_k$  belonging to the composite neighborhood  $\mathcal{N}_1 \otimes \dots \otimes \mathcal{N}_k$  we forbid all the moves  $m' = m'_1 m'_2 \dots m'_k$  such there is at least an index  $i \in \{1, \dots, k\}$  for which  $m'_i = m_i^{-1}$ .*

It is worth noticing that, differently from the union operator, the order of the  $\mathcal{N}_i$  for sequence is relevant, therefore it is meaningful to repeat the same  $\mathcal{N}_i$  in the sequence.

### 3.3 Neighborhood Composition

The neighborhood composition generalizes both the concepts of neighborhood union and neighborhood sequence presented above. A pictorial view of the operator is in Figure 3, whereas the formal definition of neighborhood composition is as follows.

**Definition 3.3** (Neighborhood Composition). *Given  $k$  neighborhoods functions  $\mathcal{N}_1, \dots, \mathcal{N}_k$ , and an integer  $h$ , we call neighborhood composition of step  $h$  the union of  $h$  possible sequences (also with repetitions) of all  $k$  neighborhoods. We denote a composition by  $\odot_h \mathcal{N}_1, \dots, \mathcal{N}_k$ . A move in this neighborhood is an ordered sequence of  $h$  moves  $m_1 m_2 \dots m_h$  such that  $m_i \in \mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k$ .*

*A random move in the composite neighborhood can be drawn by picking a random move in the union  $\mathcal{N}_1 \oplus \dots \oplus \mathcal{N}_k$  for each element  $m_i$  ( $1 \leq i \leq h$ ) of the move sequence.*

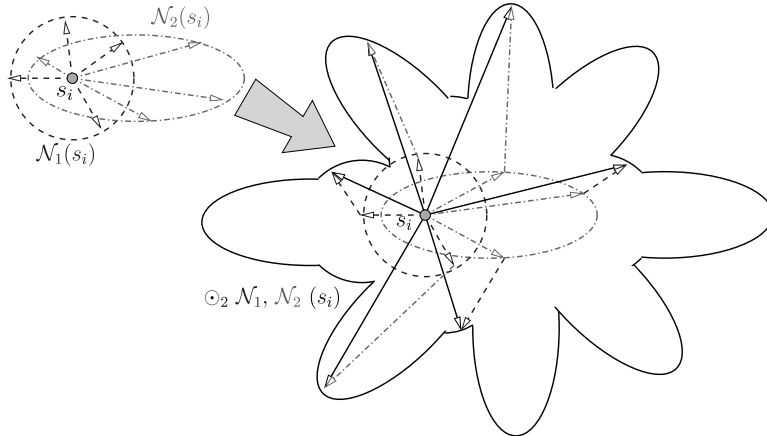


Figure 3: An example of the neighborhood composition operator  $\odot_h : \times_{i=1}^k \mathcal{F}_S \rightarrow \mathcal{F}_S$  for the neighborhood  $\odot_2 \mathcal{N}_1, \mathcal{N}_2$ .

As in the case of simple neighborhood sequence, the exhaustive exploration of the composite neighborhood is performed by exploring in inverse lexicographic ordering the whole union neighborhood for each move  $m_i$  ( $1 \leq i \leq h$ ) in the sequence.

The inverse of a move  $m = m_1 m_2 \dots m_h$  that belongs to a composition  $\odot_h \mathcal{N}_1, \dots, \mathcal{N}_k$  is the sequence of inverses of the  $h$  constituent moves taken in the reverse order, that is  $m^{-1} = m_h^{-1} \dots m_2^{-1} m_1^{-1}$ .

Note that, a composition of  $k = 1$  neighborhoods of step  $h$  gives rise to a sequence of  $h$  moves  $m_1 m_2 \dots m_h$  of the same type.

## 4 Solving Strategies

So far we have presented the operators that combine sets of neighborhoods in order to build a new composite neighborhoods, which can be plugged into any meta-heuristic giving rise to a complete local search algorithm. However, it is possible also to combine whole algorithms that make use of different neighborhood functions.

Consider again a set  $k$  of neighborhoods  $\mathcal{N}_1, \dots, \mathcal{N}_k$  defined on a search space  $S$ . Given also a set of  $n$  local search techniques, we can define  $k \times n$  different search algorithms  $t_i$ , called *searchers*, by providing each technique with any neighborhood.

A searcher can either be a basic local search technique (e.g., HC or TS) or a special purpose local search component used for intensification or diversification (see the *kickers* presented below). In what follows, the searchers based on a basic local search algorithm are called *runners*.

A solving strategy controls the search of a set of searchers  $t_i$ . For example, such a strategy prescribes in which order the algorithms  $t_i$  should be activated, or under which conditions to accept their outcomes, and so on.

In this paper we experimented with just one solving strategy, which turned out to be very effective in the solution of practical problems. The strategy is named *token-ring search* and is presented below.

The token-ring search is a sequential solving strategy for combining local search algorithms, possibly based on different neighborhoods. Given an initial state and a set of algorithms, the token-ring search makes circularly a run of each algorithm, always starting from the best solution found by the previous one.

An illustration of token-ring search is provided in Figure 4, while the formal definition of token-ring search is as follows.

**Definition 4.1** (Token-Ring Search). *Given an initial state  $s_0$ , and a set of  $q$  searchers  $t_1, \dots, t_q$  (possibly based on different neighborhoods), the token-ring search, denoted by  $t_1 \triangleright \dots \triangleright t_q$ , makes circularly a run of all  $t_i$ . The search begins applying  $t_1$  to the state  $s_0$ . Then, each searcher  $t_i$  is run, always starting from the final solution of the previous searcher  $t_{i-1}$ . When all searchers have been run, a new round is started letting  $t_1$  begin from the final solution found by  $t_q$ .*

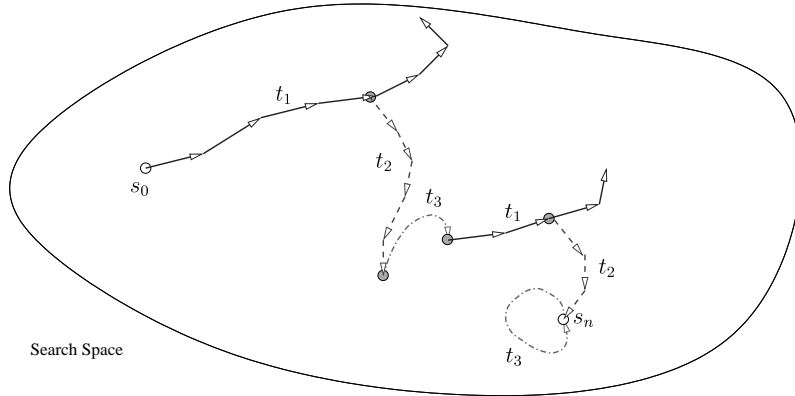


Figure 4: An example of Token-Ring Search: starting from an initial state  $s_0$  three searchers  $t_1, t_2, t_3$  are interleaved in the search reaching the state  $s_n$ .

*The token-ring search keeps track of the global best state, and it stops when performs a fixed number of rounds without an improvement of this global best. Each component searcher  $t_i$  stops according to its own specific criterion.*

Many specific cases of the general idea of the token-ring strategy have been studied in the literature. For example, the effectiveness of token-ring search for exactly two searchers has been stressed by several authors (see [6]). For example, the alternation of a TS using a small neighborhood with HC using a larger neighborhood has been used by [14] for the high-school timetabling problem. Specifically, when one of the two searchers, say  $t_2$ , is not used with the aim of improving the cost function, but rather for diversifying the search region, this idea falls under the name of *Iterated local search* (see, e.g., [17]). In this case the run with  $t_2$  is normally called the *mutation* operator or the *kick* move.

## 5 Kickers

As stressed by several authors (see, e.g., [11]), local search can benefit from alternating regular runs with some perturbations that allow the search to escape from the attraction area of a local minimum.

In the Neighborhood Portfolio settings, we define a form of perturbation, that we name *kick*, in terms of neighborhood operators. A *kicker* is a special-purpose searcher that makes just one single move, and uses a compound neighborhood of a relatively long length. A kicker can perform either a random kick, i.e., a random sequence of moves, or a best kick, which means an exhaustive exploration of the neighborhood searching for the best neighbor.

Random kicks roughly correspond to the notion of random walk used in [16], whereas the notion of best kicks is related to the idea of very large neighborhood (see, e.g., [2]).

Notice that the cardinality of a sequence is the product of the cardinalities of all the base neighborhoods, therefore if the base neighborhoods have some few thousand members, the computation of the best kick for a sequence of length  $h = 3$  or more is normally intractable. In order to reduce this complexity, we introduce the notion of *synergy*.

For every pair of neighborhood  $\mathcal{N}_1$  and  $\mathcal{N}_2$ , the user might define a set of constraints that specifies whether two moves  $m_1$  and  $m_2$ , in  $\mathcal{N}_1$  and  $\mathcal{N}_2$  respectively, are synergic or not. This relationship is typically based on equality constraints of some variables that represent the move features. If no constraint is added, the kicker assumes that all moves are synergic. The concept of synergy is illustrated in Figure 5.

A kick belonging to the neighborhood sequence is evaluated only if all pairs of adjacent moves are synergic. The intuition behind this concept is that a combination of moves, which are not all focused on the same features of the current state  $s$ , have little chance to produce improvements. In that case, in fact, the improvements would have been found by one of the runners that make one step at the time. Conversely, a good sequence of “coordinated” moves can be easily overlooked by a runner based on a simple neighborhood function.

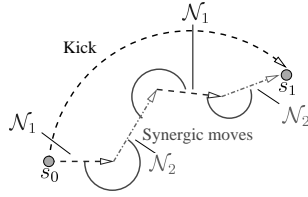


Figure 5: A kick example: starting from the state  $s_0$ , and alternating  $h = 4$  pairwise synergic moves in the neighborhoods  $\mathcal{N}_1$  and  $\mathcal{N}_2$ , the kick reaches the state  $s_1$ .

In order to build the potential kicks, the kicker employs a constraint-based backtracking algorithm that builds it starting from the current state  $s$ , along the lines presented in [12]. Differently from [12], however, all variables describing a move are instantiated simultaneously, and backtracking takes place only at “move granularity” rather than at the level of each individual variable. That is, the algorithm backtracks at level  $i$  if the current move  $m_i$  has no synergic move in the neighborhood  $\mathcal{N}_{i+1}$  that is feasible if applied in the state reached from  $s$  executing the moves of the partial sequence built up to level  $i$ .

Different definition of synergy are possible for a given problem and, in general, there is a trade-off between the time necessary to explore the neighborhood and the probability to find good moves.

## 6 Experimental methodology

In order to evaluate the Neighborhood Portfolio approach we are facing the problem of choosing an appropriate experimental methodology which takes care of the combinatorial growth of the number of algorithms to be tested. Indeed, for a simple local search model consisting of  $k$  neighborhoods and with  $h$  driving meta-heuristics we have a pool of  $k \times |\{\oplus, \otimes, \odot\}| \times h$  different algorithms (each of them even with different parameters). Then, each algorithm can be employed in a different token-ring solver and interleaved with one or more kickers exploding the number of possibilities to be tested.

To deal with this situation, we decided to analyze the algorithms by means of the RACE approach developed by Birattari *et al.* [3]. This procedure has been developed for the purpose of selecting the parameters of a single meta-heuristic by testing each candidate configuration on a set of trials. The configurations that perform poorly are discarded and not tested anymore as soon as sufficient statistical evidence against them is collected.

This way, only the statistically proven “good” configurations continue the race, and the overall number of tests needed to find the best configuration(s) is limited. Each trial is performed on the same randomly chosen problem instance for all the remaining configurations and a rank-based statistical test is used to assess which of them are discarded.

Every algorithm selected from the Neighborhood Portfolio is regarded as a race candidate and it is granted with the same amount of CPU time (300 seconds on a AMD Athlon 1.5GHz running Linux for both problems investigated in this paper).

Each trial is performed by all the remaining algorithms on the same instance and the same randomly generated initial solution. However, since the amount of CPU time is usually greater than the average running time of each algorithm we allow each algorithm to take advantage of a multi-start strategy proportionally with its speed, thus having increased chances to reach a better solution. The solution employed for restarting the search is chosen from a pool of random initial solutions that is common to all the algorithms.

Since the distribution of the final solution values is expected to be not normal, among the tests considered in the RACE approach, we are forced to employ the non-parametric Friedman test (see, e.g., [4]) for comparing the ranks of the candidate algorithms.

We choose to test the Neighborhood Portfolio approach on two timetabling problems, namely COURSE TIMETABLING and EVENT TIMETABLING which are thoroughly described in the following sections. For each problem we identify the set of basic neighborhoods and select a set of candidate algorithms by letting HC and TS drive each member of the Neighborhood Portfolio. We combine the resulting runners in a set of token-ring solver featuring a set of meaningful combinations. The first set of experiments aims

at evaluating these solvers.

Secondly we select a set of kickers built upon the basic neighborhoods and we augment a selected subset of the previous solvers with those kickers. The second set of experiments aims at looking whether the addition of the kickers improves the behavior of the solvers.

In the case of the COURSE TIMETABLING problem, the experiments have been performed on real data instances which have been made available through a web site. Conversely, for the EVENT TIMETABLING problem we exploit a set of benchmark instances used in the actual competition.

The Neighborhood Portfolio algorithms for the two problems have been implemented in the C++ language, exploiting the EASYLOCAL++ framework [5]. The current version of EASYLOCAL++ includes a set of classes that eases the efforts in the implementation of Neighborhood Portfolio-based algorithms generating automatically the code for exploration of composite neighborhood starting from the code for the basic ones. This is very important, from the practical point of view, so that the test for composite techniques is very inexpensive not only in terms of design efforts, but also in terms of human programming resources.

## 7 Case Study 1: Course Timetabling Problem

The COURSE TIMETABLING problem consists in the weekly scheduling of a set of lectures for several university courses within a given number of rooms and time periods. There are various formulations of COURSE TIMETABLING (see, e.g., [15]), which differ from each other mostly for the (hard) constraints and the objectives (or soft constraints).

We consider here the version of the problem that is used in the Faculty of Engineering of the University of Udine. However, the version of the problem in this paper is simplified, by eliminating very specific constraints, so as to reduce it to a more general form. Experiments are performed on real instances (simplified accordingly), and the data is made publicly available through the web at the URL <http://www.diegm.uniud.it/schaerf/projects/coursett/>.

A version of the software that considers the full-fledged problem is actually used to make the working timetable at the University.

### 7.1 Problem Definition

There are  $q$  courses  $c_1, \dots, c_q$ ,  $p$  periods  $1, \dots, p$ , and  $m$  rooms  $r_1, \dots, r_m$ . Each course  $c_i$  consists of  $l_i$  lectures to be scheduled in distinct time periods, and it is attended by  $s_i$  students. Each room  $r_j$  has a capacity  $cap_j$ , in terms of number of seats. There are also  $g$  groups of courses, called *curricula*, such that any two courses of a curriculum have students in common.

The output of the problem is an integer-valued  $q \times p$  matrix  $T$ , such that  $T_{ik} = j$  (with  $1 \leq j \leq m$ ) means that course  $c_i$  has a lecture in room  $r_j$  at period  $k$ , and  $T_{ik} = 0$  means that course  $c_i$  has no class in period  $k$ . We search for the matrix  $T$  such that the following *hard* constraints are satisfied, and the violations of the *soft* ones are minimized.

- (1) **Lectures (hard):** The number of lectures of course  $c_i$  must be exactly  $l_i$ .
- (2) **Room Occupancy (hard):** Two distinct lectures cannot take place in the same room in the same period.
- (3) **Conflicts (hard):** Lectures of courses in the same curriculum must be all scheduled at different times.

To deal with this constraint, we define a conflict matrix  $CM$  of size  $q \times q$ , such that  $cm_{ij} = 1$  if there is a curriculum that includes both  $c_i$  and  $c_j$ ,  $cm_{ij} = 0$  otherwise.

- (4) **Availabilities (hard):** A lecture cannot be scheduled in a period in which the teacher is declared available.
- (5) **Room Capacity (soft):** The number of students that attend a course must be less or equal to the number of seats of all the rooms that host its lectures.
- (6) **Minimum working days (soft):** Each period therefore belongs to a specific week day. The lectures of each course must be spread into the minimum number of days fixed for that course.
- (7) **Curriculum compactness (soft):** The daily schedule of a curriculum should be as much compact as possible, avoiding gaps between courses. A gap is a free period between two lectures scheduled in the same day and that belong to the same curriculum.

Instance	$q$	$p$	$\sum_i l_i$	$m$	Conflicts	Occupancy
1	46	20	207	12	4.63%	86.25%
2	52	20	223	12	4.75%	92.91%
3	56	20	252	13	4.61%	96.92%
4	55	25	250	10	4.78%	100.00%

Table 1: Features of the COURSE TIMETABLING instances used in the experiments. For each instance the columns content should be read as follows:  $q$  number of courses;  $p$  number of periods;  $\sum_i l_i$  total number of lectures;  $m$  number of room; **Conflicts** density of the conflict matrix; **Occupancy** percentage of occupancy of the rooms.

Moving to the local search modeling of this problem, our search space is composed of all the assignment matrices  $T_{ik}$  for which the constraints (1) and (4) hold. States for which the hard constraints (2) and (3) do not hold are allowed, but are penalized within the cost function.

The cost function is thus a weighted sum of the violations of the aforementioned hard constraints and the violations of the soft constraints. In order to give precedence to feasibility over the objectives, hard constraints are assigned the weight  $\alpha$ , which is a value greater than the maximum value of soft constraints violations.

The initial solution is drawn at random by creating a random matrix  $T$  that satisfies constraints (1) and (4). This is made by assigning each lecture of a course to a randomly selected available period, and to a random room, neglecting the fact whether the lectures are assigned to the same period.

## 7.2 Neighborhoods

Since in the COURSE TIMETABLING problem we are dealing with the assignment of two kinds of resources, one can very intuitively define two basic neighborhoods which deal separately with each one of these components. We call these neighborhoods Time and Room (or simply T and R for short) respectively.

The first neighborhood is defined by simply changing the period assigned to a lecture of a given course to a new one (the room remains unchanged). A move of the Time type is identified by a triple of variables  $\langle C, P, Q \rangle$ , where  $C$  represent a course,  $P$  and  $Q$  are the old and the new periods of the lecture, respectively.

The Room neighborhood, instead, is defined by changing the room assigned to a lecture in a given period. A move of this type is identified by a triple of variables  $\langle C, P, R \rangle$ , where  $C$  is a course,  $P$  is a period, and  $R$  is the new room assigned to the lecture.

Given these basic neighborhoods we define the neighborhood union  $\text{Time} \oplus \text{Room}$  whose moves are either a Time or a Room. Conversely, the neighborhood sequence  $\text{Time} \otimes \text{Room}$  involves both the resources at once. For the sequence neighborhood, we define a move  $\langle C_1, P_1, Q_1 \rangle$  of type Time and a move  $\langle C_2, P_2, R_2 \rangle$  of type Room as synergic if  $C_1 = C_2 \wedge Q_1 = P_2$ .

## 7.3 Experimental Results

Real data have been simplified to adapt to the problem version of this work, but the overall structure of the instances is not affected by the simplification.

We experiment our solvers on four real-world instances from the School of Engineering of our university. The main features of these instances are reported in Table 1. All of them have to be scheduled in 5 days of 4 or 5 periods each.

The column denoted by  $\sum_i l_i$  reports the overall number of lectures and is a measure of the size of an instance. The column ‘‘Conflicts’’ shows the density of the conflict matrix<sup>2</sup>  $CM$ , while the column ‘‘Occupancy’’ contains the percentage of occupancy of the rooms  $\sum_i l_i / (m \cdot p)$ , i.e., the ratio between the total number of lectures and the total number of ‘‘slots’’ (rooms multiplied by periods) in which these lecture could be scheduled. These two features are the main indicators of how constrained the instances are.

<sup>2</sup>The density of an  $n \times n$  matrix is defined as the ratio between the number of non-zero entries  $h$  in the matrix and the total number of non-diagonal elements:  $\frac{2h}{n(n-1)}$ .

### 7.3.1 Experiment 1: Token-ring Search

We define a pool of eight runners, obtained by equipping the hill climbing and tabu search meta-heuristics with the four neighborhoods: `Time`, `Room`, `Time⊕Room`, and `Time⊗Room`. In the following, we denote with  $\text{HC}[\mathcal{N}]$  (resp.  $\text{TS}[\mathcal{N}]$ ) the hill climbing runner that employs the neighborhood  $\mathcal{N}$ . The parameters of the runners were set with the best values found in a preliminary test phase. Namely, the tabu list is dynamic and the tabu tenure varies from 20 to 30. Concerning the number of idle iterations allowed, it is 1 million for HC and 2000 for TS.

The set of candidate solvers employed in this experimentation has been built by taking all the meaningful simple and token-ring combinations of up to three runners selected from the pool. In the end we came up with 64 different solvers.

The solvers have been tested on all the four test instances employing the RACE procedure described in Section 6 limiting the number of trials to 20. For each instance  $i$  we identify the set  $S_i$  of solvers that exhibit the best performances on that instance (i.e., they reach the last stage of the RACE procedure).

For the sake of compactness, we report only the results of those solvers that belong at least to two of the  $S_i$  sets, plus the solvers that find the best value of the objective function.

The distributions of the objective function found by these solvers are summarized in Figure 6 by means of box-and-whiskers plots. That is, for each algorithm denoted by a number on the  $x$ -axis, the graph shows the range of variation (the interval  $[\text{min\_cost}, \text{max\_cost}]$ ), indicated by the dashed vertical line, and the frequency distribution of the solutions founded. The latter measure is expressed by means of a boxed area showing the range between the 1st and the 3rd quartile of the distribution (the so-called interquartile range, that accounts for 50% of the total frequency). Finally, the horizontal line within the box indicates the median of the distribution.

We perform a post-hoc comparison among all the solvers shown in the figure employing a pairwise Wilcoxon test with Holm correction. For each instance, the median values of solvers with different subscripts differ significantly at  $p < 0.05$ .

From the results, it turns out that there is no clear winner among all instances, even though it seems that the `Time⊕Room` neighborhood is a key ingredient of a successful algorithm for this problem.

This result is not completely intuitive. Indeed, one may guess that a larger neighborhood (as `Time⊗Room`) should have better chances to find good solutions. Instead, for this problem the advantage of exploring a larger neighborhood is not worth its computational cost.

### 7.3.2 Experiment 2: Search & Kick

In order to evaluate the effect of kickers in joint action with the proposed solvers we define two sets of kickers, the first one based on the composition  $\odot_h \text{Time,Room}$  and the second one featuring only sequence moves,  $\odot_h \text{Time}\otimes\text{Room}$ .

For the composition kickers, we provide also two different definitions of the synergic moves for the four neighborhood combinations. The first one, presented above, is more strict and requires that the moves “insist” on the same period and on the same room. The second one is more relaxed and allows also combinations of moves on different rooms (it is denoted by a \* in the following figures). In our experimentation, we employ these kickers with random kicks of size  $h = 10$ ,  $h = 20$  and  $h = 30$ , and best kicks with  $h = 2$  or  $h = 3$  steps.

In the following, a kicker of length equipped with the neighborhood  $\mathcal{N}$  will be denoted with  $\text{K}_{type}[\mathcal{N}]$ . The subscript *type* can assume values *b* or *r* indicating that the strategy employed by the kicker is to find the best kick or just to draw a random kick, respectively.

As in the previous experiment, we report only the results of the solvers which remained in the RACE for at least two instances plus the best performers.

The results of the solvers are shown in Figure 7 in four different plots (one for each of the instances). The solvers denoted by the symbol † exhibit a similar behavior for all the random kick lengths, therefore we report only the data for  $h = 10$ .

In the plots, each solver is identified by a number and, for each instance, common subscripts indicate median cost values that do not differ significantly by paired post-hoc Wilcoxon tests ( $p < 0.05$ ). For example, in the case of Instance 3, solver 3 has subscripts *bc* meaning that its median is significantly different from the median of solver 1, which has the subscript *a* (and its behavior is superior than the

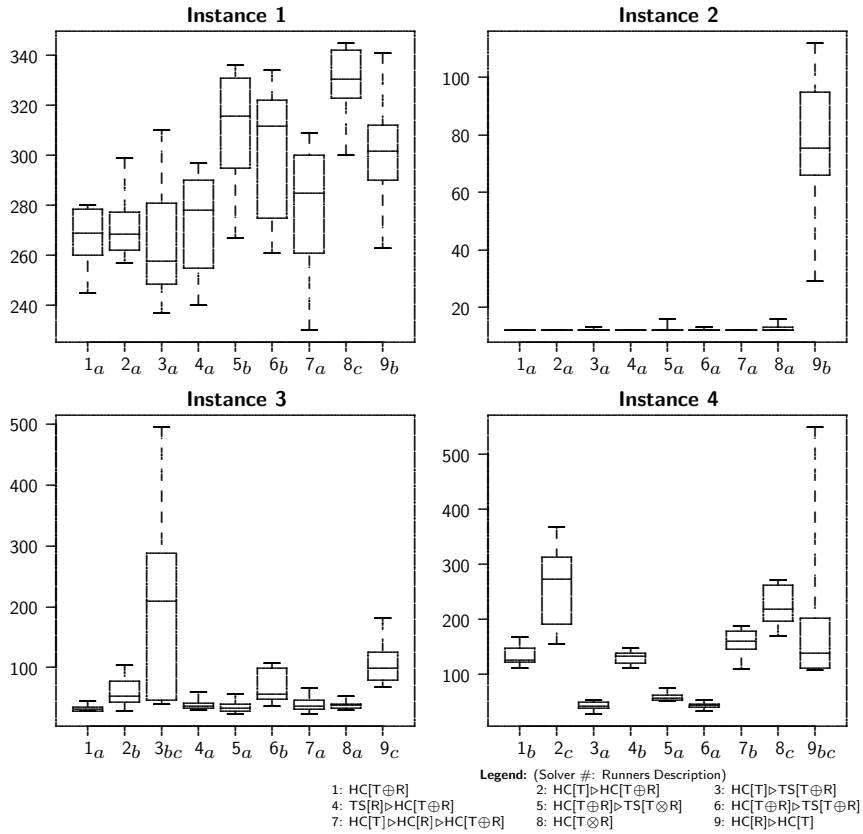


Figure 6: Cost value distributions of Token-Ring solvers for COURSE TIMETABLING. For each instance, the median value of solvers with different subscripts differ significantly at  $p < 0.05$ , by a two-tailed Wilcoxon test with Holm correction. For a description of the solvers see the text.

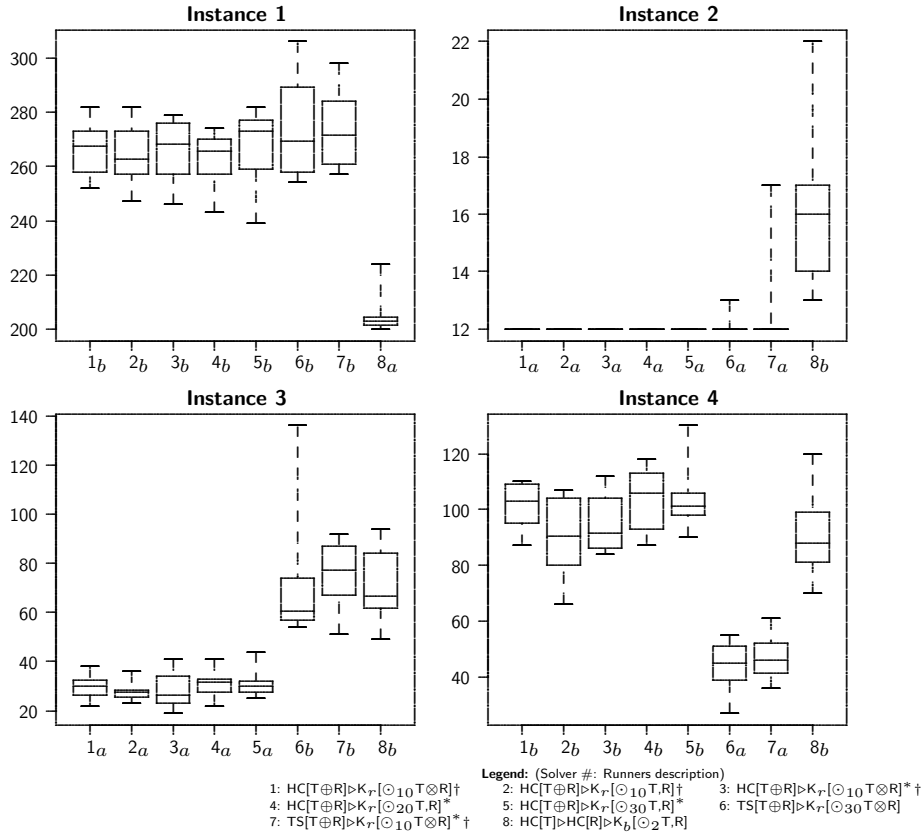


Figure 7: Cost value distributions of Kicker-equipped solvers for COURSE TIMETABLING. For each instance, the median value of solvers with different subscripts differ significantly at  $p < 0.05$ , by a two-tailed Wilcoxon test with Holm correction. The solvers denoted by † exhibit a similar behavior for all the random kick lengths, For a description of the solvers see the text.

one of solver 1). However solver 3 does not differ significantly neither with the median of solver 2 (with subscript  $b$ ) nor with that of solver 9 (with subscript  $c$ ).

Comparing these results with those of the previous table, we see that the use of kickers provides a remarkable improvement on the solvers. Specifically, kickers implementing the random kick strategy increase the ability of the local search algorithms independently of the search technique employed.

Another effect of the kickers is the improvement of algorithm robustness measured in terms of interquartile range (i.e., the difference between the 3rd quartile and the 1st quartile values) of the results. In other words, the outcomes of the single trials aggregate toward the average value, while they are more scattered with the plain solvers only.

Looking at the influence of different synergy definitions, it is possible to see that there is no clear benefit in employing one definition rather than the other.

The most important outcome, however, is that the “winning” solvers are not simple ones, but complex token ring ones automatically generated by the approach.

## 8 Case Study 2: Event Timetabling Problem

The EVENT TIMETABLING problem consists in the scheduling of a set of university events within a given number of rooms and time periods. As for the COURSE TIMETABLING, there are many formulations of EVENT TIMETABLING. The version we consider here is the one designed for the International Timetabling Competition TTComp2002.

The competition has been organized by the Metaheuristic Network, and it had 24 feasible submissions from all over the world. All information about the detailed problem definition, instances, rules, and solu-

tion evaluation of TTComp2002 is available at the webpage <http://www.idsia.ch/Files/ttcomp2002/>, but we report the most relevant parts in this section.

The solver described in this paper resulted fourth in the general ranking.

## 8.1 Problem statement

The timetabling problem consists in assigning a set of  $n$  events  $e_1, \dots, e_n$  to a set of  $p$  timeslots  $1, \dots, p$ , in a set of  $m$  rooms  $r_1, \dots, r_m$ . In this formulation, the number of timeslots  $p$  is fixed to 45, split in 5 days of 9 timeslots each, whereas  $n$  and  $m$  vary from instance to instance.

It is also given a set of students  $S$  that attend the events, and a set of features  $F$  that may be available in rooms and are required by events. Rooms also have a fixed capacity, expressed in terms of seats for students.

The constraints of the problem are the following ones:

- (1) **Occupancy (hard)**: No more than one event per room per timeslot is allowed.
- (2) **Conflicts (hard)**: Events that share common students cannot be scheduled in the same timeslot.
- (3) **Compatibility (hard)**: In order to allocate an event in a room, the room must fulfill the feature requirements of the event, and its capacity must be at least equal to the number of students attending the event.
- (4) **Late event (soft)**: a student should not attend an event in the last slot of a day
- (5) **Consecutive events (soft)**: a student should not attend more than two consecutive events (the last event of a day and the first of the following day are not considered consecutive)
- (6) **Isolated event (soft)**: a student should not attend only one single event in the whole day

The (integer-valued) objective function is the *unweighted* sum of the soft constraints. That is, each violation of any of the three kinds accounts as one point in the objective function.

According to the constraints, from the input data we compute two boolean-valued matrices: an  $n \times n$  event-event matrix, that stores conflicting events, and an  $n \times m$  event-room matrix that stores the compatibility relation between events and rooms. These two matrices, plus the student-event *enrolment* matrix, are all we need for solving an instance.

## 8.2 Search Space, Cost Function, and Initial State

The search space for the problem is represented by a vector  $V$  of size  $|E|$  (where  $E$  is the set of events) that stores pairs of integers representing, respectively, the timeslots and the rooms assigned to each event. A given vector represents a legal state if it satisfies component (3).

The cost function is thus a sum of the violations of the hard constraints (1) and (2) (with a high weight) plus the violations of the soft constraints.

The initial solution is selected at random. That is, we create a random vector  $V$  that satisfies constraints (3). This is made by assigning each event to a randomly selected period, and to a random room in the set of the compatible ones.

## 8.3 Neighborhoods

For this problem we identified four promising neighborhoods that have been used in the solver developed for the competition and that we want to investigate more thoroughly in the Neighborhood Portfolio approach. The first two neighborhoods are the straightforward adaptation to this problem of the two neighborhoods Time and Room proposed for COURSE TIMETABLING.

We define also a more restricted version of the Room neighborhood, denoted by Room\*, which allows only moves that assign the given event to an empty room. Following the notation introduced in the previous section, this corresponds for the move  $\langle C = e_i, P = k, R = r_j \rangle$  to constraint room  $r_j$  to be empty in the current state.

The fourth neighborhood SwapTime (or simply ST for short), instead, is characteristic to this problem and it is defined as the swap of the timeslots assigned to two events, each of them taking the room of the other.

On the basis of these basic neighborhoods we define the neighborhood union Time $\oplus$ Room $\oplus$ SwapTime and the two neighborhood sequences Time $\otimes$ Room and Time $\otimes$ Room\* with the same synergy definition employed in the previous case study.

Instance	$n$	$m$	$ F $	$ S $	Conflicts
1	400	10	10	200	36.23%
3	400	10	10	200	38.84%
6	350	10	5	300	51.75%
15	350	10	10	300	44.76%

Table 2: Features of the EVENT TIMETABLING instances employed in the experimentations. For each instance, the columns should be read as follows:  $n$  number of events;  $m$  number of rooms;  $|F|$  cardinality of the set of features;  $|S|$  number of students; **Conflicts** density of the conflict matrix.

Parameter	$d \in [0, 0.38[$	$d \in [0.38, 0.45[$	$d \in [0.45, 0.5[$	$d \in [0.5, 1]$
Idle iterations	5,000	4,000	3,000	2,000
Total iterations	15,000	12,000	9,000	6,000
$k_{min} \div k_{max}$	20–30	20–30	30–40	30–40

Table 3: Parameter settings of the TS runners. The various parameters are set on the basis of the density  $d$  of the conflict matrix.

## 8.4 Experimental Results

The solvers were tested on four out of the twenty competition instances, which were selected on the basis of the density of the event-event conflict matrix. We have selected one instance for each of the density thresholds identified in the development of the competition solver. As explained in the following subsection, each threshold value corresponds to a specific parameter settings of the TS runners.

The features of the instances selected are reported in Table 2.

### 8.4.1 Experiment 1: Token-Ring Search

Based on preliminary tests performed during the competition time, we identified four promising runners, namely a HC driving the neighborhood union  $\text{Time} \oplus \text{Room} \oplus \text{SwapTime}$ , a TS equipped with the  $\text{Time} \otimes \text{Room}^*$  move, a HC equipped with the  $\text{Time} \otimes \text{Room}$  move and, finally, a TS driving the  $\text{SwapTime}$  neighborhood.

The parameter settings for the runners are as follows. The stopping criterion for the HC runners is based on the number of *idle iterations*, that is the iterations elapsed since the last improvement. We fixed this number to 300,000 for our experiments.

The stopping condition for the TS runners is a compound one: the meta-heuristics stops after a number of idle iterations that is inversely proportional to the density of the event-event conflict matrix of the instance or when a fixed total number of iterations has elapsed.

We have been forced to give less idle iterations to denser instance in order to save time for the other stages (time limit is the same for all instances). In fact, for a given value of idle iterations, a run of a denser instance normally takes longer because it tends to spend more iterations between two improvements, and consequently the stopping criterion must be more stringent.

The parameter settings for the TS runners are summarized in Table 3 and they are grouped by the density  $d$  of the conflict matrix. The density thresholds were experimentally adjusted to give a good compromise between the running time granted to the algorithm and the quality of the solution found.

The set of candidate solvers employed in this experimentation has been built by taking all the meaningful simple and token-ring combinations of up to three runners selected from the pool, totalling a number of 14 solvers.

The solvers have been tested on the four instances and, as in the previous case, we report only the results of those solvers that remained in the RACE procedure for both groups, plus the solvers that find the best value of the objective function. For each instance we allow 20 trials of the RACE procedure.

The distributions of the objective function found by the solvers are summarized in Figure 8. Solvers that are found to be significantly different (w.r.t. median values) after the post-hoc pairwise Wilcoxon test are denoted by different subscripts.

The solver denoted by  $\text{HC}[\text{T} \oplus \text{R} \oplus \text{ST}] \triangleright \text{TS}[\text{T} \otimes \text{R}]^*$  is the actual solver employed in the competition. The presented results confirm our choice, since this solver exhibits the best behavior for the EVENT

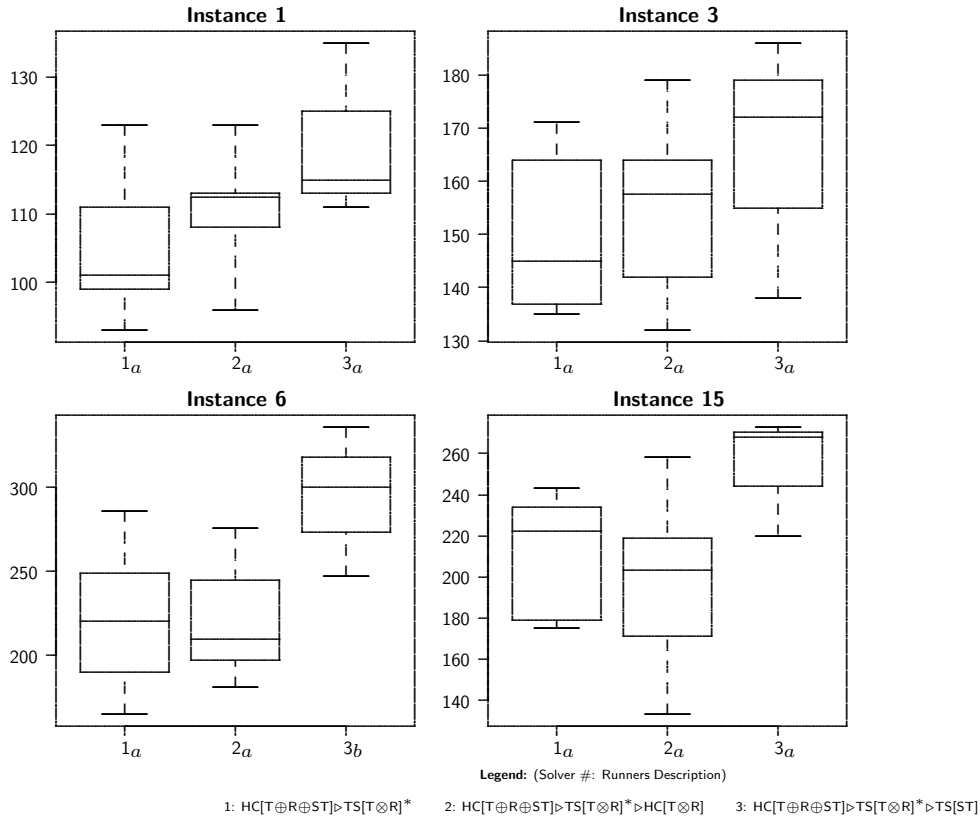


Figure 8: Cost value distributions of Token-Ring solvers for EVENT TIMETABLING. For each instance, the median value of solvers with different subscripts differ significantly at  $p < 0.05$ , by a two-tailed Wilcoxon test with Holm correction. For a description of the solvers see the text.

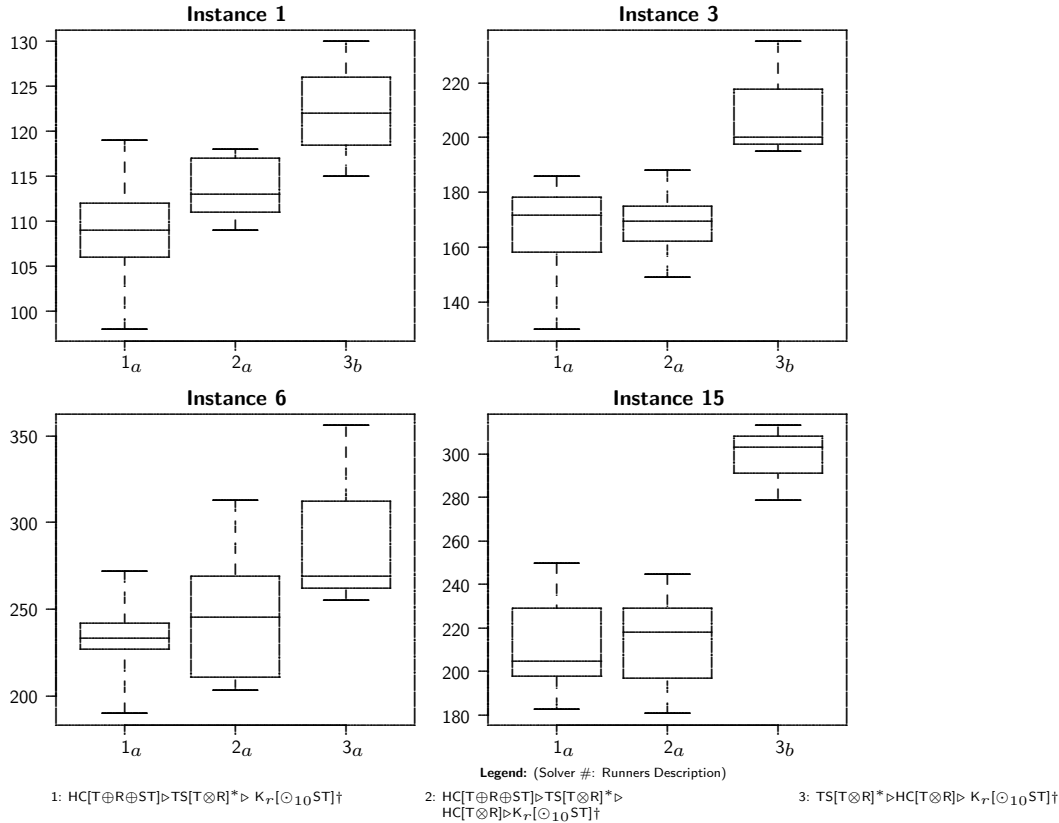


Figure 9: Cost value distributions of Kicker-equipped solvers for EVENT TIMETABLING. For each instance, the median value of solvers with different subscripts differ significantly at  $p < 0.05$ , by a two-tailed Wilcoxon test with Holm correction. The solvers denoted by  $\dagger$  exhibit a similar behavior for all the random kick lengths. For a description of the solvers see the text.

TIMETABLING problem.

These results, however, are inferior to the best values found by the  $HC[T \oplus R \oplus ST] \triangleright TS[T \otimes R]^*$  solver in the competition. Unfortunately, though, the results are not directly comparable since the competition rules favor the algorithms that are able to find a single good outcome even in a high number of attempts. The purpose of the experimentation, instead, is to evaluate the general behavior of the solvers.

#### 8.4.2 Experiment 2: Search & Kick

For the second experiment we equip the 14 solvers employed in the previous experiment with a kicker based on the `SwapTime` move. We test this kicker both with random and best kicks of length  $h = 10, 20, 30$  and  $h = 2, 3$ , respectively.

The results of the solvers are reported in Figure 9. As in the previous case study, the solvers denoted by a  $\dagger$  are not statistically different for all lengths of the kicks.

Differently from the previous case study, for the EVENT TIMETABLING the use of the proposed Kicker does not increase the ability of the solvers to find better solutions.

Other experiments performed for the competition, not reported here, confirm this behavior also for other kind of kickers. A complete understanding of the reasons of this behavior will be matter for further investigations.

In conclusions, like for the COURSE TIMETABLING problem, the best solvers for EVENT TIMETABLING are complex combinations of neighborhoods and runners generate as automatic combinations of the basic ones.

## 9 Discussion and Conclusions

We have presented a systematic approach for combining different neighborhoods for a given local search problem, that generalizes previous ideas presented in the literature. The Neighborhood Portfolio approach is based on a set of operators for neighborhood combination and in a solving strategy that interleaves basic algorithms possibly equipped with different neighborhoods. Furthermore we define a local search component, called kicker, that is meant to implement a perturbation mechanism based on neighborhood composition.

The benefits of the proposed approach resides in the complete generality of our neighborhood operators, in the sense that, given the basic neighborhoods, the synthesis of the proposed algorithms is automatic, and it requires the human intervention only for a few features (e.g., the synergy relation).

The experimental results confirm that the Neighborhood Portfolio approach is superior to the plain algorithms based on the simple neighborhoods. In fact, for both problems all algorithms of the latter kind were discarded in the RACE tests.

Regarding the comparison with previous work, for COURSE TIMETABLING this is unfortunately not possible because no public instances were available. For EVENT TIMETABLING, our solver resulted fourth in the general ranking (out of about 24 participants), but it is actually the sixth best solver if we include also the two teams that were not included in the list because the authors were members of the Network. Despite the ranking, it is quite difficult to compare the quality of the local search algorithms, because all the solvers ahead of ours used a different search space (see [13]). Specifically, the room assignment is not part of the search space but it is generated a posteriori by a matching algorithm that assigns events to available rooms, separately for each timeslot. The room matching can be done in polynomial time in a perfect way and it is indeed performed efficiently.

We followed our approach in order to experiment line of research which would have been entirely based on local search. This choice was dictated also by the possibility to extend the solver to more complex real settings, in which the matching is not a viable option. For example, if we are allowed to express a constraint requiring that events in different periods have to be scheduled in the same room, then the matching becomes more critical. In fact, the addition of constraints on the matching makes, in most cases, the problem NP-complete [9].

### Acknowledgements

We thank the two anonymous reviewers for their detailed comments and suggestions for improving this paper. This work has been partly funded by the Italian Ministry of Education, University and Research (MIUR) under the project PRIN 2003 “Design and implementation of a solver based on local search for the execution of declarative specifications for combinatorial problems”.

### References

- [1] Aarts, E. and J. K. Lenstra (eds.): 1997, *Local search in combinatorial optimization*. Chirchester, UK: John Wiley & Sons.
- [2] Ahuja, R. K., J. B. Orin, and D. Sharma: 2000, ‘Very large scale neighborhood search’. *International Transactions in Operations Research* **7**, 301–317.
- [3] Birattari, M., T. Stützle, L. Paquete, and K. Varrentrapp: 2002, ‘A Racing Algorithm For Configuring Metaheuristics’. In: W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska (eds.): *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA, pp. 11–18.
- [4] Conover, W.: 1999, *Practical Nonparametric Statistics*. New York, NY, USA: John Wiley & Sons, third edition.
- [5] Di Gaspero, L. and A. Schaerf: 2003, ‘EASYLOCAL++: An object-oriented framework for flexible design of local search algorithms’. *Software—Practice and Experience* **33**(8), 733–765.

- [6] Glover, F. and M. Laguna: 1997, *Tabu search*. Boston, MA, USA: Kluwer Academic Publishers.
- [7] Hansen, P. and N. Mladenović: 1999, ‘An Introduction to Variable Neighbourhood Search’. In: S. Voß, S. Martello, I. Osman, and C. Roucairol (eds.): *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Boston, MA, USA: Kluwer Academic Publishers, pp. 433–458.
- [8] Hoos, H. H. and T. Stützle: 2005, *Stochastic local search foundations and applications*. San Francisco, CA, USA: Morgan Kaufmann.
- [9] Itai, A., M. Rodeh, and S. L. Tanimoto: 1977, ‘Some Matching Problems for Bipartite Graphs’. Technical Report TR93, IBM Israel Scientific Center, Haifa, Israel.
- [10] Kirkpatrick, S., C. D. Gelatt Jr. and M. P. Vecchi: 1983, ‘Optimization by simulated annealing’. *Science* **220**, 671–680.
- [11] Lourenço, H. R., O. Martin, and T. Stützle: 2003, ‘Applying Iterated local search to the permutation flow shop problem’. In: F. Glover and G. Kochenberger (eds.): *Handbook of Metaheuristics*. Boston, MA, USA: Kluwer Academic Publishers.
- [12] Pesant, G. and M. Gendreau: 1999, ‘A constraint programming framework for local search methods’. *Journal of Heuristics* **5**, 255–279.
- [13] Rossi-Doria, O. et al.: 2003, ‘A comparison of the performance of different metaheuristic on the timetabling problem’. In: E. Burke and P. D. Causmaecker (eds.): *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002), selected papers*, Vol. 2740 of *Lecture Notes in Computer Science*. Berlin-Heidelberg, Germany, pp. 329–351.
- [14] Schaerf, A.: 1996, ‘Tabu Search Techniques for Large High-School Timetabling Problems’. In: *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*. Portland, OR, USA, pp. 363–368.
- [15] Schaerf, A.: 1999, ‘A Survey of Automated Timetabling’. *Artificial Intelligence Review* **13**(2), 87–127.
- [16] Selman, B., H. A. Kautz, and B. Cohen: 1994, ‘Noise strategies for improving local search’. In: *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*. Seattle, WA, USA, pp. 337–343.
- [17] Stützle, T.: 1998, ‘Iterated local search for the quadratic assignment problem’. Technical Report AIDA-99-03, FG Intellektik, TU Darmstadt.
- [18] Vaessens, R., E. Aarts, and J. K. Lenstra: 1996, ‘Job shop scheduling by local search’. *INFORMS Journal of Computing* **8**(3), 302–317.
- [19] Voß, S., S. Martello, I. Osman, and C. Roucairol (eds.): 1999, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Boston, MA, USA: Kluwer Academic Publishers.