Multi-neighbourhood Local Search with Application to Course Timetabling

Luca Di Gaspero[1] and Andrea Schaerf[2]

[1] Dipartimento di Matematica e Informatica,
Università di Udine,
via delle Scienze 206, I-33100, Udine, Italy
digasper@dimi.uniud.it
[2] Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica,
Università di Udine,
via delle Scienze 208, I-33100, Udine, Italy
schaerf@uniud.it

Abstract. A recent trend in local search concerns the exploitation of several different neighbourhood functions so as to increase the ability of the algorithm to navigate the search space.
In this paper we investigate the use of local search techniques based on various combinations of neighbourhood functions, and we apply this to a timetabling problem. In particular, we propose a set of generic operators that automatically compose neighbourhood functions, giving rise to more complex ones. In the exploration of large neighbourhoods, we rely on constraint techniques to prune the list of candidates. In this way, we are able to select the most effective search technique through a systematic analysis of all possible combinations built upon a set of basic, human-defined, neighbourhood functions.
The proposed ideas are applied to a practical problem, namely the Course Timetabling problem. Our algorithms are systematically tested and compared on real-world instances. The experimental analysis shows that neighbourhood composition leads to much better results than traditional local search techniques.

1 Introduction

Local search is a successful meta-heuristic paradigm for the solution of constraint satisfaction and optimization problems. Main local search strategies, such as hill climbing, simulated annealing and tabu search (see, e.g., [1]), have proved to be very effective in a large number of problems.

One of the most critical features of local search is the definition of the neighbourhood structure. In fact, for most popular problems, many different neighbourhood structures have been considered and experimented with. For example, for Job–Shop Scheduling, at least ten different ones have appeared in the literature (see [19]). Moreover, for most common problems, there is more than one neighbourhood structure that is sufficiently natural and intuitive to deserve systematic investigation.

E. Burke and P. De Causmaecker (Eds.): PATAT 2002, LNCS 2740, pp. 262–275, 2003.

One of the attractive properties of the local search paradigm is its flexibility, in the sense that different techniques and neighbourhoods can be combined and alternated to give rise to complex algorithms. The main motivation for considering the combination of diverse neighbourhoods is related to the diversification of search needed to escape from local minima. In fact a solution that is a local minimum for a given definition is not necessarily a local minimum for another one, and thus an algorithm that uses both has more chances to move toward better solutions.

There are actually many ways to combine different neighbourhoods and different algorithms. In this work, we formally define and investigate the following three:

Neighbourhood union. We consider as neighbourhood the union of many neighbourhoods. The algorithm at each iteration selects a move belonging to any of the components.

Neighbourhood composition. We consider as atomic moves, chains of moves belonging to different neighbourhoods.

Token-ring search. Given an initial state and a set of algorithms based on different neighbourhood functions, the token-ring search makes circularly a run of each algorithm, always starting from the best solution found by the previous one.

These three notions are not completely new, and they have been proposed in the literature in similar forms (under various names). For example, the effectiveness of token-ring search for two neighbourhoods has been stressed by several authors (e.g. [7]). In particular, when one of the two algorithms is not used with the aim of improving the cost function, but exclusively for diversifying the search region, this idea falls under the name of *iterated local search* [11]. As an example, in [15] we employ the alternation of tabu search using a small neighbourhood with hill climbing using a larger neighbourhood for the solution of the high-school timetabling problem.

The alternation of simple search and move chains is also the basis of the so-called *Variable Neighbourhood Search* strategy proposed by Hansen and Mladenović [8], which has been used in many applications (see, e.g., [3]).

Our contribution consists in the attempt to systematize the different ideas in a general multi-neighbourhood framework, and to perform a comprehensive experimental analysis on a real application. In addition, we want to exploit constraint propagation techniques in local search, in the spirit of [13], so as to speed up the exploration of large neighbourhoods.

Our case study is the Course Timetabling (CTT) problem [2,16]. An application built around the algorithms presented here is actually used to make the working timetable at the Faculty of Engineering of the University of Udine. However, the version of the problem we consider 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 are available at http://www.diegm.uniud.it/schaerf/projects/coursett/.

The experimental results confirm that algorithms based on combinations of neighbourhoods performs much better than basic ones.

2 Local Search

Local search is a family of general-purpose search techniques, which was first introduced more than 35 years ago [10]. It has become quite popular in AI after the seminal papers by Minton et al. [12] and Selman et al. [18]. Local search techniques are *non-exhaustive* in the sense that they do not guarantee to find a feasible (or optimal) solution, but they search non-systematically until a specific stop criterion is satisfied.

2.1 Local Search Basics

Given an instance p of a search or optimization problem P, we associate a *search space* S with it. Each element $s \in S$ corresponds to a potential solution of p, and is called a *state* of p. Local search relies on a function N which assigns to each $s \in S$ its *neighbourhood* $N(s) \subseteq S$. Each $s' \in N(s)$ is called a *neighbour* of s.

A local search algorithm t 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 one state s_i to one of its neighbours s_{i+1}.

The neighbourhood of a state s can be described in terms of changes (called *moves*) that are applied to transform s in the members of $N(s)$. A move is typically composed by a limited set of attributes (or variables) that describes the changes to the state. Given a state s and a move m, we denote by $s \circ m$ the state obtained from s applying the move m. Therefore a neighbourhood can be seen as a set of moves, even though not all moves can be applied in any state s, because some moves might be *infeasible*, i.e. they lead to a state outside the search space.

Local search techniques differ from each other according to the strategy they use both to select the *move* in each state and to stop the search. In all techniques, the search is driven by a *cost function* f that estimates the quality of the state. For optimization problems, f generally accounts for the number of violated constraints and for the objective function of the problem.

Two of the most common local search techniques are *hill climbing* (HC) and *tabu search* (TS). We describe them here; however, a full description of HC and TS is beyond the scope of this paper (see, e.g., [7]). We only present the formulations and the concepts which are used in this work.

2.2 Hill Climbing

HC is actually 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.

We employ the so-called *randomized non-ascending* strategy which selects a random move m_i at each iteration i, and if $f(s_i \circ m_i) \leq f(s_i)$ then let $s_{i+1} = s_i \circ m_i$, otherwise let $s_{i+1} = s_i$.

HC does not stop when it reaches a local minimum. In fact, the search might loop infinitely by cycling among two or more states 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, i.e. it stops at iteration j such that $f(s_j) = f(s_{j-1}) = \cdots = f(s_{j-n})$.

2.3 Tabu Search

At each state s_i, TS explores exhaustively the current neighbourhood $N(s_i)$. Among the elements in $N(s_i)$, the one that gives the minimum value of the cost function becomes the new current state s_{i+1}, independently of whether $f(s_{i+1})$ is less or greater than $f(s_i)$.

Such a choice allows the algorithm to *escape* from local minima, but creates the risk of cycling among a set of states. In order to prevent cycling, the so-called *tabu list* is used, which determines the forbidden moves. This list stores the most recently accepted moves. The *inverses* of the moves in the list are forbidden.

The simplest way to run the tabu list is as a queue of fixed size k. That is, when a new move is added to the list, the oldest one is discarded. We employ a more general mechanism which assigns to each move that enters the list a random *tenure*, i.e. each move remains in the list for a random number of steps varying between two values k_{min} and k_{max}. When its tabu period is expired, a move is removed from the list. In this way the size on the list is not fixed, but varies dynamically between k_{min} and k_{max}.

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

Also in this case, like HC, the search is stopped when no improvement of the cost function is found after n iterations.

As a final remark, we must mention that we use just one of the simplest forms of TS: more involved ones include sophisticated prohibition strategies and mechanisms for long-term memory. However, as a matter of fact, the algorithm described here is the most employed in the TS literature.

3 Multi-neighbourhood Search

Consider a problem, a search space S for it and a set k of neighbourhood functions $N_1, \ldots, N_k$ defined on S. Given also a set of n local search techniques (in this work $n = 2$, namely HC and TS), we can define $k \times n$ different search algorithms, called *runners*, by combining any technique with any neighbourhood function.

The functions N_i are obviously problem dependent, and they are defined by the person who investigates the problem. In this section, we show that, given a set of human-defined neighbourhood functions, we can automatically create new runners, using some composition operators.

3.1 Neighbourhood Union

Given k neighbourhood functions $N_1, \ldots, N_k$, we call a *union*, written as $N_1 \oplus \cdots \oplus N_k$, the neighbourhood function such that, for each state s, the set $N_1 \oplus \cdots \oplus N_k(s)$ is equal to $N_1(s) \cup \cdots \cup N_k(s)$.

According to the above definition, a HC runner that uses the neighbourhood $N_1 \oplus \cdots \oplus N_k$ selects at each iteration a random move from any N_i, whereas a TS runner explores all N_i exhaustively and selects the overall best solution.

The random distribution for selecting a move in $N_1 \oplus \cdots \oplus N_k$ from s is the following: we first select a random i (with $1 \leq i \leq k$) and then a random state $s' \in N_i(s)$. The selection thus is not uniform, because it is not weighted based on the cardinality of the sets $N_i(s)$.

3.2 Neighbourhood Composition

Given k neighbourhood functions $N_1, \ldots, N_k$, we call *composition*, denoted by $N_1 \otimes \cdots \otimes N_k$, the neighbourhood function defined as follows. Given two states s_a and s_b, then s_b belongs to $N_1 \otimes \cdots \otimes N_k(s_a)$ if there exist $k-1$ states $s_1, \ldots s_{k-1}$ such that $s_1 \in N_1(s_a)$, $s_2 \in N_2(s_1)$, $\ldots$, and $s_b \in N_1(s_{k-1})$.

Intuitively, a composite move is an ordered sequence of moves belonging to the component neighbourhoods, i.e. $m = m_1 m_2 \ldots m_k$ with $m_i \in N_i$. Differently from the union operator, for composition the order of the N_i is relevant, and it is meaningful to repeat the same N_i in the composition.

Given the k neighbourhood functions and an integer h, we call *total composition* of step h the union of all possible compositions (also with repetitions) of all k neighbourhoods. We denote a total composition by $\odot_h N_1, \ldots, N_k$. A move in this neighborhood is an ordered sequence of h moves $m_1 m_2 \ldots m_h$ such that $m_i \in N_1 \oplus \cdots \oplus N_k$. In other words, each move m_i ($1 \leq i \leq h$) can be chosen in any neighborhood N_j ($1 \leq j \leq k$).

3.3 Token-Ring Search

Given an initial state s_0, and a set of q *runners* $t_1, \ldots, t_q$, the token-ring search, denoted by $t_1 \triangleright \cdots \triangleright t_q$, makes circularly a run of all t_i. Each t_i always starts from the final solution of the previous runner t_{i-1} (or t_q if $i = 1$).

The token-ring search keeps track of the global best state, and stops when it performs a fixed number of rounds without an improvement of this global best. The component runners t_i stop according to their own specific criteria.

3.4 Local Search Kickers

As noticed 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 our settings, we define a form of perturbation, that we call *kick*, in terms of neighbourhood compositions. A *kicker* is a runner that makes just one single move, and uses a neighbourhood composition (total or simple) 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 composite neighbourhood searching for the best sequence.

Random kicks roughly correspond to the notion of random walk used in [17]. The notion of best kicks is based on the idea of ejection chains (see, e.g., [14]), and generalizes it to generic chains of moves (from different neighbourhoods). Experiments with kickers as part of a token-ring search, called Run & Kick, are shown in our case study, and, as highlighted in Section 5, the use of best kicks turned out to be very effective in our test instances.

Notice that the cardinality of a composition is the product of the cardinalities of all the base neighbourhoods, therefore if the base neighbourhoods have some few thousand members, the computation of the best kick for a composition of length 3 or more is normally intractable. In order to reduce this complexity, we introduce the problem-dependent notion of *synergic moves*. For every pair of neighbourhood functions N_1 and N_2, the user might define a set of constraints that specifies whether two moves m_1 and m_2, in N_1 and 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.

A move sequence belonging to the neighbourhood composition is evaluated only if all pairs of adjacent moves are synergic. The intuition behind the idea of synergic moves is that a combination of moves that 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 neighbourhood function.

In order to build kicks, i.e. chains of synergic moves, the kicker makes use of a constraint-based backtracking algorithm that builds it starting from the current state s, along the lines of [13]. Differently from [13], 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 neighbourhood 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.

Notice that the use of a backtracking algorithm for the exploration of the composite neighbourhood does not mean that this process is exponential in nature. In fact, the size of the compound neighborhood for a kick of step n is bound by

the product of the size of the component neighborhoods N_i ($i = 1, \ldots, n$). More correctly, the size of the compound neighborhood grows exponentially w.r.t. the number of neighborhoods involved, but in our experimentation we always choose a constant value for n that is small enough to ensure an efficient computation of kicks.

Different definitions of synergy are possible for a given problem. In general, there is a trade-off between the time necessary to explore the neighbourhood and the probability to find good moves. In our case study, we experiment with two different definitions of synergy and compare their results.

4 A Case Study: Course Timetabling

The CTT problem consists in the weekly scheduling of lectures for a set of courses. There are various formulations of the CTT problem (see, e.g., [16]), which mostly differ from each other in the hard and soft constraints (or objectives) they consider. For the sake of generality, we consider in this work a basic version of the problem.

4.1 Problem Definition

There are q courses $c_1, \ldots, c_q$, p periods $1, \ldots, p$, and m rooms $r_1, \ldots, r_m$. 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. Hard constraints must be always satisfied in the final solution of the problem, whereas soft constraints can be violated, but at the price of deteriorating the solution quality.

(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.
    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): Teachers might be not available for some periods. We define an availability matrix A of size $q \times p$, such that $a_{ik} = 1$ if lectures of course c_i can be scheduled at period k, $a_{ik} = 0$ otherwise.
(5) Room capacity (soft): The number of students that attend a course must be less than or equal to the number of seats of all the rooms that host its lectures.

(6) Minimum working days (soft): The set of periods p is split into wd days of p/wd periods each (assuming p divisible by wd). Each period therefore belongs to a specific week day. The lectures of each course c_i must be spread into a minimum number of days d_i (with $d_i \leq k_i$ and $d_i \leq wd$).
(7) Curriculum compactness (soft): The daily schedule of a curriculum should be as 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.

4.2 Search Space, Cost Function, and Initial State

In order to solve CTT by local search, first we have to define the search space. 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 considerably penalized within the cost function.

The cost function is thus a weighted sum of the violations of the aforementioned hard constraints plus the violations of the soft constraints (5)–(7).

The weight of constraint type (5) is the number of students without a seat, whereas the weight of constraint types (6) and (7) is fixed to 5 and 2, respectively. Hard constraints are assigned the weight 1000.

The initial solution is selected at random. That is, we create a random matrix T that satisfies constraints (1) and (4).

4.3 Neighbourhood Functions

In the CTT problem, we are dealing with the assignment of a lecture to two kinds of resources: the time periods and the rooms. Therefore, one can very intuitively define two basic neighbourhood structures which deal separately with each one of these components. We call these neighbourhoods `Time` and `Room` (or simply `T` and `R` for short) respectively.

The first neighbourhood is defined by simply changing the period assigned to a lecture of a given course to a new one which satisfies the constraints (4). A move of the `Time` type is identified by a triple of variables $\langle C, P, Q \rangle$, where C represents a course, and P and Q are the old and the new periods of the lecture, respectively.

The `Room` neighbourhood, 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.

Obviously, there are some constraints (part of the so-called *interface* constraints in [13]) for a given move m to be applicable. In detail, a `Time` move $\langle C = c_i, P = k_1, Q = k_2 \rangle$ is feasible in a given state only if in that state the course c_i has a lecture at time k_1, it has no lecture at time k_2, and the teacher of c_i is available at k_2. Instead, we consider a `Room` move $\langle C = c_i, P = k, R = r_j \rangle$ as applicable in a state if the course c_i has a lecture at time k which is assigned to a room $r_{j'}$ with $j \neq j'$.

Table 1. Features of the instances used in the experiments

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	4.61%	100.00%

Given these two basic neighbourhoods we define the neighbourhood union `Time`$\oplus$`Room` whose moves are either a `Time` or a `Room`. Conversely, the neighbourhood composition `Time`$\otimes$`Room` involves both the resources at once. For the composite neighbourhood, 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 under the constraints $C_1 = C_2 \wedge Q_1 = P_2$.

4.4 Runners and Kickers

We define eight runners, obtained equipping HC and TS with the four neighbourhoods: `Time`, `Room`, `Time`$\oplus$`Room` and `Time`$\otimes$`Room`.

We also define two kickers both based on the total composition $\odot_h$`Time`,`Room` of the basic neighbourhoods. The two kickers differ from each other in the definition of the synergic moves for the four combinations. The first one 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 also allows combination of moves on different rooms.

All the above runners and kickers are combined in various token-ring strategies, as described in the next section.

5 Experimental Results

To our knowledge, no benchmark instance for the CTT problem has been made available in the scientific community. For this reason we decided to test our algorithms with four real-world instances from the School of Engineering of our university, which will be made available through the web. 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.

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, while the columns "Conflicts" and "Occupancy" show the density of the conflict matrix, and the percentage of occupancy of the rooms ($\sum_i l_i/(m \cdot p)$), respectively. The first feature is a measure of instance size, whereas the other two are the main indicators of instance constrainedness.

Table 2. Results for the plain multi-neighbourhood HC and TS algorithms

Instance	HC(T⊕R)	HC(T⊗R)	HC(T)▷HC(R)
1	288	**285**	295
2	**18**	22	101
3	**72**	169	157
4	**140**	159	255

Instance	TS(T⊕R)	TS(T⊗R)	TS(T)▷TS(R)
1	**238**	277	434
2	**35**	175	262
3	**98**	137	488
4	**150**	150	2095

The proposed algorithms are coded in C++ and have been tested on a PC running Linux equipped with an AMD Athlon 1.5 GHz processor and 384 MB of central memory. In order to obtain a fair comparison among all algorithms, we fix an upper bound on the overall computational time (600 s per instance) of each solver during multiple trials, and we record the best value found up to that time. In this way, each algorithm can take advantage of a multi-start strategy proportionally with its speed, thus having increased chances to reach a good local minimum.

5.1 Multi-neighbourhood Search

We run the HC and TS multi-neighbourhood algorithms on the three instances with the best parameter settings found in a preliminary test phase. Namely, the tabu list is a dynamic one and the tabu tenure varies in the range 20–30. Concerning the number of idle iterations allowed, it is one million for HC and 1000 for TS.

All algorithms found a feasible solution for all trials. Concerning the objective function, the best costs found by the algorithms are summarized in Table 2, where the neighbourhood is in parentheses. The best results found by each technique are displayed in bold face.

From the results, it turns out that the HC algorithms are superior to the TS ones for three out of four instances. Concerning the comparison of neighbourhood operators, the best results are obtained by the Time⊕Room neighbourhood for both HC and TS.

Notice that the thorough exploration of Time⊗Room performed by TS does not give good results. This highlights the trade-off between the steepness of search and the computational cost.

5.2 Multi-neighbourhood Run & Kick

In this section, we evaluate the effect of $\odot_h$`Time,Room` kickers in joint action (i.e. token-ring) with the proposed local search algorithms.

We take into account three types of kicks. The first two are the best kicks with the strict and the more relaxed definition of move synergy (denoted in Tables 3 and 4 by b and b^*, respectively). In the aim of maintaining the computation time below a certain level we experiment with these kickers only with steps $h = 2$ and $h = 3$.

We compare these kicks with random kicks of length $h = 10$ and $h = 20$ (denoted in Tables 3 and 4 by r). In preliminary experiments, we have found that shorter random walks are almost always undone by the local search algorithms in token-ring alternation with the kicker. In contrast, longer walks perturb the solution too much, leading to a waste of computation time.

The results of the multi-neighbourhood Run & Kick are reported in Tables 3 and 4. In the column "Kick" is reported the length of the kick and the selection mechanism employed.

For each technique we list the best state found and the percentage of improvement obtained by Run & Kick w.r.t. the corresponding plain algorithm presented in the previous section. As before, the best results for each instance are displayed in bold face.

Comparing these results with those of the previous table, we see that the use of kickers can provide a remarkable improvement on the algorithms. In particular, kickers implementing the best kick strategy of length 2 increase the ability of the local search algorithms independently of the search technique employed. Unfortunately, the same conclusion does not hold for the best kicks of length 3. In fact, the time limit granted to the algorithms makes it possible only to perform a single best kick of this length at early stages in the search. Therefore, for instances of this size the improvement in the search made by these kicks is hidden because of their high computational cost.

Furthermore, it is possible to see that for TS the random kick strategy obtains moderate improvements in joint action with `T`$\oplus$`R` and `T`$\otimes$`R` neighbourhoods, favouring a diversification of the search. Conversely, the behaviour of the HC algorithms with this kind of kick is not uniform, and it deserves further investigation.

Concerning the influence of different synergy definitions, it is possible to see that the stricter one has a positive effect in joint action with TS, while it seems to have little or no impact with HC. In our opinion this is related to the thoroughness of neighbourhood exploration performed by TS.

Another effect of the Run & Kick strategy, which is not shown in the tables, is the improvement of algorithm robustness measured in terms of standard deviations of the results.

Table 3. Results for the HC & Kick algorithms

Instance	Kick	HC(T⊕R)		HC(T⊗R)		HC(T)▷HC(R)	
1	b_2	207	−28.1%	212	−25.6%	**200**	−32.2%
1	b_2^*	206	−28.5%	217	−23.9%	203	−31.2%
1	b_3	271	−5.9%	518	81.8%	439	48.8%
1	b_3^*	341	18.4%	515	116%	773	171%
1	r_{10}	271	−5.9%	275	−3.5%	414	30.3%
1	r_{20}	284	−1.4%	294	3.2%	440	49.2%
2	b_2	18	0.0%	21	−4.6%	27	−73.3%
2	b_2^*	18	0.0%	**17**	−22.7%	23	−77.2%
2	b_3	71	294%	67	205%	239	137%
2	b_{3^*}	79	339%	92	318%	481	376%
2	r_{10}	19	5.6%	21	−4.6%	156	54.5%
2	r_{20}	24	33.3%	19	−13.6%	182	80.2%
3	b_2	64	−11.1%	94	−44.4%	78	−50.3%
3	b_2^*	**55**	−23.6%	87	−48.5%	79	−49.7%
3	b_3	182	153%	329	94.7%	853	443%
3	b_3^*	235	226%	436	158%	1632	940%
3	r_{10}	94	30.6%	202	19.5%	206	31.2%
3	r_{20}	95	31.9%	113	−33.1%	181	15.3%
4	b_2	132	−5.71%	146	−8.18%	**113**	−55.69%
4	b_2^*	139	−0.71%	151	−5.03%	142	−44.31%
4	b_3	250	78.57%	565	255.35%	1242	387.06%
4	b_3^*	180	28.57%	3417	2049.06%	19267	7455.69%
4	r_{10}	115	−17.86%	250	57.23%	3292	1190.98%
4	r_{20}	130	−7.14%	172	8.18%	4344	1603.53%

6 Discussion and Conclusions

We have proposed a set of multi-neighbourhood search strategies to improve local search capabilities. This is only a step toward a full understanding of the capabilities of multi-neighbourhood techniques.

Our neighbourhood operators are completely general, in the sense that, given the basic neighbourhood functions, the synthesis of the proposed algorithms requires only the definition of the synergy constraint, but no further domain knowledge.

With respect to other multi-neighbourhood meta-heuristics, such as Variable Neighbourhood Search [8] and Iterated Local Search [11], we have tried to give a more general picture in which these previous (successful) proposals fit naturally.

Our software tool [4,5,6] generates automatically the code for exploration of a composite neighbourhood starting from the code for the basic ones. This is very important, from the practical point of view, in order that the test for composite techniques be very inexpensive not only in terms of design efforts, but also in terms of human programming resources.

Table 4. Results for the TS and Kick algorithms

Instance	Kick	TS(T⊕R)		TS(T⊗R)		TS(T)▷TS(R)	
1	b_2	**208**	−12.6%	214	−22.7%	210	−57.0%
1	b_2^*	208	−12.6%	210	−24.2%	226	−53.7%
1	b_3	287	20.6%	424	53.1%	347	−20.0%
1	b_3^*	273	14.7%	464	67.5%	399	−8.1%
1	r_{10}	265	11.3%	314	13.4%	546	11.9%
1	r_{20}	220	−7.6%	274	−1.1%	569	16.6%
2	b_2	**13**	−62.9%	40	−77.1%	27	−89.7%
2	b_2^*	18	−48.6%	34	−80.6%	47	−82.1%
2	3_b	82	134%	445	154%	491	87.4%
2	b_3^*	97	177%	798	356%	1703	550%
2	r_{10}	17	−51.4%	40	−77.1%	544	108%
2	r_{20}	20	−42.9%	32	−81.7%	726	177%
3	b_2	**76**	−22.5%	83	−50.9%	101	−79.3%
3	b_2^*	**78**	−20.4%	97	−42.6%	145	−70.3%
3	b_3	227	132%	312	127%	1019	109%
3	b_3^*	259	164%	476	248%	1348	176%
3	r_{10}	71	−27.6%	147	−13.0%	832	70.5%
3	r_{20}	72	−26.5%	139	−17.8%	966	98.0%
4	b_2	78	−48.00%	99	−34.00%	105	−94.99%
4	b_2^*	87	−42.00%	126	−16.00%	88	−95.80%
4	b_3	103	−31.33%	201	34.00%	1356	−35.27%
4	b_3^*	177	18.00%	2189	1359.33%	12020	473.75%
4	r_{10}	134	−10.67%	123	−18.00%	4105	95.94%
4	r_{20}	101	−32.67%	159	6.00 %	4324	106.40%

The typical way to solve CTT is by a decomposition: first schedule lectures neglecting the rooms, then assigns the rooms (see, e.g., [9]). In our framework, this would correspond to a token-ring A(Time)▷A(Room) (where A is any technique) with one single round, with the initial solution in which all lectures are in the same room. Experiments show that this choice gives much worse results than those shown in this paper.

It is worth noticing that for CTT, it is natural to compose the neighbourhoods because they are complementary, as they work on different features of the current state (the search space is not connected under them). However, preliminary results with other problems show that multi-neighbourhood search also helps for problems that have completely unrelated neighbourhoods, and thus could be solved also relying on a single neighbourhood function.

References

1. Aarts, E., Lenstra, Jan Karel.: Local Search in Combinatorial Optimization. Wiley, Chichester (1997)
2. Burke, E., Erben, W. (Eds.): Practice and Theory of Automated Timetabling III (PATAT 2000, Konstanz, Germany, August, selected papers). Lecture Notes in Computer Science, Vol. 2079. Springer-Verlag, Berlin Heidelberg New York (2001)
3. den Besten, M., Stützle, T.: Neighborhoods Revisited: An Experimental Investigation into the Effectiveness of Variable Neighborhood Descent for Scheduling. In: Pinho de Sousa, J. (Ed.): Proc. 4th Metaheuristics Int. Conf. (MIC-01) (2001) 545–550
4. Di Gaspero, L., Schaerf. A.: EasyLocal++: An Object-Oriented Framework for Flexible Design of Local Search Algorithms. Technical Report UDMI/13/2000/RR. Dipartimento di Matematica e Informatica, Università di Udine (2000). Available at http://www.diegm.uniud.it/schaerf/projects/local++
5. Di Gaspero, L., Schaerf. A.: A Case-Study for EasyLocal++: the Course Timetabling Problem. Technical Report UDMI/13/2001/RR. Dipartimento di Matematica e Informatica, Università di Udine (2001). Available at http://www.diegm.uniud.it/schaerf/projects/local++
6. Di Gaspero, L., Schaerf. A.: EasyLocal++: An Object-Oriented Framework for Flexible Design of Local Search Algorithms. Softw. – Pract. Exper. (to appear)
7. Glover, F., Laguna, M.: Tabu search. Kluwer, Dordrecht (1997)
8. Hansen, P., Mladenović, N.: An Introduction to Variable Neighbourhood Search. In: Voß, S., Martello, S., Osman, I.H., Roucairol, C. (Eds.): Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer, Dordrecht (1999) 433–458
9. Laporte, G., Desroches, S.: Examination Timetabling by Computer. Comput. Oper. Res. **11** (1984) 351–360
10. Lin, S.: Computer Solutions of the Traveling Salesman Problem. Bell Syst. Tech. J. **44** (1965) 2245–2269
11. Lourenço, H.R., Martin, O., Stützle, T.: Applying Iterated Local Search to the Permutation Flow Shop Problem. In: Glover, F., Kochenberger, G. (Eds.): Handbook of Metaheuristics. Kluwer, Dordrecht (2001) (to appear)
12. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method. In: Proc. 8th Natl Conf. Artif. Intell. (AAAI'90) AAAI Press/MIT Press, Boston, MA (1990) 17–24
13. Pesant, G., Gendreau, M.: A Constraint Programming Framework for Local Search Methods. J. Heuristics **5** (1999) 255–279
14. Pesch, E., Glover, F.: TSP Ejection Chains. Discr. Appl. Math. **76** (1997) 175–181
15. Schaerf, A.: Local Search Techniques for Large High-School Timetabling Problems. IEEE Trans. Syst. Man Cybern. **29** (1999) 368–377
16. Schaerf, A.: A Survey of Automated Timetabling. Artif. Intell. Rev. **13** (1999) 87–127
17. Selman, B., Kautz, H.A., Cohen, B.: Noise Strategies for Improving Local Search. In: Proc. 12th Natl Conf. Artif. Intell. (AAAI'94) (1994) 337–343
18. Selman, B., Levesque, H., Mitchell, D.: A New Method for Solving Hard Satisfiability Problems. In: Proc. 10th Natl Conf. Artif. Intell. (AAAI'92) (1992) 440–446
19. Vaessens, R., Aarts, E., Lenstra, J.K.: Job Shop Scheduling by Local Search. INFORMS J. Comput. **8** (1996) 302–317