J Heuristics (2011) 17: 97–118
DOI 10.1007/s10732-010-9128-0

Neighborhood analysis: a case study on curriculum-based course timetabling

Zhipeng Lü · Jin-Kao Hao · Fred Glover

Received: 30 January 2009 / Revised: 14 October 2009 / Accepted: 14 February 2010 /
Published online: 2 March 2010

Abstract In this paper, we present an in-depth analysis of neighborhood relations for local search algorithms. Using a curriculum-based course timetabling problem as a case study, we investigate the search capability of four neighborhoods based on three evaluation criteria: percentage of improving neighbors, improvement strength and search steps. This analysis shows clear correlations of the search performance of a neighborhood with these criteria and provides useful insights on the very nature of the neighborhood. This study helps understand why a neighborhood performs better than another one and why and how some neighborhoods can be favorably combined to increase their search power. This study reduces the existing gap between reporting experimental assessments of local search-based algorithms and understanding their behaviors.

Keywords Neighborhood structure · Neighborhood combination · Local search · Timetabling · Metaheuristics

Z. Lü (✉) · J.-K. Hao
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
e-mail: lu@info.univ-angers.fr

Z. Lü
e-mail: zhipeng.lui@gmail.com

J.-K. Hao
e-mail: hao@info.univ-angers.fr

F. Glover
OptTek Systems, Inc., 2241 17th Street, Boulder, CO 80302, USA
e-mail: glover@opttek.com

1 Introduction

Neighborhood search or local search is known to be a highly effective metaheuristic framework for solving a large number of constraint satisfaction and optimization problems (Hoos and Stützle 2004). For a given neighborhood and starting from an initial solution, local search attempts to improve progressively the present solution by exploring its neighborhoods. In this way, the current solution is iteratively replaced by one of its neighbors (often improving) until a specific stop criterion is satisfied.

One of the most important features of local search is thus the definition of its neighborhood. In general, good neighborhoods offer a high search capability and consequently lead to good results largely independent of the initial solution while the search performance induced by weak neighborhoods is often highly correlated to the initial solution (Papadimitriou and Steiglitz 1998). Moreover, the behavior of local search depends strongly on the characteristics of its neighborhood. For instance, some neighborhoods allow the search to obtain solution improvements in a quick and important manner, but the improvement occurs only for a limited number of iterations. On the contrary, other neighborhoods only enable small improvements, but for a long time.

In addition, if two or more neighborhoods present complementary characteristics, it is then possible and interesting to create more powerful combined neighborhoods. The advantage of such an approach was demonstrated using a tabu search strategic oscillation design in Glover et al. (1984), and additional variants of strategic oscillation for transitioning among alternative neighborhoods are discussed in Glover (1996). More recently, the metaheuristic approach called Variable Neighborhood Search in Mlandenovic and Hansen (1997) has effectively used a transition scheme that always returns to the simplest neighborhood when improvement occurs, while the transition scheme that cycles through higher levels before returning to the simplest (also studied in Glover et al. (1984)) was examined in Di Gaspero and Schaerf (2006) and elaborated more fully in the metaheuristic context in Goëfon et al. (2008).

However, one finds few studies in the literature concerning a number of important and basic questions (Johnson 2002): why does one particular neighborhood lead to better computational results than another one? what are the main characteristics of a good neighborhood? When would a combination of two or more neighborhoods be preferred to a single neighborhood and in which manner?

Without claiming to answer all these important questions, we present in this work an experimental analysis of neighborhoods. For this purpose, we introduce three evaluation criteria to characterize the search capability of a neighborhood: percentage of improving neighbors, improvement strength and search steps. As a case study, we consider the so called curriculum-based course timetabling problem (CB–CTT), which is the topic of the Second International Timetabling Competition[1] (see McCollum 2007; McCollum et al. 2008). In particular, we investigate three existing neighborhoods (with three moves called *SimpleMove*, *SimpleSwap*, *KempeMove*) from

[1] Track 3: curriculum based course timetabling, http://www.cs.qub.ac.uk/itc2007/.

the literature as well as a newly proposed neighborhood (with a move called *KempeSwap*). The analysis shows that the computational results are strongly correlated with the values and trends of the above evaluation criteria. Furthermore, the analysis sheds light on why and how some neighborhoods can be used in a combined manner.

The remaining part of this paper is organized as follows. Section 2 gives the description of the CB–CTT problem of ITC–2007. Following that, four distinct neighborhoods are described in Sect. 3. Section 4 is dedicated to the computational experimentations based on Steepest Descent (SD) method and the corresponding neighborhood analysis. In Sect. 5, we investigate whether the conclusions drawn in Sect. 4 could be expected on more advanced local search methods. Eventually in Sect. 6, conclusions are drawn.

2 Curriculum-based course timetabling

The CB-CTT problem consists of a set of n courses $C = \{c_1, c_2, \ldots, c_n\}$ to be scheduled in a set of p periods $T = \{t_1, t_2, \ldots, t_p\}$ and a set of m rooms $R = \{r_1, r_2, \ldots, r_m\}$. Each course c_i is composed of l_i same lectures to be scheduled. In the CB–CTT problem, the set of lectures of n courses must be assigned into the p periods and m rooms subject to a given set of *hard* constraints and *soft* constraints. Note that conflicts between courses for the CB-CTT problem are set according to the curricula published by the university, which is quite different from the post enrollment-based course timetabling where the course timetable is scheduled on the basis of the students' enrollment data (McCollum et al. 2008). Hard constraints must be strictly satisfied under any circumstances, while soft constraints are not necessarily satisfied but their violations should be desirably minimized. A timetabling assignment that satisfies all the following four hard constraints H_1–H_4 is called a *feasible* assignment. Then, the objective of the CB-CTT problem is to minimize the number of soft constraint violations in a feasible solution. The four hard constraints H_1–H_4 and four soft constraints S_1–S_4 are:

H1. **Lectures**: All lectures of a course must be scheduled to a distinct period and a room.
H2. **Room Occupancy**: Any two lectures cannot be assigned in the same period and the same room.
H3. **Conflicts**: Lectures of courses in the same curriculum or taught by the same teacher cannot be scheduled in the same period, i.e., any period cannot have an overlapping of students or teachers.
H4. **Availability**: If the teacher of a course is not available at a given period, then no lectures of the course can be assigned to that period.

In addition, a feasible timetable satisfying the above hard constraints incurs a penalty cost for the violations of the following four soft constraints.

S1: **Room Capacity**: For each lecture, the number of students attending the course should not be greater than the capacity of the room hosting the lecture.

S2: **Room Stability**: All lectures of a course should be scheduled in the same room. If this is impossible, the number of occupied rooms should be as few as possible.
S3: **Minimum Working Days**: The lectures of a course should be spread into the given minimum number of days.
S4: **Curriculum Compactness**: For a given curriculum a violation is counted if there is one lecture not adjacent to any other lecture belonging to the same curriculum within the same day, which means the agenda of students should be as compact as possible.

We choose a direct solution representation for simplicity reasons. A candidate solution is represented by a $p \times m$ matrix X where $x_{i,j}$ corresponds to the course label assigned at period t_i and room r_j. If there is no course assigned to period t_i and room r_j, then $x_{i,j}$ takes the value "-1". For the mathematical formulation of the CB–CTT problem, please refer to Lü and Hao (2010) for more details.

3 Neighborhoods and algorithms

3.1 Initial solution and search space

Starting from an empty timetable, our initial feasible solution is generated in a constructive way by means of a fast greedy procedure. This feasible solution is obtained by sequentially selecting one appropriate lecture of a course each time and assigning the lecture to a period and a room. In our initial solution generator, we also take into account the soft constraints by introducing a weighted cost function. We simply mention that for all the 21 competition instances, this greedy heuristic can easily obtain feasible solutions. The main ideas of this greedy heuristic are given in Lü and Hao (2010).

When a feasible assignment is reached, i.e. satisfying all the hard constraints, the local search procedure is used to reduce the number of soft constraint penalties without breaking any hard constraint. Therefore, the search space of our local search algorithm is limited to the feasible timetables, composed of the set $\mathcal{X}$ of assignment matrices for which the four hard constraints H_1–H_4 hold.

3.2 Neighborhoods

In a local search procedure, applying a move mv to a candidate solution X leads to a new solution denoted by $X \oplus mv$. Let $M(X)$ be the set of all possible moves which can be applied to X and does not create any infeasibility, then the neighborhood of X is defined by: $N(X) = \{X' \in \mathcal{X} \mid X' = X \oplus mv,\ mv \in M(X)\}$. For the CB–CTT problem, we consider four distinct *move*s denoted by *SimpleMove*, *SimpleSwap*, *KempeMove* and *KempeSwap*, leading to four neighborhoods denoted by N_1, N_2, N_3 and N_4, where only the moves producing those neighbors that do not incur any violation of the hard constraints are accepted. Let us mention that N_4 is a new neighborhood while other three ones have been proposed for solving other timetabling problems in the literature (Burke and Newall 2004; Burke et al. 2006; Chiarandini et al. 2006; Lewis 2008; Schaerf 1999).

Neighborhood N_1 A move of type *SimpleMove* consists simply in moving one lecture of course $x_{i,j}$ at period t_i and room r_j to a free position (period $t_{i'}$ and room $r_{j'}$) where $i' \neq i$ or $j' \neq j$. After this move, $x_{i',j'} = x_{i,j}$ and $x_{i,j} = -1$. $x_{i,j} = -1$ means that there is no any lecture scheduled at period t_i and room r_j. The size of neighborhood N_1 is bounded by $O(l \cdot (p \cdot m - l))$ where $l = \sum_{i=1}^{n} l_i$ because there are l lectures and the total number of free positions is bounded by $O(p \cdot m - l)$. Note that if the total number of lectures l is equal to that of the available positions $(m \cdot p)$, the size of this neighborhood is zero.

Neighborhood N_2 A *SimpleSwap* move consists in exchanging the hosting periods and rooms assigned to two lectures of different courses. Applying the *SimpleSwap* move to two different courses $x_{i,j}$ and $x_{i',j'}$ for the solution X consists in assigning the value of $x_{i,j}$ to $x_{i',j'}$ and inversely the value of $x_{i',j'}$ to $x_{i,j}$. Since there are l lectures, the size of N_2 is bounded by $O(l^2)$.

Neighborhood N_3 A move of type *KempeMove* is defined by Kempe chain interchanges. For the CB–CTT problem, a candidate solution X can be considered as a graph G where nodes are lectures and edges connect lectures with students or teacher in common. Note that an edge exists between any two lectures belonging to a same course. A Kempe chain is defined as a set of lectures that form a connected component in the subset of lectures that belong to two distinct periods. Let K be a Kempe chain with respect to two periods t_i and t_j and L_i (L_j) be the set of lectures in period t_i (t_j), a Kempe chain interchange produces an assignment by replacing L_i with $(L_i \backslash K) \cup (L_j \cap K)$ and L_j with $(L_j \backslash K) \cup (L_i \cap K)$ (Chiarandini et al. 2006). Notice that in each *KempeMove*, at least three lectures are involved, i.e., $|K| \geq 3$.

Once lectures have been scheduled to a period, the room assignment can be solved by an exact bipartite matching algorithm (Rossi-Doria et al. 2002; Sedgewick 1988). Since *KempeMove* can be considered as moving one lecture and afterward several other related lectures in the Kempe chain being moved, the size of N_3 is bounded by $O(l \cdot p)$.

For example, Fig. 1 depicts a subset of lectures deduced by two periods t_i and t_j with each room having one lecture. In this small example, there are four Kempe chains: $K_1 = \{c_1\}$, $K_2 = \{c_5, c_{11}\}$, $K_3 = \{c_2, c_3, c_7, c_8, c_{10}\}$ and $K_4 = \{c_4, c_6, c_9, c_{12}\}$. However, only K_4 can produce a feasible *KempeMove* since other three are forbidden. For K_1 and K_2, the number of involved lectures are less than 3. For K_3, it is not able to lead to a feasible solution, since interchanging $\{c_2, c_3\}$ and $\{c_7, c_8, c_{10}\}$ makes the number of lectures in period t_i greater than the total number of the available rooms. We call this restriction the so called *room allocation violation*. Indeed, the asymmetry property of lecture numbers largely restricts the number of acceptable candidate solutions for this neighborhood and constitutes its weakness. Compared with *KempeMove* move, the next neighborhood is much more flexible and will avoid this limitation.

Neighborhood N_4 In the *KempeMove* (i.e. N_3), only one connected component of a subset of lectures is considered concerning two distinct periods. We intro-

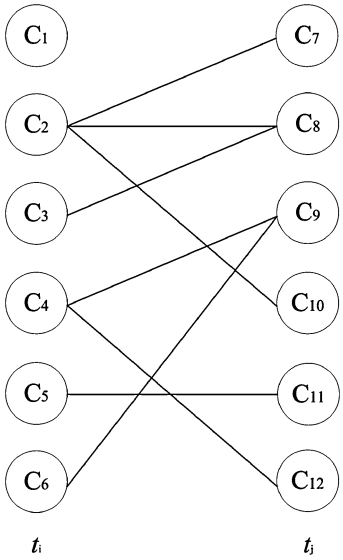

Fig. 1 Kempe chain illustrations

duce now a new move called *KempeSwap* which consists in interchanging the lectures of *two* distinct Kempe chains. Formally, let K_1 and K_2 be two Kempe chains in the subgraph with respect to two periods t_i and t_j, a *KempeSwap* produces an assignment by replacing L_i with $(L_i \backslash (K_1 \cup K_2)) \cup (L_j \cap (K_1 \cup K_2))$ and L_j with $(L_j \backslash (K_1 \cup K_2)) \cup (L_i \cap (K_1 \cup K_2))$. It is noteworthy to notice that our double Kempe chains interchange can be considered as a generalization of the single Kempe chain interchange known in the literature (Casey and Thompson 2003; Chiarandini et al. 2006; Côté et al. 2005; Merlot et al. 2003).

For instance, in Fig. 1, interchanging lectures $\{c_1, c_2, c_3\}$ and $\{c_7, c_8, c_{10}\}$ is a move of *KempeSwap* which concerns two distinct connected components K_1 and K_3. Feasible *KempeSwap* moves also include interchanging K_2 and K_4. Note that the room allocation procedure after period interchange is the same as *KempeMove*. For each move of N_4, at least three lectures are involved too, i.e., $|K_1| + |K_2| \geq 3$. Since *KempeSwap* can be considered as an extended version of *SimpleSwap* (i.e., swapping two lectures), the size of neighborhood N_4 is bounded by $O(l^2)$.

As mentioned above, except for neighborhood N_4, the other three neighborhoods have been proposed in the previous literature (Chiarandini et al. 2006). However, we will show in the following sections (Sects. 4 and 5) that our newly proposed neighborhood N_4 is more powerful and explain why this is the case.

3.3 Neighborhood combinations

In order to increase the search capability of single neighborhoods, it has become a popular practice to combine two or more different neighborhoods, especially when those neighborhoods have complementary characteristics. In fact, there are many ways for combining different neighborhoods. In this paper we focus on two of

them: neighborhood union and token-ring search (see Di Gaspero and Schaerf 2006; Glover et al. 1984).

In neighborhood union, at each iteration the neighborhood structure includes all the moves of two different neighborhoods. If we consider two different neighborhoods N_a and N_b, then the neighborhood union of these two neighborhoods can be represented as $N_a \cup N_b$.

In token-ring search, different neighborhoods are consecutively used on the local optimum of the previous neighborhood until no improvement is possible. More precisely, we start one local search procedure with one neighborhood. When the search ends with its best local optimum, we restart the local search from this local optimum, but with the other neighborhood. This process is repeated until no improvement is possible (Di Gaspero and Schaerf 2006; Glover et al. 1984). The token-ring search of two neighborhoods can be denoted as $N_a \rightarrow N_b$ (starting from N_a) or $N_b \rightarrow N_a$ (starting from N_b).

If there are more than two neighborhoods and we want to combine them in a more meaningful way, it is possible to produce more complex neighborhood combinations. For example, neighborhood combination $(N_a \cup N_b) \rightarrow N_c$ denotes that neighborhood union of N_a and N_b is combined with neighborhood N_c in a token-ring way and the search starts from $N_a \cup N_b$.

3.4 Local search algorithms

In this paper, a study of the behaviors of different neighborhoods and their combinations is conducted. For this purpose, we employ a steepest descent (SD) algorithm in Papadimitriou and Steiglitz (1998). This choice can be justified by the fact that the SD algorithm is completely parameter free, and thus it allows a direct comparison of different neighborhoods without bias. Notice that SD is also the basic search strategy commonly used in several advanced metaheuristics, such as Tabu Search in Glover and Laguna (1997), Variable Neighborhood Search in Hansen and Mladenovi (2001), Mlandenovic and Hansen (1997), Iterated Local Search in Lourenco et al. (2003) and so on.

From a feasible timetable $X \in \mathcal{X}$, the SD algorithm repeatedly replaces the current solution X by a *best improving* solution in its neighborhood until no improving neighbor exists. The reason for not using a First Improvement strategy (FI) lies in the fact that the SD algorithm generally obtains slightly better results than the FI algorithm according to our experience. Moreover, in our implementations, the computational cost for an SD move is practically the same as for an FI move. This is possible thanks to an incremental evaluation of neighborhood moves, enabling the fast identification of the move. The main idea of this incremental evaluation technique is to maintain in a special data structure the *move value* for each possible move of the current solution. Each time a move is carried out, the elements of this data structure affected by the move are updated accordingly.

In addition, in order to further verify whether similar conclusions can be expected with more advanced local search-based metaheuristics, we implement three other metaheuristic algorithms: Tabu Search (TS), Iterated Local Search (ILS) and Adaptive Tabu Search (ATS). The details of these algorithms are described in Sect. 5.

4 Experimental results and analysis

In this section, we first test the SD algorithm on a set of 14 competition instances for the four neighborhoods $N_1 \sim N_4$ (Sect. 3.2) and their various combinations (Sect. 3.3). Based on the computational results, we carried out our experiments to analyze the search capability of single neighborhoods and their combinations in terms of three criteria: percentage of improving neighbors, improvement strength and search steps. Following that, some concluding remarks are presented.

4.1 Computational results based on SD algorithm

In order to assess the practical performance of the four neighborhoods and their different combinations, we apply the SD algorithm with each of the four neighborhoods to solve the 14 competition instances. The main features of these instances are listed in Table 1. The last two columns denoted by *occupancy* and *conflicts* represent the percentage of occupancy of rooms (denoted by $l/(p \cdot m)$) and the density of the conflict matrix (denoted by $2 \cdot n_e / l \cdot (l-1)$ where n_e represents the total number of edges connecting two conflicting lectures), respectively.

The average soft costs for neighborhoods $N_1 \sim N_4$ over 50 independent runs are given in Table 2 and the average CPU time are given in brackets (best results for each instance are indicated in bold). Note that these results are all rounded up. From Table 2, it is easily observed that neighborhood N_4 outperforms all the others in terms of solution quality. When comparing the three neighborhoods N_1, N_2 and N_3 with each other, one finds that the average soft costs of N_1 are the best and those of N_2 are the worst. We performed a 95% confidence t-test for each pair of neighborhoods to compare the results of Table 2. Except some rare cases, the observed differences are statistically significant and the dominance of N_4 is confirmed. We can confirm that the search capability of these four neighborhoods with respect to solution quality can be ranked as follows: $N_4 > N_1 \approx N_3 > N_2$.

Table 1 Features of the 14 competition instances

Instances	n	m	p	l	Occupancy	Conflicts
comp01	30	6	30	160	88.89%	13.2%
comp02	82	16	25	283	70.75%	7.97%
comp03	76	12	25	251	62.75%	8.17%
comp04	79	18	25	286	63.56%	5.42%
comp05	54	9	36	152	46.91%	21.7%
comp06	108	18	25	361	80.22%	5.24%
comp07	131	20	25	434	86.80%	4.48%
comp08	86	18	25	324	72.00%	4.52%
comp09	76	18	25	279	62.00%	6.64%
comp10	115	18	25	370	82.22%	5.3%
comp11	30	5	45	162	72.00%	13.8%
comp12	88	11	36	218	55.05%	13.9%
comp13	82	19	25	308	64.84%	5.16%
comp14	85	17	25	275	64.71%	6.87%

Table 2 Average soft costs for N_1 to N_4 obtained by the SD algorithm over 50 independent runs

Instances	$\bar{f}$			
	N_1	N_2	N_3	N_4
comp01	42 (0.04)	33 (0.05)	49 (0.03)	**24** (0.12)
comp02	194 (0.39)	228 (0.17)	204 (0.37)	**143** (1.42)
comp03	217 (0.37)	248 (0.20)	245 (0.28)	**193** (1.09)
comp04	153 (0.71)	199 (0.37)	194 (0.61)	**132** (3.45)
comp05	1016 (0.25)	995 (0.17)	847 (0.81)	**684** (0.38)
comp06	207 (0.70)	260 (0.36)	255 (0.65)	**158** (4.56)
comp07	203 (1.07)	247 (0.63)	230 (1.27)	**140** (8.23)
comp08	154 (0.70)	205 (0.28)	185 (0.63)	**139** (3.22)
comp09	238 (0.43)	273 (0.19)	244 (0.43)	**193** (2.01)
comp10	195 (0.82)	250 (0.44)	249 (0.88)	**145** (5.12)
comp11	16 (0.07)	16 (0.06)	25 (0.03)	**9** (0.11)
comp12	807 (0.47)	874 (0.31)	885 (1.61)	**746** (0.54)
comp13	197 (0.68)	233 (0.38)	224 (0.65)	**151** (3.68)
comp14	180 (0.46)	213 (0.23)	206 (0.34)	**151** (1.23)

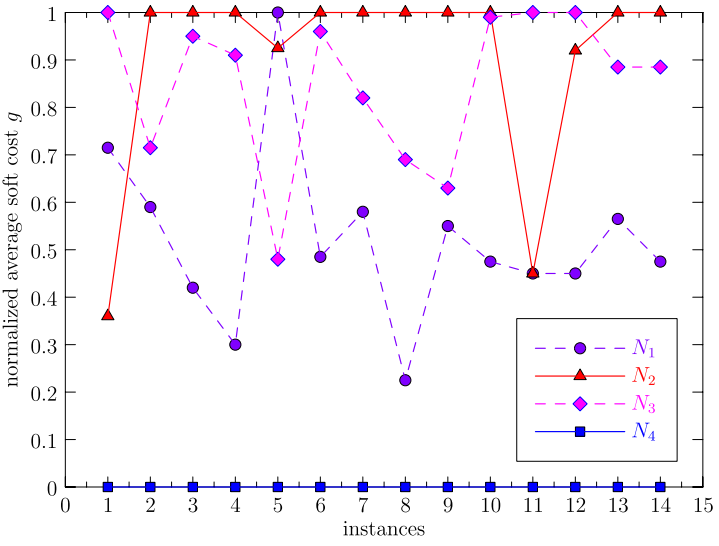

Fig. 2 Average solution quality comparisons for the SD algorithm over 50 runs

When it comes to the average CPU time, it is clear that N_4 costs more than others. This can be explained by the fact that neighborhood N_4 involves much more neighborhood moves than N_3 and the neighborhood move evaluation is more time-consuming than N_1 and N_2.

Figure 2 shows the comparisons of the *normalized average costs* for the SD algorithm over 50 independent runs. For each instance, the normalized average soft

Table 3 Average soft costs for various neighborhood combinations

Instance	Average soft costs						
	$N_1 \cup N_2$	$N_3 \cup N_4$	$N_1 \cup N_4$	$N_1 \cup N_3 \cup N_4$	$N_1 \rightarrow N_4$	$N_2 \rightarrow N_4$	$(N_1 \cup N_2) \rightarrow N_4$
comp01	31 (0.1)	23 (0.1)	19 (0.2)	***18*** (***0.2***)	24 (0.1)	20 (0.1)	19 (0.1)
comp02	186 (0.4)	143 (1.8)	136 (2.2)	135 (2.5)	132 (1.5)	140 (2.1)	***122*** (***1.6***)
comp03	210 (0.4)	187 (1.2)	177 (1.8)	172 (2.2)	173 (1.0)	184 (1.1)	***170*** (***1.0***)
comp04	152 (0.7)	131 (3.5)	116 (6.5)	109 (7.4)	***103*** (***2.7***)	121 (2.9)	105 (3.1)
comp05	871 (0.4)	627 (0.4)	591 (0.5)	***547*** (***0.6***)	574 (0.8)	609 (0.6)	580 (0.8)
comp06	197 (0.8)	162 (4.7)	151 (8.9)	150 (8.9)	***139*** (***3.3***)	155 (4.4)	141 (3.1)
comp07	190 (1.2)	141 (8.4)	123 (16)	114 (17)	***113*** (***5.2***)	126 (7.1)	114 (5.0)
comp08	154 (0.7)	129 (3.4)	112 (7.3)	113 (7.7)	***103*** (***2.9***)	119 (3.4)	105 (2.5)
comp09	231 (0.5)	189 (2.1)	182 (3.0)	183 (3.5)	***176*** (***1.8***)	189 (1.9)	177 (1.8)
comp10	186 (0.9)	147 (5.3)	128 (8.8)	128 (9.5)	***118*** (***3.3***)	130 (4.7)	127 (3.0)
comp11	11 (0.1)	11 (0.1)	***6*** (***0.2***)	***6*** (***0.2***)	9 (0.1)	8 (0.1)	7 (0.1)
comp12	774 (0.5)	743 (0.5)	684 (0.8)	663 (0.9)	***639*** (***1.1***)	685 (1.1)	668 (1.1)
comp13	186 (0.8)	151 (3.9)	135 (7.5)	137 (7.8)	***131*** (***2.7***)	145 (3.6)	***131*** (***2.7***)
comp14	175 (0.5)	156 (1.3)	132 (2.6)	139 (2.6)	125 (1.5)	148 (1.4)	***121*** (***1.6***)

cost g is represented as $g = (f - f_{\min})/(f_{\max} - f_{\min})$, where f is the original average soft cost while $f_{\max}$ and $f_{\min}$ respectively denote the worst and the best average soft costs obtained by the four neighborhoods. According to its definition, the value of the normalized soft cost g lies in the interval [0, 1]. It is obvious that the modified cost function for the best neighborhood is equal to 0 while for the worst it is equal to 1. Figure 2 discloses that the SD algorithm with N_4 performs much better than $N_1 \sim N_3$ in terms of the average cost.

We now consider the performance of several combined neighborhoods. In fact, the number of possible combinations to be considered is extremely large and thus we attempt to limit the combinations to be examined by using the results mentioned above. According to the definitions of neighborhood structures, it is obvious that N_1 and N_2 are basic neighborhoods and N_3 and N_4 are advanced ones. From the computational results above, one observes that N_3 is worse than N_1 in terms of solution quality yet requires a CPU time similar to that of N_1. This fact convinces us that N_3 is a poor neighborhood. On the other hand, N_4 is a good neighborhood in terms of solution quality. Therefore, we focus on the different combinations of N_4 with others, especially N_1. According to the above analysis and our experience, we consider the following typical neighborhood combinations: $N_1 \cup N_2$, $N_3 \cup N_4$, $N_1 \cup N_4$, $N_1 \cup N_3 \cup N_4$, $N_1 \rightarrow N_4$, $N_2 \rightarrow N_4$ and $(N_1 \cup N_2) \rightarrow N_4$.

We run the SD algorithm with these neighborhood combinations on the set of 14 competition instances. Table 3 shows the average soft costs over 50 independent runs for different neighborhood combinations. One finds that for the four ways of neighborhood union, $N_1 \cup N_4$ and $N_1 \cup N_3 \cup N_4$ produce much better results than $N_1 \cup N_2$ and $N_3 \cup N_4$. However, in $N_1 \cup N_3 \cup N_4$, the introduction of N_3 does not contribute much to the neighborhood union $N_1 \cup N_4$, i.e., there is only 1.8% improvement in

terms of the average solution quality. This further implies that N_3 is not a value-added neighborhood for this problem.

With respect to the token-ring search combinations, one observes that $N_1 \rightarrow N_4$ performs much better than $N_2 \rightarrow N_4$, even better than $(N_1 \cup N_2) \rightarrow N_4$. Therefore, one of the most promising ways for token ring search of the four neighborhoods is probably $N_1 \rightarrow N_4$.

We now directly compare the elite neighborhood union $(N_1 \cup N_4)$ and token-ring search $(N_1 \rightarrow N_4)$. For this purpose, we performed a 95% confidence t-test to compare these two elite neighborhood combinations and found that for 11 out of the 14 instances (except for *comp02*, *comp11* and *comp13*), the computational results obtained by $N_1 \rightarrow N_4$ are significantly better than the ones with $N_1 \cup N_4$.

Finally, we have tested several other token-ring combinations using N_3 as a source or destination, such as $N_1 \rightarrow N_3$, $N_3 \rightarrow N_4$ and $N_2 \rightarrow N_3$. We observed without surprise that these token-ring combinations produce worse results than those using N_4. This can be explained by the fact that N_4 has a dominant performance over N_3 as already demonstrated in several aforementioned experiments.

4.2 Neighborhood analysis

The above computational results show that the proposed neighborhood N_4 performs much better than other three in terms of solution quality. As for the various combinations of different neighborhoods, it is clear that the token-ring combination of the two neighborhoods N_1 and N_4 produces lower soft costs than other combinations. In this section, we attempt to explain what causes the effectiveness of a single neighborhood and a certain combination of different neighborhoods. For this purpose, we introduce three evaluation criteria to characterize the search capacity of different neighborhoods: *percentage of improving neighbors*, *improvement strength* and *search steps*.

4.2.1 Evaluation criteria and experimental protocol

For a candidate solution X, a given neighborhood function $N : \mathcal{X} \rightarrow 2^{\mathcal{X}}$ and a neighborhood solution $X' \in N(X)$, define $\Delta f = f(X') - f(X)$, these criteria are then defined as follows.

- *Improving neighbors* $I(X)$: the set of the improving neighbors in the neighborhood $N(X)$, i.e. $I(X) = \{X' \in N(X) \mid \Delta f < 0\}$. Therefore, the *percentage of improving neighbors* is defined as $|I(X)|/|N(X)| \times 100$.
- *Improvement strength* Δf^*: the cost variation between the current solution X and a best improving neighbor, i.e., $\Delta f^* = \max\{|\Delta f| : \Delta f \in I(X)\}$.
- *Search steps*: the search steps of N is defined as the number of iterations that the SD algorithm can run to reach a local optimum.

We argue that good neighborhoods should have one or more of these features: high percentage of improving neighbors (for more improvement possibilities), strong

improvement strength (for important improvements) and long search steps (for long term improvements).

To calculate the values of each criterion, 50 independent runs of the SD algorithm with a given neighborhood are carried out for solving each problem instance. For each run, data corresponding to the above evaluation criteria are calculated; $I(X)$ and Δf^* values are collected at each iteration while *search steps* is simply the iteration number when SD stops. All the reported results correspond to the average of these 50 independent runs.

4.2.2 Search capability of different neighborhoods

The first experiment aims to evaluate and compare the performance of the four neighborhoods $N_1 \sim N_4$ (see Sect. 3.2) using the above three criteria. The results are based on the largest instance comp07 (very similar results are observed for other instances). Figure 3 shows the percentage of improving neighbors for N_1 to N_4, evolving with the local search iterations.

Figure 3 shows that N_1 and N_2 have quite similar evolving trends in terms of the percentage of improving neighbors, so do N_3 and N_4. At the beginning of the local search, the percentage of improving neighbors is above 70% for N_3 and N_4, while it is only approximately 2.2% for N_1 and 0.5% for N_2. In other words, N_3 and N_4 offer many more opportunities to find improving neighbors during the first iterations of the search (first 10 iterations for this particular instance).

On the other hand, compared with N_2 and N_3 respectively, there exist long tails for the percentage of improving neighbors for N_1 and N_4, meaning that they allow the descent algorithm to run a large number of iterations. This property is another important sign for good neighborhoods. It should be clear now that N_4 has not only the

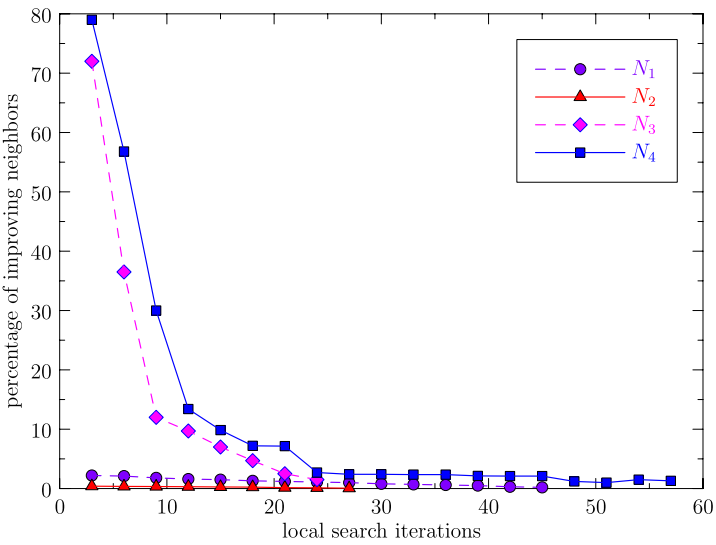

Fig. 3 Percentage of improving neighbors evolving with iterations for N_1 to N_4

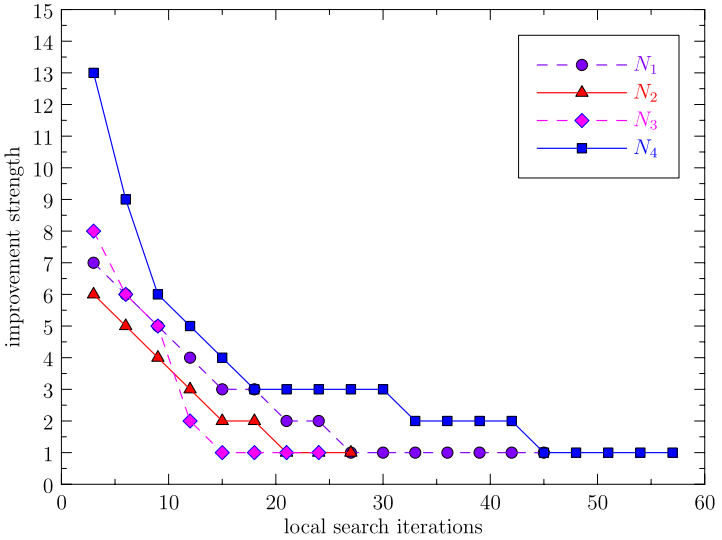

Fig. 4 Improvement strength Δf^* evolving with iterations for N_1 to N_4

Table 4 Performance level of different neighborhoods on the given criteria

Criteria	N_1	N_2	N_3	N_4
Improv. neighbors	poor	poor	**good**	**good**
Search steps	**good**	poor	poor	**good**
Improv. strength	poor	poor	poor	**good**

largest percentage of improving neighbors but also the greatest number of iterations, while N_1 continues for many iterations but with very small percentage of improving neighbors and N_3 has large percentage of improving neighbors at the very beginning of the search but its number of iterations is rather small. N_2 performs very poorly for both two criteria.

We then evaluate the four neighborhoods using the *improvement strength* criterion (Δf^*). Figure 4 shows how Δf^* of each neighborhood evolves with the local search iterations. Once again, one observes that at the beginning of the search, the improvement strength of N_4 is much stronger than others, which matches well with the trend of the percentage of improving neighbors.

In order to have a better understanding of the performance of these four neighborhoods on the three given criteria, we illustrate in Table 4 the performance level for each neighborhood-criterion pair. One observes that N_4 performs well on all the three criteria, while N_1 and N_3 only performs well on one criterion. As expected, N_2 performs very poorly on all the criteria, which explains the computational results reported in Table 2.

Considering these observations, the following conclusions can be formulated.

1. Neighborhood N_4 offers a higher percentage of improving neighbors, and greater improvement strength than the other three neighborhoods during the local search

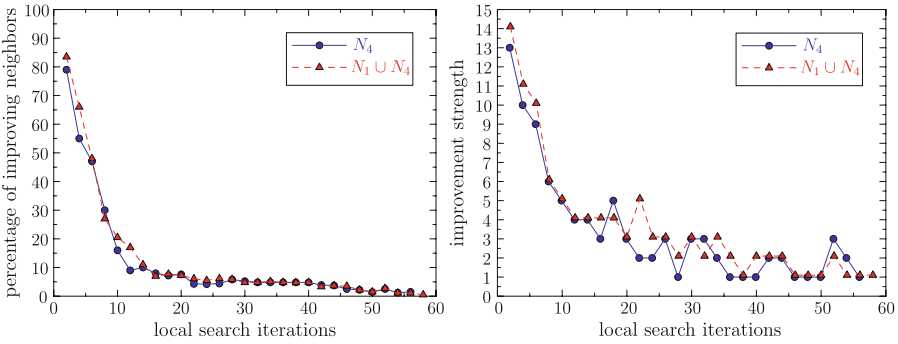

Fig. 5 Percentage of improving neighbors and improvement strength for $N_1 \cup N_4$ and N_4

iterations. As a result, and at least during the first iterations, N_4 provides a quick and effective solution process.

2. Neighborhood N_1 offers improving neighbors (with weaker improvement strengths) for a larger number of iterations than N_2 and N_3. Consequently, local search can continue for a longer time with N_1.
3. Although neighborhood N_3 offers a larger percentage of improving neighbors during the first iterations, its improvements quickly disappear, limiting its search capability.
4. Neighborhood N_2 performs quite poorly on all the three criteria and thus is not a good neighborhood.

4.2.3 Combinations of multi-neighborhoods

In this section, we turn our attention to neighborhood combinations and aim at analyzing their computational results reported in Table 3 in terms of the three proposed criteria. According to the results in Table 3, one observes that the neighborhood union of the two elite neighborhoods N_1 and N_4 ($N_1 \cup N_4$) obtains much better results than $N_1 \cup N_2$ and $N_3 \cup N_4$. On the other hand, the token-ring search of these two neighborhood ($N_1 \rightarrow N_4$) performs much better than $N_2 \rightarrow N_4$, even comparable with $(N_1 \cup N_2) \rightarrow N_4$. These results prompt us to focus on investigating in this section the two representative neighborhood combinations: $N_1 \cup N_4$ and $N_1 \rightarrow N_4$.

Moreover, one finds that $N_1 \cup N_4$ produces slightly better results than N_4, while $N_1 \rightarrow N_4$ obtains much better results than not only N_4 but also $N_1 \cup N_4$. In this section, we attempt to show evidence for these phenomena in terms of the three evaluation criteria.

At first, we investigate why the neighborhood union $N_1 \cup N_4$ performs only slightly better than N_4. To answer this question, we observe the influence of the advanced neighborhood N_4 over the combined neighborhood $N_1 \cup N_4$. Figure 5 shows the percentage of improving neighbors (left) and improvement strength (right) for neighborhoods $N_1 \cup N_4$ and N_4, evolving with local search iterations. One finds that both the percentage of improving neighbors and the improvement strength with $N_1 \cup N_4$ and N_4 evolve in quite similar ways, showing that N_4 plays a dominating

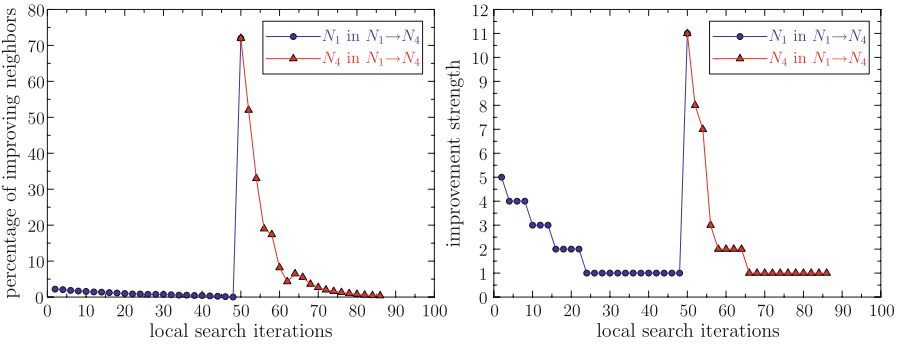

Fig. 6 Percentage of improving neighbors and improvement strength for $N_1 \rightarrow N_4$

role in this union neighborhood. This is why N_4 and $N_1 \cup N_4$ lead to very similar results.

As for the token-ring search of N_1 and N_4 ($N_1 \rightarrow N_4$), one observes that $N_1 \rightarrow N_4$ performs much better than $N_1 \cup N_4$. Figure 6 shows the percentage of improving neighbors (left) and improvement strength (right) for neighborhood $N_1 \rightarrow N_4$. First, it is important to notice that in this case, the local optimum obtained with neighborhood N_1 can be further improved with neighborhood N_4 for a relatively large number of iterations. Moreover, we can see from Fig. 6 that at the beginning of the search for N_4 in $N_1 \rightarrow N_4$, the percentage of improving neighbors is approximately 74% and the improvement strength is rather strong. That is to say, the local minimum of N_1 is typically not a local minimum of N_4 and thus the solution quality can be further improved. This phenomenon constitutes an important explanation for the excellent performance of $N_1 \rightarrow N_4$.

We should mention that it is possible and interesting to combine the two neighborhoods N_1 and N_4 in another token-ring way: starting the search from the advanced neighborhood N_4 ($N_4 \rightarrow N_1$). Although we did not report the results of this combination in this paper, we confirm that $N_4 \rightarrow N_1$ does produce quite similar results to $N_1 \rightarrow N_4$ in terms of solution quality. This can be explained by the fact that the two neighborhoods N_1 and N_4 are alternately and repeatedly used until no improvement is possible.

To summarize, the analysis of this last subsection confirms that:

1. Neighborhood union is not an appropriate way for combining N_1 and N_4 because of the dominance of N_4 in the neighborhood $N_1 \cup N_4$.
2. Token-ring search is a better strategy for combining N_1 and N_4 due to their complementary characteristics.

5 Extensions to more advanced metaheuristics

5.1 Advanced metaheuristics

In the above section, we carried out a series of computational experimentations and a detailed analysis to show and explain the performance of the four neighborhoods and

their different combinations in the SD algorithm. One may wonder whether we can expect the same results with other advanced metaheuristics. In this section, we try to answer this important question. For this purpose, we implemented three metaheuristic algorithms: Tabu Search (TS) (Glover and Laguna 1997), Iterated Local Search (ILS) (Lourenco et al. 2003) and Adaptive Tabu Search (ATS) (Lü and Hao 2010). A brief overview of these three algorithms is given below.

The TS algorithm is a simple version of TS with a self adaptive tabu list. Within TS, a *tabu list* is introduced to forbid the previously visited solutions to be revisited. In our TS algorithm, when moving one lecture from one position (period-room pair) to another (using N_1 or N_2), or from one period to another (using N_3 or N_4), this lecture is declared tabu and cannot be moved back to the previous position (for N_1 or N_2) or period (for N_3 or N_4) for a certain number of iterations. In our experiments, TS is applied to a token-ring search of two neighborhoods. Algorithm 1 gives a brief description of our TS algorithm based on a token-ring search of neighborhoods N_a and N_b. Each TS phase stops when its best solution cannot be improved within a given number θ of moves that we call the depth of TS. If there is only one neighborhood, we can just omit line 6 in Algorithm 1. Interested readers are referred to Lü and Hao (2010) for more details.

Algorithm 1 Tabu Search Algorithm: TS(X_0,θ)

1: **Input:** $X_0 \leftarrow$ the feasible initial solution; $\theta \leftarrow$ *the depth of TS*
2: **Output:** $X^* \leftarrow$ the best feasible solution found so far
3: $X^* \leftarrow X_0$
4: **repeat**
5: $\quad X' \leftarrow TS_{N_a}(X_0)$ based on neighborhood N_a with *depth of TS* equal to θ
6: $\quad X' \leftarrow TS_{N_b}(X')$ based on neighborhood N_b with *depth of TS* equal to θ
7: $\quad$ **if** X' is better than X^* **then**
8: $\quad\quad X^* \leftarrow X'$
9: $\quad$ **end if**
10: $\quad X_0 \leftarrow X'$
11: **until** (stop condition is met)

Our ILS algorithm takes the SD algorithm as the local search procedure and uses a Critical Element-Guided Perturbation (CEGP) operator to jump out of the local optima trap. The operator consists of identifying critical lectures by scoring all lectures according to their contribution to the total soft constraints and then adaptively perturbing the solution using the highly scored lectures. Algorithm 2 is a brief description of our CEGP-based ILS algorithm and interested readers are referred to Lü and Hao (2009) for more details.

Our ATS algorithm is a dynamic combination of the above TS and ILS algorithms. Specifically, the ATS algorithm is a reinforced ILS algorithm that replaces the SD algorithm by the above TS algorithm. Two additional adaptive procedures are also used to control the *depth* of TS and the perturbation strength of ILS. Interested readers are referred to Lü and Hao (2010) for complete details of this ATS algorithm.

Algorithm 2 Iterated Local Search Algorithm: ILS(X_0)

1: **Input:** $X_0 \leftarrow$ the feasible initial solution
2: **Output:** $X^* \leftarrow$ the best feasible solution found so far
3: $X' \leftarrow \text{SD}(X_0)$
4: $X^* = X'$
5: **repeat**
6: Score all lectures of X' according to their contribution to the total soft constraints
7: Randomly select a certain number η (perturbation strength) of highly scored lectures to be perturbed
8: $X'' \leftarrow$ Critical Element-Guided Perturbation Operator(X')
9: $X^{*\prime} \leftarrow \text{SD}(X'')$
10: **if** $X^{*\prime}$ is better than X^* **then**
11: $X^* = X^{*\prime}$
12: **end if**
13: $X' \leftarrow$ Acceptance Criterion($X^{*\prime}$,X^*)
14: **until** stop condition met

To make the comparison as fair as possible, all these three algorithms follow the same stop conditions, i.e., the ITC-2007 competition timeout. On our computer with 3.4 GHz CPU and 2 GB Memory, this corresponds to 390 seconds.

5.2 Computational results using advanced metaheuristics

In what follows, we focus on the computational results of the algorithms TS, ILS and ATS on the 14 competition instances. We first compare the average soft costs for different neighborhoods and neighborhood combinations. We consider here the four neighborhoods $N_1 \sim N_4$ and the two representative combined neighborhoods $N_1 \cup N_4$ and $N_1 \rightarrow N_4$.

Figures 7 to 9 show the comparisons of the *average costs* respectively for TS, ILS and ATS algorithms over 50 independent runs. In order to clearly distinguish among different neighborhoods and their combinations, we use again the normalized soft cost function to present these three graphs.

From Figs. 7 to 9, one observes that these advanced metaheuristic algorithms with N_4 perform much better than $N_1 \sim N_3$ in terms of the average cost, which coincides with the results obtained by the SD algorithm. In order to confirm this conclusion, we performed a 95% confidence t-test to compare N_4 with $N_1 \sim N_3$ respectively on TS, ILS and ATS algorithms and found that for at least 11 out of the 14 instances, the computational results obtained by N_4 are statistically better than the ones obtained by any of other three neighborhoods in any of the three algorithms.

However, one exception is that N_3 is even worse than N_2 for all the three metaheuristic algorithms, which further indicates that the single Kempe chain neighborhood N_3 is not a good one for this specific problem. We argue that the poor performance of N_3 might be caused by the *room allocation violation* restriction previously mentioned in Sect. 3.2.

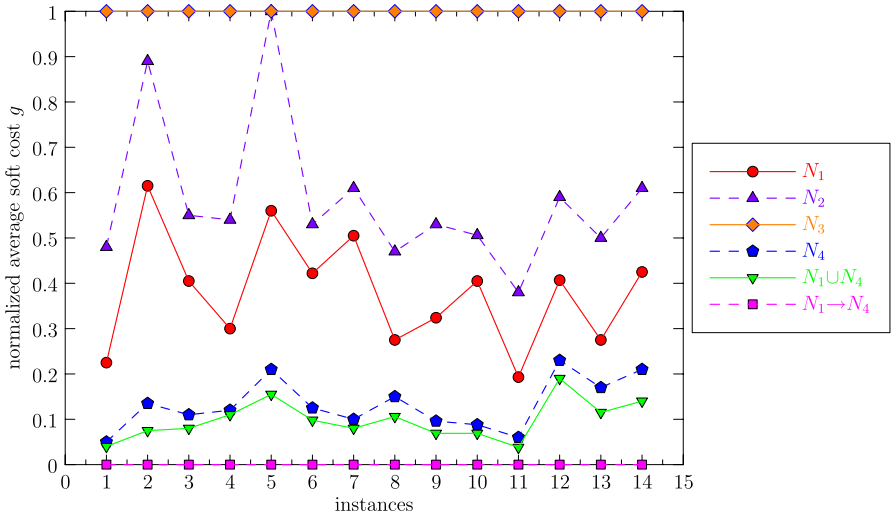

Fig. 7 Average solution quality comparisons for the TS algorithm over 50 runs

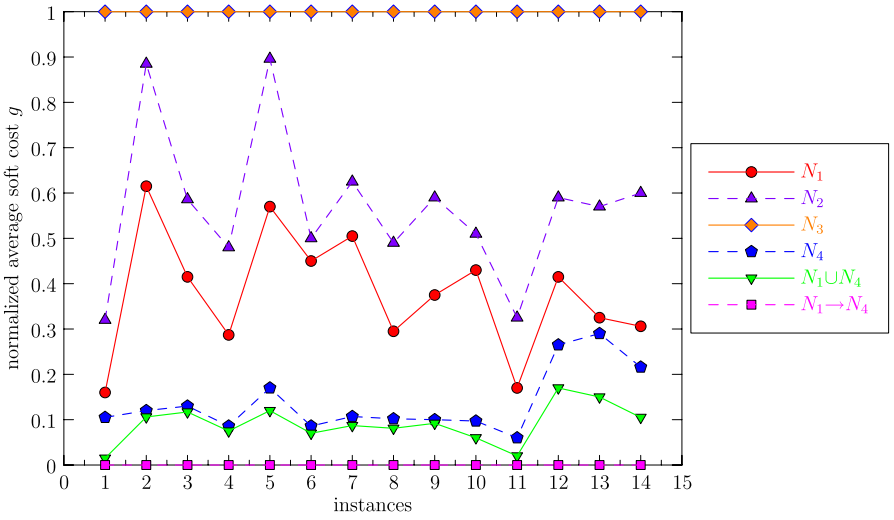

Fig. 8 Average solution quality comparisons for the ILS algorithm over 50 runs

For neighborhood combinations, the token-ring search of two complementary neighborhoods N_1 and N_4 ($N_1 \rightarrow N_4$) produces much better results than both the single neighborhoods $N_1 \sim N_4$ and the neighborhood union $N_1 \cup N_4$, which perfectly coincides with the results obtained by the SD algorithm. Although the results of other five neighborhood combinations ($N_1 \cup N_2$, $N_3 \cup N_4$, $N_1 \cup N_3 \cup N_4$, $N_2 \rightarrow N_4$ and $(N_1 \cup N_2) \rightarrow N_4$) are not reported here, their results are consistent with those for the SD algorithm.

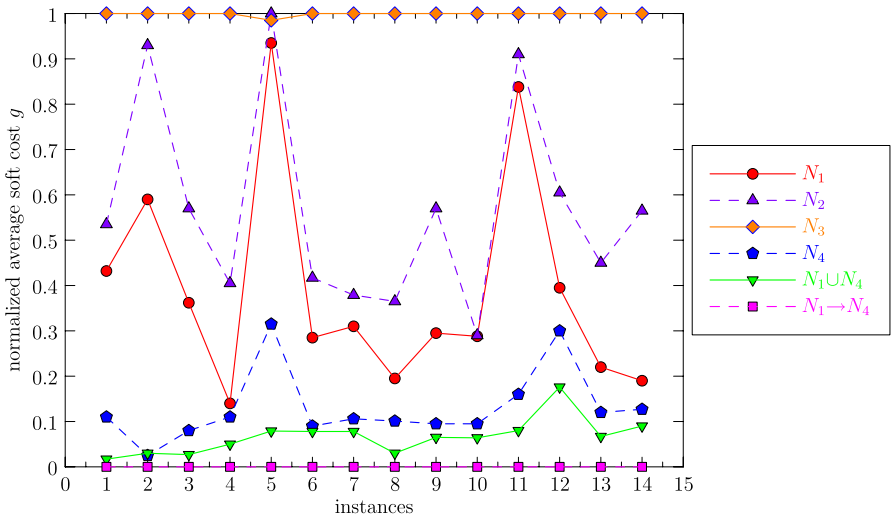

Fig. 9 Average solution quality comparisons for the ATS algorithm over 50 runs

Table 5 Best results obtained by TS, ILS and ATS algorithms with N_1, N_4, $N_1 \cup N_4$ and $N_1 \rightarrow N_4$

Instance	TS				ILS				ATS				Best in Müller (2008)
	N_1	N_4	$N_1 \cup N_4$	$N_1 \rightarrow N_4$	N_1	N_4	$N_1 \cup N_4$	$N_1 \rightarrow N_4$	N_1	N_4	$N_1 \cup N_4$	$N_1 \rightarrow N_4$	
comp01	8	5	5	5	6	5	5	5	6	5	5	**5**	**5**
comp02	123	65	63	55	111	55	54	48	96	52	52	**40**	43
comp03	145	108	102	92	124	81	80	76	114	79	80	**71**	72
comp04	89	59	55	47	82	55	52	42	55	51	44	39	**35**
comp05	431	362	354	320	416	355	346	305	401	345	336	**298**	**298**
comp06	136	82	76	58	129	70	68	54	87	67	67	47	**41**
comp07	133	58	50	35	124	54	50	26	69	44	36	21	**14**
comp08	98	69	69	53	89	65	59	48	67	60	56	43	**39**
comp09	167	142	130	110	147	121	119	106	135	115	109	**101**	103
comp10	110	62	50	28	98	45	40	24	57	35	31	18	**9**
comp11	0	0	0	0	0	0	0	0	0	0	0	**0**	**0**
comp12	446	378	368	332	429	390	385	324	396	380	374	**320**	331
comp13	120	91	84	71	105	98	90	69	95	90	78	**65**	66
comp14	96	68	67	57	86	64	61	53	63	58	59	55	**53**

We now turn our attention to the *best costs* that the three advanced metaheuristic algorithms can obtain. To compute the above results, Table 5 presents the *best costs* obtained by these three algorithms with neighborhoods N_1, N_4, $N_1 \cup N_4$ and $N_1 \rightarrow N_4$ over 50 independent runs. The reasons for discarding N_2 and N_3 are due to their relatively poor performance and the space limit. It should be noticed that the trends of the best costs are perfectly coincident with the average costs mentioned above for all the considered single neighborhoods and neighborhood combinations.

In order to show the performance of the proposed neighborhood N_4 compared with other reference algorithms, we listed in Table 5 (the last column) the best costs ob-

tained by the winner of the ITC–2007 (Müller 2008). The algorithm in Müller (2008) uses the same stopping condition as we do. However, one still finds that for the 14 competition instances, ATS with neighborhood $N_1 \rightarrow N_4$ reaches better (respectively worse) results than the winning algorithm in Müller (2008) for 5 (respectively 6) instances, with matching results for the remaining 3 instances. Note that our algorithm ATS with a neighborhood combination $(N_1 \cup N_2) \rightarrow (N_3 \cup N_4)$ ranks in second place for the track 3 of the ITC–2007.[2]

6 Conclusions

Understanding, explaining and predicting the performance of a neighborhood used by a local search algorithm is an important and difficult topic (Schuurmans and Southey 2001). In this paper, we present an attempt to analyze the intrinsic characteristics of four neighborhoods and their combinations for a real world application, i.e. the curriculum-based course timetabling problem. To this end, we introduce three evaluation criteria to characterize the search capability of a neighborhood: percentage of improving neighbors, improvement strength and search steps. The experimental analysis based on these criteria and a steepest descent allow us to understand to some extent the relative advantages and weaknesses of the four studied neighborhoods and identify the possibilities of combining them.

In particular, the analysis provides useful indications as to why the new neighborhood N_4 induced by the *KempeSwap* move is more powerful than the other existing neighborhoods. This analysis also discloses the complementary characteristics of *SimpleMove* and *KempeSwap*, giving a foundation for a meaningful combination of the two respective neighborhoods (N_1 and N_4). Concerning neighborhood combinations, the analysis explains why it is more advantageous to use N_1 and N_4 in a token-ring search ($N_1 \rightarrow N_4$) than in a neighborhood union ($N_1 \cup N_4$).

To further evaluate the impact of this study on practical problem solving with advanced metaheuristics, we carried out a series of experiments using three algorithms: Tabu Search, Iterate Local Search and Adaptive Tabu Search. Results confirm the advantage of *KempeSwap*-based neighborhood over other single neighborhoods on the one hand and the superiority of the token-ring combination of N_1 and N_4 over all other single and combined neighborhoods on the other hand. Moreover, using $(N_1 \rightarrow N_4)$ within the Adaptive Tabu Search algorithm has led to very competitive results on the 14 instances of the ITC–2007 competition compared with the winning algorithm of the competition.

To conclude, while the evaluation criteria introduced in this paper alone cannot fully explain or predict the performance of local search for a given neighborhood, They constitute useful indicators of good neighborhoods. It should be clear that the approach reported here is general, consequently it can be applied to other problems for neighborhood analysis and new neighborhood designs.

Finally, we observe that the neighborhood concept deserves to be looked upon in a more general way than it is customarily viewed in the metaheuristic area, by adopting

[2]This result is available at: http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm.

the perspective from tabu search whereby moves that are made during constructive and destructive processes deserve likewise to be conceived as offering types of neighborhoods to be exploited (Glover et al. 1984; Glover 1996). This is relevant not only for customary multi-start methods, but also for the application of strategic oscillation approaches that intervene in various stages of iterated construction or destruction to refine and upgrade the structures produced at these stages (Glover 1995, 1996). Questions worthy of investigation concern how best to integrate multiple neighborhoods in this setting as well, and the measures and procedures we have introduced here can be adapted in a natural manner for application in such contexts.

Acknowledgements We would like to thank the anonymous referees for their helpful comments and questions. The work is partially supported by a "Chaire d'excellence" from "Pays de la Loire" Region (France) and regional MILES (2007–2009) and RaDaPop projects (2009–2012).

References

Burke, E.K., Newall, J.P.: Solving examination timetabling problems through adaptation of heuristic orderings. Ann. Oper. Res. **129**, 107–134 (2004)

Burke, E.K., MacCarthy, B.L., Petrovic, S., Qu, R.: Multiple-retrieval case-based reasoning for course timetabling problems. J. Oper. Res. Soc. **57**(2), 148–162 (2006)

Casey, S., Thompson, J.: Grasping the examination scheduling problem. In: Burke, E.K., Causmaecker, P.D. (eds.) Proceedings of the 4th PATAT Conference. LNCS, vol. 2740, pp. 232–246. Springer, Berlin (2003)

Chiarandini, M., Birattari, M., Socha, K., Rossi-Doria, O.: An effective hybrid algorithm for university course timetabling. J. Sched. **9**, 403–432 (2006)

Côté, P., Wong, T., Sabourin, R.: Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. In: Burke, E.K., Trick, M. (eds.) Proceedings of the 5th PATAT Conference. LNCS, vol. 3616, pp. 151–168. Springer, Berlin (2005)

Di Gaspero, L., Schaerf, A.: Neighborhood portfolio approach for local search applied to timetabling problems. J. Math. Model. Algorithms **5**(1), 65–89 (2006)

Glover, F.: Tabu thresholding: Improved search by nonmonotonic trajectories. ORSA J. Comput. **7**(4), 426–442 (1995)

Glover, F.: Tabu Search and adaptive memory programming—advances, applications and challenges. In: Interfaces in Computer Science and Operations Research, pp. 1–75. Kluwer Academic, Dordrecht (1996)

Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Boston (1997)

Glover, F., McMillan, C., Glover, R.: A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots. J. Oper. Manag. **4**(2), 113–128 (1984)

Goëfon, A., Richer, J.M., Hao, J.K.: Progressive tree neighborhood applied to the maximum parsimony problem. IEEE/ACM Trans. Comput. Biol. Bioinform. **5**(1), 136–145 (2008)

Hansen, P., Mladenovi, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**(3), 449–467 (2001)

Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, Elsevier, San Francisco (2004)

Johnson, D.S.: A theoretician's guide to the experimental analysis of algorithms. In: Goldwasser, M.H., Johnson, D.S., McGeoch, C.C. (eds.) Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, pp. 215–250. American Mathematical Society, Providence (2002)

Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum **30**(1), 167–190 (2008)

Lourenco, H.R., Martin, O., Stützle, T.: Iterated local search. In: Handbook of Meta-heuristics, pp. 321–353. Springer, Berlin (2003)

Lü, Z., Hao, J.K.: A critical element-guided perturbation strategy for iterated local search. In: Cotta, C., Cowling, P. (eds.) EvoCop 2009. LNCS, vol. 5482, pp. 1–12. Springer, Berlin (2009)

Lü, Z., Hao, J.K.: Adaptive tabu search for course timetabling. Eur. J. Oper. Res. **200**(1), 235–244 (2010)
McCollum, B.: A perspective on bridging the gap between research and practice in university timetabling. In: Burke, E.K., Rudova, H. (eds.) Proceedings of the 6th PATAT Conference. LNCS, vol. 3867, pp. 3–23. Springer, Berlin (2007)
McCollum, B., McMullan, P., Paechter, B., Lewis, R., Schaerf, A., Di Gaspero, L., Parkes, A.J., Qu, R., Burke, E.K.: Setting the research agenda in automated timetabling: the second international timetabling competition. Technical Report http://www.cs.qub.ac.uk/itc2007/ITC2007_Background_Techreportv1.pdf (2008)
Merlot, L.T.G., Boland, N., Hughes, B.D., Stuckey, P.J.: A hybrid algorithm for the examination timetabling problem. In: Burke, E.K., Causmaecker, P.D. (eds.) Proceedings of the 4th PATAT Conference. LNCS, vol. 2740, pp. 207–231. Springer, Berlin (2003)
Mlandenovic, N., Hansen, P.: Variable neighbourhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
Müller, T.: Solver description: a hybrid approach. In: Burke, E.K., Gendreau, M. (eds.) Proceedings of the 7th PATAT Conference. http://www.unitime.org/papers/itc2007.pdf (2008)
Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Dover, Mineola (1998)
Rossi-Doria, O., Paechter, B., Blum, C., Socha, K., Samples, M.: A local search for the timetabling problem. In: Burke, E.K., Causmaecker, P.D. (eds.) Proceedings of the 4th PATAT Conference. Gent, Belgium (2002)
Schaerf, A.: A survey of automated timetabling. Artif. Intell. Review **13**(2), 87–127 (1999)
Schuurmans, D., Southey, F.: Local search characteristics of incomplete sat procedures. Artif. Intell. **132**(2), 121–150 (2001)
Sedgewick, R.: Algorithms, 2nd edn. Addison-Wesley, Reading (1988)