European Journal of Operational Research 200 (2010) 235–244

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

Adaptive Tabu Search for course timetabling

Zhipeng Lü [a,b,*], Jin-Kao Hao [a]

[a] LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
[b] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

ARTICLE INFO

Article history:
Received 13 September 2008
Accepted 1 December 2008
Available online 24 December 2008

Keywords:
Timetabling
Heuristic
Tabu Search
Iterated Local Search
Perturbation operator
ITC-2007

ABSTRACT

This paper presents an Adaptive Tabu Search algorithm (denoted by ATS) for solving a problem of curriculum-based course timetabling. The proposed algorithm follows a general framework composed of three phases: initialization, intensification and diversification. The initialization phase constructs a feasible initial timetable using a fast greedy heuristic. Then an adaptively combined intensification and diversification phase is used to reduce the number of soft constraint violations while maintaining the satisfaction of hard constraints. The proposed ATS algorithm integrates several distinguished features such as an original double Kempe chains neighborhood structure, a penalty-guided perturbation operator and an adaptive search mechanism. Computational results show the high effectiveness of the proposed ATS algorithm, compared with five reference algorithms as well as the current best known results. This paper also shows an analysis to explain which are the essential ingredients of the ATS algorithm.

1. Introduction

Timetabling is an area of increasing interest in the community of both research and practice in recent decades. In essence, it consists in assigning a number of events, each with a number of features, to a limited number of resources subject to certain (hard and soft) constraints. Typical cases in this area include educational timetabling [8], sport timetabling [25], employee timetabling [3], transport timetabling [26] and so on. In this paper, we consider an educational timetabling problem.

Educational timetabling problems are usually classified into two categories [18]: exam timetabling and course timetabling. The latter can be further divided into two sub-categories: post enrollment-based course timetabling and curriculum-based course timetabling. The main difference is that for post enrollment timetabling, conflicts between courses are set according to the students' enrollment data, whereas the curriculum-based course timetable is scheduled on the basis of the curricula published by the university. In this paper, our study is focused on the curriculum-based course timetabling (CB-CTT), which was recently proposed as the third track of the Second International Timetabling Competition (ITC-2007) [1]. One of the main objectives of this competition is to reduce the gap between research and practice within the area of educational timetabling [21].

For university curriculum-based course timetabling, a set of lectures must be assigned into timeslots and rooms subject to a given set of constraints. Usually, two types of constraints can be defined: Those which must be strictly satisfied under any circumstances (hard constraints) and those which are not necessarily satisfied but whose violations should be desirably minimized (soft constraints). An assignment that satisfies all the hard constraints is called a *feasible* timetable. The objective of the CB-CTT problem is to minimize the number of soft constraint violations in a feasible timetable.

The general timetabling problem is known to be complex and difficult. In this context, exact solutions would be only possible for problems of limited sizes. Instead, heuristic algorithms based on metaheuristics have shown to be highly effective. Examples of these algorithms include graph coloring heuristics [6], Tabu Search [31], simulated annealing [29], evolutionary algorithms [27], case-based reasoning [5], two-stage heuristic algorithms [8,10,17] and so on. Interested readers are referred to [18] for a comprehensive survey of the automated approaches for university timetabling presented in recent years.

The objective of this paper is two-fold: Describing a three-phases solution algorithm for solving the CB-CTT problem and investigating some essential ingredients of the proposed algorithm. The proposed ATS algorithm follows a general framework composed of three phases: Initialization, intensification and diversification (Section 3). The initialization phase builds a feasible initial timetable using a fast greedy heuristic. Then the intensification and diversification phases are adaptively combined to reduce the number of soft constraint violations while maintaining the satisfaction of hard constraints. The performance of the proposed

* Corresponding author. Address: LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France. Tel.: + 33 2 41 73 52 94.
E-mail addresses: zhipeng.lui@gmail.com, lu@info.univ-angers.fr (Z. Lü), hao@info.univ-angers.fr (J.-K. Hao).

hybrid algorithm was assessed on a set of 4 instances used in the literature and a set of 21 public competition instances from ITC-2007, showing very competitive results (Section 4).

As the second objective of this paper, we carefully investigate several important features of the proposed algorithm (Section 5). The analysis shed light on why some ingredients of our ATS algorithm are essential and how they lead to the efficiency of our ATS algorithm.

2. Curriculum-based course timetabling

2.1. Problem description

The CB-CTT problem consists in scheduling lectures of a set of courses into a weekly timetable, where each lecture of a course must be assigned a period and a room in accordance with a given set of constraints [13]. A feasible timetable is one in which all lectures have been scheduled at a timeslot and a room, so that the hard constraints $H_1 - H_4$ (see below) are satisfied. In addition, a feasible timetable satisfying the four hard constraints incurs a penalty cost for the violations of the four soft constraints $S_1 - S_4$. 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 and four soft constraints are:

- H_1. **Lectures**: Each lecture of a course must be scheduled in a distinct period and a room.
- H_2. **Room occupancy**: Any two lectures cannot be assigned in the same period and the same room.
- H_3. **Conflicts**: Lectures of courses in the same curriculum or taught by the same teacher cannot be scheduled in the same period, i.e., no period can have an overlapping of students nor teachers.
- H_4. **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.
- S_1. **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.
- S_2. **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.
- S_3. **Minimum working days**: The lectures of a course should be spread into the given minimum number of days.
- S_4. **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 present below a mathematical formulation of the problem which is missing in the literature.

2.2. Problem formulation

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. For simplicity and when no confusion is possible, we do not distinguish between *lecture*, *course* and *course label* in the following context. A period is a pair composed of a day and a timeslot, p periods being distributed in d days and h daily timeslots, i.e., $p = d \times h$. In addition, there are a set of s curricula $CR = \{Cr_1, Cr_2, \ldots, Cr_s\}$ where each curriculum Cr_k is a group of courses that share common students.

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". With this representation we ensure that there will be no more than one course assigned to each room in any period, meaning that the second hard constraint H_2 will always be satisfied. For courses, rooms, curricula and solution representation X, a number of symbols and variable definitions are presented in Table 1.

Given these notations, we can describe the CB-CTT problem in a formal way for a candidate solution X. The four hard constraints and the penalty costs for the four soft constraints are as follows:

- H_1. **Lectures**: $\forall c_k \in C$,

$$\sum_{i=1,\ldots,p, j=1,\ldots,m} \chi\{x_{i,j} = c_k\} = l_k,$$

where χ is the truth indicator function which takes values of 1 if the given proposition is true and 0 otherwise.

Table 1
Notations used for the CB-CTT problem.

Symbols	Description
n	The total number of courses
m	The total number of rooms
d	The number of working days per week
h	The number of timeslots per working day
p	The total number of periods, $p = d \times h$
s	The total number of curricula
C	Set of the courses, $C = \{c_1, \cdots, c_n\}$, $\|C\| = n$
R	Set of the rooms, $R = \{r_1, \cdots, r_m\}$, $\|R\| = m$
T	Set of the periods, $T = \{t_1, \cdots, t_p\}$, $\|T\| = p$
CR	Set of the curricula, $CR = \{Cr_1, \cdots, Cr_s\}$, $\|CR\| = s$
Cr_k	The kth curriculum including a set of courses
l_i	The number of lectures of course c_i
l	The total number of all lectures, $l = \sum_1^n l_i$
std_i	The number of students attending course c_i
tc_i	The teacher instructing course c_i
md_i	The number of minimum working days of course c_i
cap_j	The capacity of room r_j
$uav_{i,j}$	Whether course c_i is unavailable at period t_j. $uav_{i,j} = 1$ if it is unavailable, $uav_{i,j} = 0$ otherwise
con_{ij}	Whether course c_i and c_j are conflict with each other; $\mathrm{con}_{ij} = \begin{cases} 0, & \text{if } (tc_i \neq tc_j) \wedge (\forall Cr_q, \quad c_i \notin Cr_q \vee c_j \notin Cr_q), \\ 1, & \text{otherwise.} \end{cases}$
$x_{i,j}$	The course assigned at period t_i and room r_j
$nr_i(X)$	Number of rooms occupied by course c_i for a candidate solution X; $nr_i(X) = \sum_{j=1}^m \sigma_{ij}(X)$, where $\sigma_{ij}(X) = \begin{cases} 1, & \text{if } \forall x_{k,j} \in X, \quad x_{k,j} = c_i, \\ 0, & \text{otherwise.} \end{cases}$
$nd_i(X)$	Number of working days that course c_i takes place at in candidate solution X; $nd_i(X) = \sum_{j=1}^d \beta_{ij}(X)$, where $\beta_{ij}(X) = \begin{cases} 1, & \text{if } \forall x_{u,v} \in X, \quad x_{u,v} = c_i \wedge [u/h] = j, \\ 0, & \text{otherwise.} \end{cases}$
$\mathrm{app}_{k,i}(X)$	Whether curriculum Cr_k appears at period t_i in candidate solution X; $\mathrm{app}_{k,i}(X) = \begin{cases} 1, & \text{if } \forall x_{i,j} \in X, \quad x_{i,j} = c_u \wedge c_u \in Cr_k, \\ 0, & \text{otherwise.} \end{cases}$

- H_2. **Room occupancy**: This hard constraint is always satisfied using our solution representation.
- H_3. **Conflicts**: $\forall x_{i,j}, x_{i,k} \in X, x_{i,j} = c_u, x_{i,k} = c_v$,

  $\mathrm{con}_{uv} = 0$.
- H_4. **Availability**: $\forall x_{i,j} = c_k \in X$,

  $uav_{k,i} = 0$.
- S_1. **Room capacity**: $\forall x_{i,j} = c_k \in X$,

  $$f_1(x_{i,j}) = \begin{cases} \mathrm{std}_k - \mathrm{cap}_j, & \text{if } \mathrm{std}_k > \mathrm{cap}_j, \\ 0, & \text{otherwise}. \end{cases}$$
- S_2. **Room stability**: $\forall c_i \in C$,

  $f_2(c_i) = nr_i(X) - 1$.
- S_3. **Minimum working days**: $\forall c_i \in C$,

  $$f_3(c_i) = \begin{cases} md_i - nd_i(X), & \text{if } nd_i(X) < md_i, \\ 0, & \text{otherwise}. \end{cases}$$
- S_4. **Curriculum compactness**: $\forall x_{i,j} = c_k \in X$,

  $$f_4(x_{i,j}) = \sum_{Cr_q \in CR} \chi\{c_k \in Cr_q\} \cdot \mathrm{iso}_{q,i}(X),$$

  where

  $$\mathrm{iso}_{q,i}(X) = \begin{cases} 1, & \text{if } (i \bmod h = 1 \vee \mathrm{app}_{q,i-1}(X) = 0), \\ & \wedge\ (i \bmod h = 0 \vee \mathrm{app}_{q,i+1}(X) = 0), \\ 0, & \text{otherwise}. \end{cases}$$

With the above formulation, we can then calculate the total soft penalty cost for a given candidate feasible solution X according to the cost function f defined in formula (1). The goal is then to find a feasible solution X^* such that $f(X^*) \leqslant f(X)$ for all X in the feasible search space

$$f(X) = \sum_{x_{i,j} \in X} \alpha_1 \cdot f_1(x_{i,j}) + \sum_{c_i \in C} \alpha_2 \cdot f_2(c_i) + \sum_{c_i \in C} \alpha_3 \cdot f_3(c_i) + \sum_{x_{i,j} \in X} \alpha_4 \cdot f_4(x_{i,j}), \qquad (1)$$

α_1, α_2, α_3 and α_4 are the penalties associated to each of the soft constraints. In the CB-CTT formulation, they are set as: $\alpha_1 = 1, \alpha_2 = 1, \alpha_3 = 5, \alpha_4 = 2$. Note that $\alpha_1 \sim \alpha_4$ are fixed in the problem formulation and should not be confused with the penalty parameters used by some solution procedures.

3. Solution method

Our Adaptive Tabu Search algorithm (ATS) follows a general framework composed of three phases: initialization, intensification and diversification. The initialization phase (Section 3.1) constructs a feasible initial timetable using a fast greedy heuristic. As soon as a feasible initial assignment is reached, the adaptively combined intensification and diversification phase is used to reduce the number of soft constraint violations. The intensification phase (Section 3.2) employs a Tabu Search algorithm [15] while the diversification phase (Section 3.3.1) is based on a penalty-guided perturbation operator borrowed from Iterated Local Search [20]. Furthermore, two self-adaptive mechanisms (Section 3.3.2) are employed to provide a tradeoff between intensification and diversification.

3.1. Initial solution

The first phase of our algorithm generates a feasible initial solution satisfying all the hard constraints ($H_1 - H_4$). This is achieved by a sequential greedy heuristic starting from an empty timetable, from which course assignments are constructed by inserting one appropriate lecture into the timetable at each time. At each step, two distinct operations are carried out: one is to select an unassigned lecture of a course, the other is to determine a period-room pair for this lecture.

In the lecture selection heuristic, the courses with a small number of available periods and a large number of unassigned lectures have priority. This heuristic is similar to the greedy coloring heuristic DSATUR [4]. Once we have chosen one lecture of a course to assign, we want to select a period among all available ones that is least likely to be used by other unfinished courses at later steps. For this purpose, when attempting to make a feasible insertion move, we count the total number of unfinished courses that become unavailable at the current period. The feasible lecture insertion moves with small value of this number are highly favored. Ties are broken according to the soft constraint penalty incurred.

We have no proof that this greedy heuristic guarantees to find a feasible solution for a given instance. However, for all the tested instances in this paper, a feasible solution is always easily obtained. Notice that infeasibility of the initial solution does not change the general ATS approach since unsatisfied hard constraints can be relaxed and incorporated into the evaluation function of the ATS algorithm.

3.2. Tabu Search algorithm

In this section, we focus on the basic search engine of our ATS algorithm – Tabu Search [15]. Our TS procedure exploits two neighborhoods (denoted by N_1 and N_2, see below) in a token-ring way [12]. More precisely, we start the TS procedure with one neighborhood. When the search ends at its best local optimum, we restart TS from this local optimum, but with the other neighborhood. This process is repeated until no improvement is possible and we say that a TS phase is achieved. In our case, the TS procedure begins with the basic neighborhood N_1: $N_1 \rightarrow N_2 \rightarrow N_1 \rightarrow N_2 \ldots$

3.2.1. Search space and evaluation function

Once a feasible timetable that satisfies all the hard constraints is reached, our intensification phase (TS algorithm) optimizes the soft constraint cost function without breaking hard constraints any more. Therefore, the search space of our TS algorithm is limited to the feasible timetables. The evaluation function is just the soft constraint violations as defined in formula (1).

3.2.2. Neighborhood structure

It is widely believed that one of the most important features of a local search algorithm is the definition of its neighborhood. 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 while maintaining feasibility, then the neighborhood N of X is defined by: $N(X) = \{X \oplus mv \mid mv \in M(X)\}$. For the CB-CTT problem, we use two distinct *moves* denoted by *SimpleSwap* and *KempeSwap*.

Basic neighborhood N_1: N_1 is composed of all feasible moves of *SimpleSwap*. 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 $x_{i',j'}$ in solution X consists in assigning the value of $x_{i,j}$ to $x_{i',j'}$ and inversely the value of $x_{i',j'}$ to $x_{i,j}$. Note that moving one lecture of a course to a free position is a special case of the *SimpleSwap* move where one of the lectures is empty and it is also included in N_1. Therefore, the size of neighborhood N_1 is bounded by $O(l \cdot p \cdot m)$ where $l = \sum_{i=0}^{n-1} l_i$ because there are l lectures and the number of swapping lectures (including free positions) is bounded by $O(p \cdot m)$.

Advanced neighborhood N_2: N_2 is composed of all feasible moves of *KempeSwap*. A *KempeSwap* move is defined by interchanging two Kempe chains. If we focus only on courses and conflicts, each problem instance can be looked as a graph G where nodes are courses and edges connect courses with students or teacher in common. In a feasible timetable, a Kempe chain is the set of nodes that forms a connected component in the subgraph of G induced by the nodes that belong to two periods. A *KempeSwap* produces a new feasible assignment by swapping the period labels assigned to the courses belonging to one or two specified 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 t_i with $(t_i \setminus (K_1 \cup K_2)) \cup (t_j \cap (K_1 \cup K_2))$ and t_j with $(t_j \setminus (K_1 \cup K_2)) \cup (t_i \cap (K_1 \cup K_2))$. Note that in the definition of N_2 at least three courses are involved, i.e., $| K_1 | + | K_2 | \geqslant 3$. For instance, Fig. 1a depicts a subgraph deduced by two periods t_i and t_j and there are five Kempe chains: $K_a = \{c_1, c_2, c_7, c_8\}$, $K_b = \{c_3, c_6, c_9\}$, $K_c = \{c_4, c_{11}, c_{12}\}$, $K_d = \{c_5\}$ and $K_e = \{c_{10}\}$. In this example, each room at periods t_i and t_j has one lecture. A *KempeSwap* of K_b and K_c produces a new assignment by moving $\{c_3, c_4, c_6\}$ to t_j and $\{c_9, c_{11}, c_{12}\}$ to t_i, as shown in Fig. 1b.

Note that in our *KempeSwap*, one of the swapping Kempe chains can be empty, i.e., we add a new empty Kempe chain $K_f = \emptyset$. In this case, the move of *KempeSwap* degenerates into a single Kempe chain interchange. Formally, it means replacing t_i with $(t_i \setminus K) \cup (t_j \cap K)$ and t_j with $(t_j \setminus K) \cup (t_i \cap K)$ where K is the non-empty Kempe chain [22,29]. For example, in Fig. 1a, if we exchange the courses of the Kempe chain K_a, it produces an assignment by moving $\{c_1, c_2\}$ to t_j and $\{c_7, c_8\}$ to t_i. 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 [8,29].

Once courses are scheduled to periods, the room assignment can be done by solving a bipartite matching problem [24], where both heuristic and exact algorithms can be employed. In this paper, we implement an exact algorithm, the augmenting path algorithm implemented in [28], which runs in $O(|V||E|)$.

Since *KempeSwap* can be considered as an extended version of swapping two lectures (and afterwards several other related lectures in the specified Kempe chain(s) being moved), the size of N_2 is bounded by $O(l \cdot (l + p))$, where the size of double Kempe chains interchange is bounded by $O(l^2)$ and the size of single Kempe chain interchange is bounded by $O(l \cdot p)$.

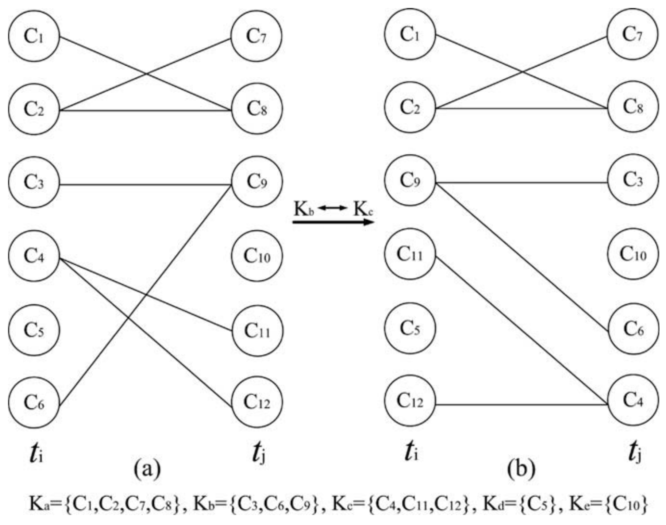

Fig. 1. Kempe chain illustrations.

In order to maintain the feasibility of the Kempe chain neighborhood solution, another important property must be verified: The number of courses in each period (after Kempe chain exchange) cannot exceed the number of available rooms. For example, in Fig. 1, with respect to the single Kempe chain interchange, only one feasible move can be produced by interchanging courses in K_a, while other four single Kempe chain interchanges (K_b, K_c, K_d and K_e) cannot produce feasible solutions since these moves violate the above-mentioned property and thus are forbidden. In fact, this property largely restricts the number of acceptable candidate solutions for single Kempe chain interchanges. We call this restriction *room allocation violation*.

However, as soon as the double Kempe chains interchange is performed, the *room allocation violation* is relaxed and a large number of feasible moves can be generated. For instance, in Fig. 1 three double Kempe chains interchanges can be produced by swapping K_b and K_e, K_c and K_d as well as K_b and K_c.

We will show in Section 5.2 that the proposed double Kempe chains move is much more powerful than other existing moves (one lecture move, two lectures swap and single Kempe chain interchange) for timetabling [8,18].

3.2.3. Incremental evaluation and neighborhood reduction

In order to evaluate the neighborhood in an efficient way, we use an incremental evaluation technique. The main idea 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.

However, the move evaluation of the advanced neighborhood N_2 needs much more computational efforts than that of N_1 due to the running of the matching algorithm. In order to save CPU time, we attempt to use the matching algorithm as few as possible. According to the problem formulation, the soft costs can be classified into the room-related (S_1 and S_2) and period-related (S_3 and S_4) costs. From the definition of N_2, it is clear that the period-related cost Δf_p can be calculated without calling the matching algorithm and therefore it is easy to calculate, while the calculation of the room-related cost Δf_r is time consuming due to the higher computational cost of the matching algorithm. In our implementation, we only record and update the period-related *move values* Δf_p for the neighborhood solutions of N_2, while for the room-related move values, a special reduction technique is employed to decide whether to call the matching algorithm or not.

In fact, we use the period-related cost Δf_p as a goodness estimation of the Kempe move. Specifically, if the period-related cost Δf_p is promising (i.e., $\Delta f_p \leqslant \tau$, practically $\tau = 2$ produces competitive results for a large class of instances), then we call the matching algorithm to make room allocations and obtain the total incremental evaluation cost Δf. Otherwise, this neighborhood candidate solution will be discarded. In this way, at each iteration only a small subset of the promising neighboring solutions are thoroughly evaluated, thus allowing us to save a considerable amount of CPU time.

3.2.4. Tabu list management

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 from one period to another (using N_2), this lecture cannot be moved back to the previous position (for N_1) or period (for N_2) for the next *tt* iterations (*tt* is called the "tabu tenure"). More precisely, in neighborhood N_1, if a lecture of a course c_i is moved from one position (t_j, r_k) to another one, then moving any lecture of course c_i to the position (t_j, r_k) is declared tabu. On the other hand, in neighborhood N_2 (either single or double Kempe chains move), if one lecture of course c_i is moved from period t_j to t_k, it is tabu to assign any lecture of c_i to t_j using a (single or double) Kempe chain move.

The tabu tenure $tt(c_i)$ of a course c_i is tuned adaptively according to the current solution quality f and the frequency of the moves involving lectures of course c_i, denoted by $tt(c_i) = f + \varphi \cdot \text{freq}(c_i)$ where φ is a parameter that takes values in $[0, 1]$. The first part of this function can be explained by the reason that a solution with high soft cost penalties should have a long tabu tenure to escape from the local optimum trap. The basic idea behind the second part is to penalize a move which repeats too often. The coefficient φ is dynamically defined as the ratio of the number of conflicting courses of c_i over the total number of courses. It is reasonable to think that a course involved in a large number of conflicts has more risk to be moved than a course having fewer conflicts. Notice that $freq(c_i)$ is the essential part of the above tabu tenure function and frequency-based tabu tenure techniques have been used in the literature, see e.g. [30].

3.2.5. Aspiration criteria and stop condition

In our TS algorithm, the tabu status of a move is disabled if it leads to a solution better than the current best solution. Our TS stops when the best solution cannot be improved within a given number θ of moves and we call this number the *depth* of TS.

3.3. Adaptive TS: Combining TS with perturbation

TS and Iterated Local Search (ILS) are two well-known metaheuristics and have proved their efficiency for solving separately a large number of constraint satisfaction and optimization problems [15,20]. In this paper, we consider the possibility of combining them to achieve very high performances for the CB-CTT problem.

TS can be used with both long and short CPU time. In general, long CPU time would lead to better results. However, if the total computation time is limited (e.g., this is the case of the ITC-2007), it would be preferred to combine short TS runs with some robust diversification operators. Interestingly, ILS provides such diversification mechanisms to guide the search to escape from the current local optimum and move towards new promising regions in the search space [20].

3.3.1. A penalty-guided perturbation strategy

In our case, when the best solution cannot be improved any more using the TS algorithm, we employ a perturbation operator to reconstruct the obtained local optimum solution. *Perturbation strength* is one of the most important factors of ILS. In general, if the perturbation is too strong, it may behave like a random restart. On the other hand, if the perturbation is too small, the search would fall back into the local optimum just visited and the exploration of the search space will be limited within a small region.

In order to guide efficiently the search to move towards new promising regions of the search space, we employ a *penalty-guided* perturbation operator to destruct the reached local optimum solution. This operator is based on the identification of a set of the first q highly-penalized lectures and a random selection of a given number of neighborhood moves (in this paper, we experimentally used $q = 30$). We call the total number of perturbation moves *perturbation strength*, denoted by η.

Specifically, when the current TS phase terminates, all the lectures are ranked in a non-increasing order according to their soft penalties involved. Then, η lectures are selected from the first q highly-penalized ones, where the lecture of rank k is selected according to the following probability distribution:

$$P(k) \propto k^{-\phi},$$

where ϕ is a positive real number (empirically set at 4.0). After that, η feasible moves of *SimpleSwap* or *KempeSwap* are randomly and sequentially performed, each involving at least one of the selected η lectures.

As previously mentioned, the *perturbation strength* η is one of the most important ingredients of ILS, which determines the quality gap between the two solutions before and after perturbation. In our case, η is adaptively adjusted and takes values in an interval $[\eta_{\min}, \eta_{\max}]$ (set experimentally $\eta_{\min} = 4, \eta_{\max} = 15$).

3.3.2. Two self-adaptive mechanisms for ATS

Adaptive Tabu Search

```
 1: **Input**: $I$: an instance of CB-CTT
 2: **Output**: $X^*$: the best solution found so far
 3: % **Initialization**: lines 6–8
 4: % **Intensification**: lines 11–17
 5: % **Diversification**: lines 10, 23
 6: $X_0 \leftarrow$ feasible initial solution
 7: $\xi \leftarrow 0$, $\theta \leftarrow \theta_0$, $\eta \leftarrow \eta_{\min}$
 8: $X^* \leftarrow TS(X_0, \theta)$
 9: **repeat**
10:     $X' \leftarrow$ Perturb$(X^*, \eta)$
        % perturb $X^*$ with strength $\eta$, get $X'$
11:     $X^{*'} \leftarrow TS(X', \theta)$
12:     **if** $f(X^{*'}) \leqslant f(X^*) + 2$ **then**
13:         **repeat**
14:         $\theta \leftarrow (1 + \mu) \cdot \theta$
            % gradually increase the *depth of TS*
15:     $X^{*'} \leftarrow TS(X^{*'}, \theta)$
16:         **until** no better solution is obtained
17: **end if**
18: **if** $f(X^{*'}) < f(X^*)$ **then**
19:         $X^* \leftarrow X^{*'}$
        % accept $X^{*'}$ as the best solution found so far
20:         $\theta \leftarrow \theta_0$, $\eta \leftarrow \eta_{\min}$
21: **else**
22:         $\theta \leftarrow \theta_0$, $\xi \leftarrow \xi + 1$
23:         $\eta \leftarrow \max\{\eta_{\min} + \lambda \cdot \xi, \eta_{\max}\}$
24: **end if**
25: **until** (stop condition is met)
```

Our ATS algorithm integrates intensification (TS) and diversification (ILS's Perturbation) phases. Instead of just simply combining the TS and ILS algorithms, we attempt to integrate them in a more meaningful way. The *depth* of TS θ and the *perturbation strength* η seem to be two essential parameters which control the behavior of the ATS algorithm. On the one hand, a greater θ value ensures a more intensive search. On the other hand, a greater η corresponds to more possibilities of escaping from the current local minimum. In order to get a continuous tradeoff between intensification and diversification, we devise a mechanism to dynamically and adaptively adjust these two important parameters according to the history of the search process.

Our Adaptive Tabu Search algorithm is summarized in Algorithm "Adaptive Tabu Search". At the beginning of the search, we run a short TS where the *depth of TS* is small (say $\theta = 10$). When TS cannot improve its best solution, perturbation is applied to the best solution with a weak strength ($\eta = \eta_{\min}$). When the search progresses, we record the number of TS phases (denoted by ξ) without improvement in cost function. The *depth* of TS θ and the *perturbation strength* η are dynamically adjusted as follows: when the local minimum obtained by TS is promising, i.e., it is close to the current best solution ($f \leqslant f_{\text{best}} + 2$), θ is gradually increased to ensure a more and more intensive search until no improvement is possible, i.e., $\theta = (1 + \mu) \cdot \theta$ at each iteration ($\mu = 0.6$). Similarly, *perturbation strength* is gradually increased so as to diversify more strongly the search if the number of non-improving TS phases increases. Moreover, the search turns back to the short TS after each perturbation, while the perturbation strength is set back to $\eta_{\min}$ as soon as a better solution is found.

For acceptance criterion in the perturbation process, we use a strong exploitation technique, i.e., only better solution is accepted as the current best solution. As soon as the local optimal solution $X^{*'}$ obtained by TS is better than the best solution X^* found so far, we replace the best known solution X^* with $X^{*'}$, as shown in lines 18 and 19 of Algorithm "Adaptive Tabu Search". In this paper, we use two stop conditions as described: The time limit imposed by the ITC-2007 competition rules and a maximum number of moves (see Section 4).

4. Experimental results

4.1. Problem instances and experimental protocol

To evaluate the efficiency of our proposed ATS algorithm, we carry out experiments on two different data sets. The first (4 instances) was previously used in the literature for the old version of the CB-CTT problem [12]. The second (21 instances) is from the Second International Timetabling Competition [1].

Our algorithm is programmed in C and compiled using GNU GCC on a PC running Windows XP with 3.4GHz CPU and 2.0Gb RAM. To obtain our computational results, each instance is solved 100 times independently with different random seeds.

All the computational results were obtained without special tuning of the parameters, i.e., all the parameters used in our algorithm are fixed (constant) or dynamically and automatically tuned during the problem solving for all the instances considered here. It is possible that better solutions would be found by using a set of instance-dependent parameters. However, our aim is to design a robust solver which is able to solve efficiently a large panel of instances. Table 2 gives the descriptions and settings of the important parameters used in our ATS algorithm.

4.2. Results under ITC-2007 timeout condition

Our first experiment aims to evaluate the ATS algorithm on the 4 previous instances (*test1–test4*) and 21 competition instances (*comp01–comp21*) of the ITC-2007, by comparing its performance with its two basic components (TS and ILS). To make the comparison as fair as possible, we implement the TS and ILS algorithms by reusing the ATS algorithm as follows. We define the TS algorithm as the ATS algorithm with its adaptive perturbation operator disabled. In order to give more search power to the TS algorithm, the *depth of TS* is gradually increased until the timeout condition is met. The ILS algorithm is the ATS algorithm with the tabu list disabled. All the other ingredients of the ATS are thus shared by the three compared algorithms.

Table 2
Settings of important parameters.

Parameters	Description	Values or updating
θ_0	Basic depth of TS	10
μ	Increment speed of θ	0.6
θ	Depth of TS	$\theta = (1+\mu)\cdot\theta$
ξ	Non-improvement TS phases	$\xi = \xi + 1$
$\eta_{\min}$	Basic perturbation strength	4
$\eta_{\max}$	Strong perturbation strength	15
η	Perturbation strength	$\eta = \max\{\eta_{\min} + \lambda\cdot\xi, \eta_{\max}\}$
λ	Updating factor of η	0.3
q	Total candidate number of perturbation lectures	30
ϕ	Importance factor for perturbation lecture selection	4.0
τ	Reduction cutoff for advanced neighborhood N_2	2

We also assess the performance of our ATS algorithm with respect to five other reference algorithms, which include ITC-2007 organizer's algorithm developed by De Cesco et al in [11], the winner algorithm of ITC-2007 by Müller in [23], the algorithm by Lach and Lübbecke in [16], the 4th place algorithm of ITC-2007 by Geiger in [14] and the 5th place algorithm of ITC-2007 by Clark et al. in [9].

All these 8 algorithms use the same stop condition which is just the timeout condition required by ITC-2007 competition rules. On our PC, this corresponds to 390 seconds. Table 3 summarizes the computational statistics of our ATS algorithm and Table 4 gives the best results obtained by our three algorithms and these reference algorithms.

In Table 3, columns 2–7 give the computational statistics of our ATS algorithm, according to the following performance indicators: the best score ($f_{\min}$), the average score (f_{ave}), the standard deviation (σ), the total number of iteration moves (*Iter*), the total number of perturbations (*Pert*) and the total CPU time on our computer needed to find the best solution $f_{\min}$ (*Time*). If there exist multiple hits on the best solution in the 100 independent runs, the values listed in Table 3 are the average over these multiple best hits.

Table 4 shows the best results obtained by our three algorithms ATS, TS and ILS, as well as the five reference algorithms. The last column in Table 4 also indicates the best known results obtained by these five reference algorithms for each instance under the ITC-2007 timeout condition. From Table 4, one clearly observes that the ATS algorithm outperforms its two basic components TS and ILS alone on all the instances (except for three where they get the same score). This demonstrates the importance of the hybrid mechanism of adaptively integrating TS and ILS.

When comparing with the best known results obtained by the five reference algorithms (last column in Table 4), one observes that the best results obtained by our ATS algorithm are quite competitive with respect to these previously best known results (best results for each instance are indicated in bold and equal best results are indicated in italic). For the 4 previous instances, ATS significantly improves the best known results obtained by De Cesco et

Table 3
Computational statistics of the ATS algorithm under the ITC-2007 competition stop conditions.

Instance	ATS					
	$f_{\min}$	f_{ave}	σ	*Iter*	*Pert*	*Time*(*s*)
test1	224	229.5	1.8	15,586	208	189
test2	16	17.1	1.0	35,271	406	182
test3	74	82.9	4.1	20,549	369	160
test4	74	89.4	6.1	37,346	735	208
comp01	5	5.0	0.0	321	5	5
comp02	34	60.6	7.5	15,647	545	370
comp03	70	86.6	6.3	8246	102	257
comp04	38	47.9	4.0	5684	68	124
comp05	298	328.5	11.7	35,435	54	191
comp06	47	69.9	7.4	13,457	245	116
comp07	19	28.2	5.6	15,646	368	383
comp08	43	51.4	4.6	17,404	190	380
comp09	99	113.2	6.9	20,379	238	370
comp10	16	38.0	10.8	16,026	160	389
comp11	0	0.0	0.0	236	3	3
comp12	320	365.0	17.5	40,760	590	382
comp13	65	76.2	6.1	16,779	182	300
comp14	52	62.9	6.4	24,427	270	368
comp15	69	87.8	7.3	20,666	275	386
comp16	38	53.7	6.4	8512	99	215
comp17	80	100.5	7.8	15,009	151	364
comp18	67	82.6	5.3	51,612	577	389
comp19	59	75.0	5.9	8788	94	225
comp20	35	58.2	8.5	6188	61	187
comp21	105	125.3	7.6	16,566	167	348

Table 4
Best results and comparison with other algorithms under the ITC-2007 timeout conditions.

Instance	ATS	TS	ILS	[11]	[23]	[16]	[14]	[9]	Best known
test1	**224**	230	226	234	–	–	–	–	234
test2	**16**	16	16	17	–	–	–	–	17
test3	**74**	82	79	86	–	–	–	–	86
test4	**74**	92	83	132	–	–	–	–	132
comp01	*5*	5	5	5	5	13	5	9	*5*
comp02	**34**	55	48	75	43	43	108	103	43
comp03	**70**	90	76	93	72	76	115	101	72
comp04	38	45	41	45	35	38	67	55	**35**
comp05	*298*	315	303	326	298	314	408	370	*298*
comp06	47	58	54	62	41	41	94	112	**41**
comp07	19	33	25	38	14	19	56	97	**14**
comp08	43	49	47	50	39	43	75	72	**39**
comp09	**99**	109	106	119	103	102	153	132	102
comp10	16	23	23	27	9	14	66	74	**9**
comp11	*0*	0	0	0	0	0	0	1	*0*
comp12	**320**	330	324	358	331	405	430	393	331
comp13	**65**	71	68	77	66	68	101	97	66
comp14	**52**	55	53	59	53	54	88	87	53
comp15	**69**	78	74	87	–	–	–	–	87
comp16	**38**	48	42	47	–	–	–	–	47
comp17	**80**	85	81	86	–	–	–	–	86
comp18	**67**	78	69	71	–	–	–	–	71
comp19	**59**	65	65	74	–	–	–	–	74
comp20	**35**	42	35	54	–	–	–	–	54
comp21	**105**	115	106	117	–	–	–	–	117

Table 5
Computational statistics of ATS algorithm under relaxed stop condition.

Instance	ATS					
	$f_{\min}$	f_{ave}	σ	*Iter*	*Pert*	*Time*(s)
test1	224	227.2	0.5	17,845	234	216
test2	16	16.0	0	32,416	351	167
test3	73	76.0	2.13	40,849	667	2078
test4	73	86.4	4.23	109,198	2054	1678
comp01	5	5.0	0.0	321	5	5
comp02	29	50.6	8.78	768,334	1032	3845
comp03	66	78.6	6.07	160,909	1903	2078
comp04	35	42.3	3.53	23,113	266	566
comp05	292	328.5	11.7	35,435	54	191
comp06	37	57.3	8.1	562,144	3213	5973
comp07	13	29.7	6.48	390,912	3508	4035
comp08	39	48.8	3.75	203,982	2352	3069
comp09	96	110.3	5.8	215,891	2711	1754
comp10	10	28.8	9.0	33,971	371	838
comp11	0	0.0	0.0	247	4	3
comp12	310	328.5	11.7	742,316	10,392	2513
comp13	59	69.9	7.4	793,989	10,078	4207
comp14	51	56.3	4.95	93,549	1165	1320
comp15	68	79.8	5.75	193,200	2429	9355
comp16	23	46.8	6.6	264,512	1174	10280
comp17	69	91.1	6.7	181,977	1995	2812
comp18	65	74.6	4.7	134,205	985	7526
comp19	57	69.4	4.6	105,983	1320	9835
comp20	22	42.1	6.7	216,482	3265	8746
comp21	93	117.8	6.9	184,065	1345	4891

al. in [11] (results for these 4 instances are not available for the other four reference algorithms). For the 21 public competition instances, ATS reaches better (respectively worse) results than the previous best known results for 13 (respectively 5) ones, with equaling results for the remaining 3 ones. Note that for the seven hidden instances (*comp15* to *comp21*), computational results are available only for the algorithm in [11] under ITC-2007 timeout condition.

4.3. Results using more computational resources

In our second experiment, we evaluate the search potential of our ATS algorithm with a relaxed stop condition. For this purpose, we use a longer CPU time and run our ATS algorithm until 800,000 iterations are reached. Table 5 shows the computational statistics of our ATS algorithm under this stop condition and indicates the following information: $f_{\min}$, f_{ave}, σ, *Iter*, *Pert* and *Time*(s) over 100 independent runs. The meaning of all these symbols are the same as in Table 3. If we compare the results of ATS shown in Tables 3 and 5, one finds that better solutions (smaller f_{best}) are found under the relaxed stop condition for 21 out of 25 instances. Moreover, the averaged results (f_{ave}) and standard deviations (σ) are also slightly better.

Table 6 shows the best results obtained by our ATS algorithm[1], compared with the best known results available on the web site [2] which is maintained by the organizers of ITC-2007. This site provides a systematic information about the CB-CTT problem and the dynamically updated best known results uploaded by researchers (the column "*best known*" in Table 6).[2] We also cited the best results obtained by Schaerf and Müller from the web site [2].

From Table 6, one finds that our ATS algorithm reaches quite competitive results. For the 25 tested instances, ATS reaches better (respectively worse) results than the previous best known results for 12 (respectively 5) ones, with equaling results for the remaining 8 ones, showing the strong search potential of our ATS algorithm.

Recently, a branch-and-cut procedure [7] and an integer programming approach [16] were proposed to find the lower bounds of the CB-CTT problem. However, except for few instances (marked with a $*$ in Table 6), these results are far from the current best known results. Given this fact, it is difficult to have an absolute assessment of these results for the moment. Therefore, tight lower bounds are necessary to be developed.

4.4. Comments on the ITC-2007 competition

In this section, we review the ITC-2007 competition rules and results. For ITC-2007, the evaluation process was divided into two phases [1]. The first phase aimed to selected the (five) *Finalists* and the selection was based on the computational results of the first 14 competition instances (*comp01–comp14*). The second phase used an additional set of 7 hidden instances (*comp15–comp21*, now they become available for researchers).

For each of the 21 competition instances, the organizers solved it with 10 independent runs using each of the five finalist algorithms. A *ranking* was then calculated based on these 50 results for the given instance. At the end, a final ranking was established according to the ranks realized on the 21 instances. The details about the rules used for ranking the algorithms can be found from the ITC-2007 competition site [1]. Table 7 shows the best results obtained by the five finalists and the final rankings (the best results for each instance are indicated in bold). According to the ITC-2007 competition rules, our ATS algorithm is the second place winner[3], just behind the algorithm presented in [23].

Let us give two final comments. First, the best results shown at the competition web site are slightly worse than those reported in this paper (true not only for our ATS algorithm, but also for the winner algorithm in [23]). This can be easily explained by the fact

[1] Our best results are available on the web site [2].
[2] These previous best known results are up to the date of November 28, 2008.

[3] See http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm.

Table 6
Best results and comparison with other algorithms under relaxed stop condition.

Instance	ATS	Schaerf	Müller	Best known [2]
test1	**224**	234	–	234
test2	*16*	16	–	*16**
test3	**73**	82	–	82
test4	**73**	130	–	130
comp01	*5*	5	5	*5**
comp02	**29**	56	35	33
comp03	*66*	79	66	*66*
comp04	*35*	38	35	*35**
comp05	**292**	316	298	298
comp06	*37*	55	37	*37*
comp07	13	26	14	**7**
comp08	39	42	38	**38**
comp09	**96**	104	100	99
comp10	10	19	7	**7**
comp11	*0*	0	0	*0**
comp12	**310**	342	320	320
comp13	**59**	72	61	60
comp14	*51*	57	53	*51**
comp15	**68**	79	70	70
comp16	**23**	46	30	28
comp17	**69**	88	70	70
comp18	**65**	75	75	75
comp19	*57*	64	57	57
comp20	22	32	22	**17**
comp21	93	107	89	**89**

Table 7
Competition results of ITC-2007: best results on all the 21 competition instances.

Instance	Müller	Lü & Hao	Atsuta	Geiger	Clark
comp01	**5**	**5**	**5**	**5**	10
comp02	51	55	**50**	111	111
comp03	84	**71**	82	128	119
comp04	37	43	**35**	72	72
comp05	330	**309**	312	410	426
comp06	**48**	53	69	100	130
comp07	**20**	28	42	57	110
comp08	41	49	**40**	77	83
comp09	109	**105**	110	150	139
comp10	**16**	21	27	71	85
comp11	**0**	**0**	**0**	**0**	3
comp12	**333**	343	351	442	408
comp13	**66**	73	68	622	113
comp14	59	**57**	59	90	84
comp15	84	**71**	82	128	119
comp16	**34**	39	40	81	84
comp17	**83**	91	102	124	152
comp18	83	69	**68**	116	110
comp19	**62**	65	75	107	111
comp20	**27**	47	61	88	144
comp21	**103**	106	123	174	169
rank	12.9	16.7	17.6	38.2	42.2

that the ITC-2007 competition ranking was based only on 10 independent runs while in this paper and [23] much more runs (100) are used. With the same timeout and given the stochastic nature of these algorithms, more runs would lead to better "best" results (f_{best}). Moreover, 10 runs may not be sufficient for reliable statistics (average, standard deviation). This is why 100 runs were preferred in this paper. Second, the competition results show that the difference between any pair of the first three best ranked algorithms is relatively small, meaning probably that they have fundamentally very similar search powers.

5. Analysis and discussion

We turn now our attention to the second objective of the paper, i.e., to analyze some important features of the proposed ATS algorithm. In this section, we attempt to answer a number of important questions: Why do we combine the two neighborhoods and why do we combine them in a token-ring way? Whether the new proposed double Kempe chains neighborhood is really interesting? How about the importance of the randomized penalty-guided perturbation strategy? We present below a series of experimental analysis and attempt to answer these questions.

5.1. Neighborhood combination

We present in Section 3.2.2 two different neighborhoods. In order to make out why these two neighborhoods should be combined, we carried out experiments to compare the performance of these two neighborhoods and their different combinations. In this paper, we study two ways of neighborhood combination: neighborhood union and token-ring search.

In neighborhood union (denoted by $N_1 \cup N_2$), at each iteration the neighborhood structure includes all the moves of two different neighborhoods, while in token-ring search, one neighborhood search is applied to the local minimum obtained by the previous one and this process continues until no further improvement is possible [12]. For token-ring combination, we begin the search in two ways: from N_1 and N_2, respectively, denoted by $N_1 \rightarrow N_2$ and $N_2 \rightarrow N_1$.

We apply the *steepest descent* (SD) algorithm with N_1, N_2, $N_1 \cup N_2$, $N_1 \rightarrow N_2$ and $N_2 \rightarrow N_1$ to solve the first 7 competition instances. The average soft cost and CPU time (seconds, in brackets) over 50 independent SD runs are given in Table 8. From Table 8, one clearly finds that $N_1 \rightarrow N_2$ and $N_2 \rightarrow N_1$ obtain much better results than not only the single neighborhoods N_1 and N_2 but also neighborhood union $N_1 \cup N_2$. When comparing two different ways of token-ring search $N_1 \rightarrow N_2$ and $N_2 \rightarrow N_1$, one observes that they produce similar results in terms of the solution quality. However, starting the search from the basic neighborhood N_1 costs less CPU time than from the advanced neighborhood N_2. These results encourage us to combine the two neighborhoods N_1 and N_2 in a token-ring way in our ATS algorithm and starting the search from the basic neighborhood N_1.

5.2. Importance of the double Kempe chains move

In order to evaluate whether the newly proposed double Kempe chains move is a value-added one, our second experiment is carried out to evaluate the search capability of this neighborhood move, compared with three other previously proposed ones. For this purpose, we redefine four neighborhoods as follows: neighborhood $N_1^{(a)}$ includes all feasible move of moving one lecture. Neighborhood $N_1^{(b)}$ is defined as all the feasible moves of swapping two lectures. Neighborhood $N_2^{(a)}$ consists in exchanging the hosting periods assigned to the lectures in a single Kempe chain, while neighborhood $N_2^{(b)}$ concerns exchanging lectures of two Kempe chains, see Section 3.2.2. Note that except $N_2^{(b)}$ move, the first three moves have been proposed in the previous literature [8]. It is easy

Table 8
Average soft costs for different neighborhoods and their combinations.

Instance	$\bar{f}$				
	N_1	N_2	$N_1 \cup N_2$	$N_1 \rightarrow N_2$	$N_2 \rightarrow N_1$
comp01	31 (0.1)	23 (0.1)	18 (0.2)	**16** (0.2)	18 (0.2)
comp02	186 (0.4)	143 (1.8)	134 (2.3)	**120** (1.6)	123 (1.7)
comp03	210 (0.4)	187 (1.2)	177 (2.0)	**170** (1.2)	173 (1.3)
comp04	152 (0.7)	131 (3.5)	117 (6.7)	105 (2.9)	**100** (4.0)
comp05	871 (0.4)	627 (0.4)	566 (0.5)	580 (0.9)	**522** (1.0)
comp06	197 (0.8)	162 (4.7)	151 (8.2)	**140** (3.1)	**140** (5.0)
comp07	190 (1.2)	141 (8.4)	122 (15.2)	**111** (5.7)	115 (8.0)

Table 9
Average soft costs for $N_1^{(a)}$ to $N_2^{(b)}$ over 50 independent runs.

Instance	$\bar{f}$				
	$N_1^{(a)}$	$N_1^{(b)}$	$N_2^{(a)}$	$N_2^{(b)}$	N_2
comp01	42 (0.0)	33 (0.1)	49 (0.0)	**24** (0.1)	23 (0.1)
comp02	194 (0.4)	228 (0.2)	204 (0.4)	**143** (1.4)	143 (1.8)
comp03	217 (0.4)	248 (0.2)	245 (0.3)	**193** (1.1)	187 (1.2)
comp04	153 (0.7)	199 (0.4)	194 (0.6)	**132** (3.5)	131 (3.5)
comp05	1016 (0.3)	995 (0.2)	847 (0.8)	**684** (0.4)	627 (0.4)
comp06	207 (0.7)	260 (0.4)	255 (0.7)	**158** (4.6)	162 (4.7)
comp07	203 (1.1)	247 (0.6)	230 (1.3)	**140** (8.2)	141 (8.4)

to see that our neighborhoods N_1 and N_2 can be represented as: $N_1 = N_1^{(a)} \cup N_1^{(b)}$ and $N_2 = N_2^{(a)} \cup N_2^{(b)}$.

Table 9 shows the average cost functions for the SD algorithm based on $N_1^{(a)}$ to $N_2^{(b)}$ over 50 independent runs. The averaged running times are given in parenthesis. From Table 9, it is observed that the new proposed double Kempe chain neighborhood $N_2^{(b)}$ dominates the other three ones in terms of the solution quality. Furthermore, one can easily find that neighborhood $N_2^{(b)}$ and N_2 (the last column) obtains quite similar results in terms of both solution quality and CPU time.

Let us mention that for both neighborhood combination (Section 5.1) and double Kempe chains analysis shown in this section, the same experiments have also been carried out on other instances and on our TS, ILS and ATS algorithms (see [19] for more details), which well coincide with the results here.

5.3. Analysis of penalty-guided perturbation strategy

In Section 3.3.1, we introduced a new penalty-guided perturbation strategy to destruct the current solution when a local optimum solution is reached. This strategy involves randomly selecting the *highly-penalized* lectures and top rank lectures have more chance to be selected. We believe that constraining the choices to the highly-penalized lectures is essential for the ATS algorithm.

In fact, there exist a lot of strategies to select the moved lectures and perturb the local minimum solution. In order to testify the efficiency of the proposed randomized penalty-guided perturbation approach, we compare the following three lecture selection strategies:

(a) Our penalty-guided perturbation strategy proposed in Section 3.3.1, called *randomized penalty-guided perturbation* (*RPGP*).
(b) The moved lectures are always the first η (η is perturbation strength) highly-penalized ones, called *intensive penalty-guided perturbation* (*IPGP*).
(c) The moved lectures are randomly selected from all the lectures, called *random perturbation* (*RP*).

Keeping other ingredients unchanged in our ATS algorithm, we tested the above three lecture selection strategies with the first 14 competition instances under the competition timeout stop condition. Fig. 2 shows the average soft costs of these three strategies over 50 independent runs. In order to compare the influence of these three perturbation techniques, we performed a 95% confidence t-test to compare *RPGP* with *IPGP* and *RS*. We found that for 11 (respectively 8) out of the 14 instances, the difference of the computational results obtained by *RPGP* and *RS* (respectively *IPGP*) is statistically significant. These results highlight the importance of the penalty-guided perturbation strategy as well as implying that always restricting moved lectures to high penalized ones is too intensive such that the search may fall easily into a previous local optimum.

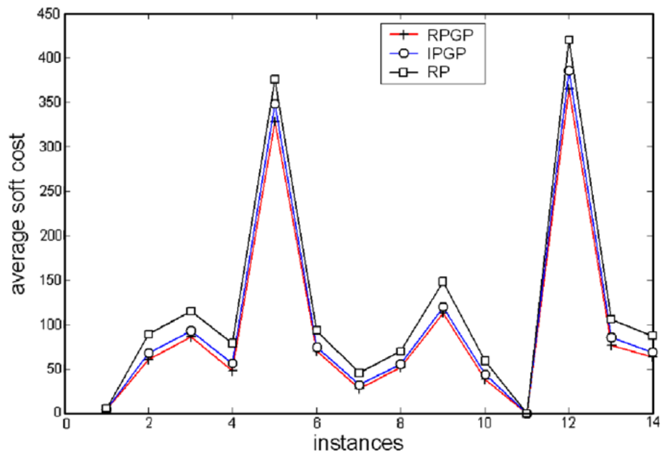

Fig. 2. Average soft costs for perturbation strategies *RPGP*, *IPGP* and *RS*.

On the other hand, from the computational results of TS and ILS reported in Table 4, we can clearly find that ILS with the penalty-guided strategy even outperforms TS (without perturbation) for almost all the 25 instances. This convinces us again that constraining the choice to highly-penalized lectures is essential because it is these lectures that contribute strongly to constraint violations (and the cost function). Meanwhile, we should also notice that the random selection strategy makes our perturbation strategy much more flexible than the intensive penalty-guided strategy.

6. Conclusions

In this paper, we dealt with the curriculum-based course timetabling problem which constitutes the track 3 of the Second International Timetabling Competition. In addition to providing a first mathematical formulation of the problem, we presented a hybrid Adaptive Tabu Search algorithm to solve this difficult problem. The proposed ATS algorithm follows a general framework composed of three phases: initialization, intensification and diversification.

The proposed algorithm integrates a number of original features. First, we have introduced the double Kempe chains neighborhood structure for the CB-CTT problem and a special technique for reducing the size of this time-consuming yet effective neighborhood. Second, we proposed a randomized penalty-guided perturbation strategy to perturb current solution when TS reaches the local optimum solution. Last but not least, for the purpose of providing the search with a continuous tradeoff between intensification and diversification, we have proposed a mechanism for adaptively adjusting the depth of TS and perturbation strength.

We assessed the performance of the proposed ATS algorithm on two sets of 25 problem instances under the ITC-2007 timeout condition. For these instances, we showed the advantageous merits of the proposed ATS algorithm over TS and ILS alone, as well as five other reference algorithms which include the winner algorithm of ITC-2007 [23] and the current best known results. We also presented the best solutions we have found so far when the competition stop condition is relaxed. These results were compared with the current best known results reported on the web site [2]. The above computational results and comparisons showed the efficiency of our ATS algorithm.

Our second contribution in this paper is to investigate several essential parts of our proposed algorithm. We first carried out experiments to demonstrate that a token-ring way of combination is appropriate for the two different neighborhoods N_1 and N_2. In addition, we carried out experiments to show that the proposed double Kempe chains move outperforms three other previous ones in the literature. Finally, we have demonstrated that our randomized penalty-guided perturbation strategy is essential for our ATS algorithm.

Let us finally comment that although the focus of this work is to propose a particular algorithm for solving a course timetabling problem, the basic ideas are quite general and would be applicable to other similar problems.

Acknowledgements

We would like to thank the ITC-2007 organizers for defining this challenging problem and providing the formidable research opportunity. Our special thanks go to the anonymous referees for their helpful comments and questions. This work was partially supported by "*Angers Loire Métropole*" and the Region of "*Pays de la Loire*" within the MILES and RADAPOP Projects.

References

[1] Second International Timetabling Competition (ITC-2007), Track 3: Curriculum-Based Course Timetabling. <http://www.cs.qub.ac.uk/itc2007/>.
[2] <http://tabu.diegm.uniud.it/ctt/index.php>.
[3] S.M. Al-Yakoob, H.D. Sherali, Mathematical programming models and algorithms for a classfaculty assignment problem, European Journal of Operational Research 173 (2) (2006) 488–507.
[4] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22 (4) (1979) 251–256.
[5] E.K. Burke, B.L. MacCarthy, S. Petrovic, R. Qu, Multiple-retrieval case-based reasoning for course timetabling problems, Journal of Operations Research Society 57 (2) (2006) 148–162.
[6] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper heuristic for timetabling problems, European Journal of Operational Research 176 (2007) 177–192.
[7] E.K. Burke, J. Marecek, A.J. Parkes, H. Rudová, A branch-and-cut procedure for the udine course timetabling problem, in: Proceedings of the Seventh PATAT Conference, 2008. <http://www.cs.nott.ac.uk/~jxm/timetabling/patat2008-paper.pdf>.
[8] M. Chiarandini, M. Birattari, K. Socha, O. Rossi-Doria, An effective hybrid algorithm for university course timetabling, Journal of Scheduling 9 (2006) 403–432.
[9] M. Clark, M. Henz, B. Love, QuikFix: A repair-based timetable solver, in: Proceedings of the Seventh PATAT Conference, 2008. <http://www.comp.nus.edu.sg/~henz/publications/ps/PATAT2008.pdf>.
[10] P. De Causmaecker, P. Demeester, G. Vanden Berghe, A decomposed metaheuristic approach for a real-world university timetabling problem, European Journal of Operational Research 195 (2009) 307–318.
[11] F. De Cesco, L. Di Gaspero, A. Schaerf, Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, and results, in: Proceedings of the Seventh PATAT Conference, 2008. <http://tabu.diegm.uniud.it/ctt/DDS2008.pdf>.
[12] L. Di Gaspero, A. Schaerf, Neighborhood portfolio approach for local search applied to timetabling problems, Journal of Mathematical Modeling and Algorithms 5 (1) (2006) 65–89.
[13] L. Di Gaspero, B. McCollum, A. Schaerf, The Second International Timetabling Competition (ITC-2007): Curriculum-based Course Timetabling (Track 3), Technical Report, 2007. <http://www.cs.qub.ac.uk/itc2007/curriculmcourse/report/curriculumtechreport.pdf>.
[14] M.J. Geiger, An application of the threshold accepting metaheuristic for curriculum based course timetabling, in: Proceedings of the Seventh PATAT Conference, 2008. <http://arxiv.org/PS_cache/arxiv/pdf/0809/0809.0757v1.pdf>.
[15] F. Glover, M. Laguna, Tabu Search, Kluwer Academic, Boston, 1997.
[16] G. Lach, M.E. Lübbecke, Curriculum based course timetabling: Optimal solutions to the udine benchmark instances, in: Proceedings of the Seventh PATAT Conference, 2008. <http://www.math.tu-berlin.de/luebbeck/papers/udine.pdf>.
[17] R. Lewis, B. Paechter, New crossover operators for timetabling with evolutionary algorithms, in: A. Lofti, (Ed.), The Fifth International Conference on Recent Advances in Soft Computing RASC2004, Nottingham, England, 2004, pp. 189–194.
[18] R. Lewis, A survey of metaheuristic-based techniques for university timetabling problems, OR Spectrum 30 (1) (2008) 167–190.
[19] Z. Lü, J.K. Hao, F. Glover, Neighborhood analysis: A case study on curriculum-based course timetabling, Technical Report, LERIA, University of Angers, France, 2009.
[20] H.R. Lourenco, O. Martin, T. Stützle, Iterated local search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Meta-heuristics, Springer-Verlag, 2003, pp. 321–353.
[21] B. McCollum, A perspective on bridging the gap between theory and practice in university timetabling, in: E.K. Burke, H. Rudová (Eds.), Proceedings of the Sixth PATAT Conference, LNCS, vol. 3867, 2007, pp. 3–23.
[22] C. Morgenstern, H. Shapiro, Coloration neighborhood structures for general graph coloring, in: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1990, pp. 226–235.
[23] T. Müller, ITC2007 solver description: A hybrid approach, in: Proceedings of the Seventh PATAT Conference, 2008. <http://www.unitime.org/papers/itc2007.pdf>.
[24] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall Inc., 1982.
[25] R.V. Rasmussen, M.A. Trick, Round robin scheduling – a survey, European Journal of Operational Research 188 (3) (2008) 617–636.
[26] M.M. Rodrigues, C.C. de Souza, A.V. Moura, Vehicle and crew scheduling for urban bus lines, European Journal of Operational Research 170 (3) (2006) 844–862.
[27] R. Santiago-Mozos, S. Salcedo-Sanz, M. DePrado-Cumplido, C. Bousono-Calzon, A two-phase heuristic evolutionary algorithm for personalizing course timetables: A case study in a spanish university, Computers and Operations Research 32 (2005) 1761–1776.
[28] J.C. Setubal, Sequential and Parallel Experimental Results with Bipartite Matching Algorithms, Technical Report EC-96-09, Institute of Computing, University of Campinas, Brasil, 1996.
[29] J. Thompson, K. Dowsland, A robust simulated annealing based examination timetabling system, Computer and Operations Research 25 (1998) 637–648.
[30] M. Vasquez, J.K. Hao, A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite, Computational Optimization and Applications 20 (2) (2001) 137–157.
[31] G.M. White, B.S. Xie, S. Zonjic, Using tabu search with longer-term memory and relaxation to create examination timetables, European Journal of Operational Research 153 (16) (2004) 80–91.