Adaptive Tabu Search for Course Timetabling ⋆

Zhipeng Lü [a,*], Jin-Kao Hao [a]

[a] *LERIA, Université d'Angers, 2 boulevard Lavoisier, 49045 Angers Cedex 01, France*

European Journal of Operational Research 200(1): 235-244, 2010.

Abstract

This paper presents an Adaptive Tabu Search algorithm (denoted by ATS) for solving the problem of curriculum-based course timetabling. The proposed algorithm follows a general framework composed of three phases: initialization, intensification and diversification. The initialization phase is primarily aimed to construct a feasible initial timetable using a fast greedy heuristic. When a feasible initial assignment is reached, an adaptively combined intensification (Tabu Search) and diversification (Perturbation Operator from Iterated Local Search) phase is used in order to reduce the number of soft constraint violations without breaking hard constraints any more. The proposed ATS algorithm integrated several distinguished features including an original double Kempe chains neighborhood structure, a penalty-guided perturbation approach and a mechanism for dynamically integrating tabu search with perturbation. Computational results indicate that better solutions can be found compared with Tabu Search and Iterated Local Search alone, as well as another reference algorithm. This paper also shows an analysis to explain which are the essential ingredients of the proposed ATS algorithm.

Key words: Hybrid Heuristic, Timetabling, Tabu Search, Perturbation Operator, Iterated Local Search

1. Introduction

As a problem that most universities must face year after year, timetabling has become an area of increasing interest in the community of both research and practice in recent decades. In essence, it consists of assigning a number of events, each with a number of features, to a limited number of timeslots and rooms subject to certain (hard and soft) constraints. Typical cases in this area include educational timetabling [12], sport timetabling [32], employee timetabling [13], transport timetabling [3] and so on. In this paper, we consider one of the problems in the category of educational timetabling.

Educational timetabling problems can be generally classified into two categories: exam timetabling and course timetabling. The later can be further divided into two sub-categories: post enrollment-based course timetabling and curriculum-based course timetabling (CB-CTT). 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, the formulation of which was recently proposed as the third track of the Second International Timetabling Competition (ITC–2007) [?]. This competition is aimed to close the gap between

⋆ This algorithm is ranked as one of the five finalists for the track 3 of the Second International Timetabling Competition (ITC–2007).

* Corresponding author.

Email addresses: lu@info.univ-angers.fr (Zhipeng Lü), hao@info.univ-angers.fr (Jin-Kao Hao).

research and practice within the area of educational timetabling [23].

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

The general university timetabling problem is known to be difficult and has been proved to be NP-hard [14,17]. In this context, exact solutions would be only possible for problems of limited sizes. Instead, heuristic algorithms based on metaheuristics have shown to be a highly effective approach to this kind of problems. Examples of these algorithms include graph coloring heuristics [5,7,9], tabu search [26,30,36], simulated annealing [2,34], evolutionary algorithms [6,15,29], constraint based approach [16,24], GRASP [10,33], case-based reasoning [4] and so on. Interested readers are referred to [8,21,31] for a comprehensive survey of the automated approaches for university timetabling presented in recent years.

The objective of this paper is two-fold: a three-phases solution algorithm for solving the CB-CTT problem was presented and some essential ingredients of the proposed algorithm were carefully investigated. The proposed ATS algorithm follows a general framework composed of three phases: initialization, intensification and diversification. The initialization phase is primarily aimed to construct a feasible initial timetable using a fast greedy heuristic. When a feasible initial assignment is reached, the intensification and diversification phases are adaptively combined in order to reduce the number of soft constraint violations without breaking hard constraints any more. The performance of the proposed hybrid algorithm was assessed on a set of 4 instances used in the literature and a set of 14 public competition instances from the ongoing Second International Timetabling Competition, showing very competitive results.

As the second objective of this paper, we carefully investigated several important features of the proposed algorithm. 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.

The rest of this paper is organized as follows. Section 2 describes the mathematical formulation of the CB-CTT problem. Section 3 introduces the main idea and the general framework of the ATS algorithm. Following that, Section 4 presents the initial solution generator based on two greedy heuristics. Section 5 describes in details the basic search engine of our ATS algorithm—Tabu Search. Section 6 depicts the penalty-guided perturbation operator and explains how TS and perturbation is dynamically combined. In Section 7 the computational results of the algorithm are presented and discussed. Section 8 presents investigations on several essential parts of the proposed ATS algorithm. Eventually in Section 9 we draw some conclusions.

2. Curriculum-Based Course Timetabling

2.1. *Problem Description*

The CB-CTT problem consists of 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. A feasible timetable is one in which all lectures have been scheduled at a timeslot and a room, so that the hard constraints $H_1 \sim H_4$ are satisfied. In addition, a feasible timetable satisfying the four hard constraints incurs a penalty cost for the violations of the four soft constraints $S_1 \sim S_4$. 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**: All lectures of a course must be scheduled to 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., any period cannot have an overlapping of students or 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 num-

ber 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 at 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.

Given the above description of this problem and in order to avoid any confusion, we present below a first 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. Without leading to confusion, we do not distinguish between *lecture* and *course* in the following context. A period is a pair composed of a day and a timeslot and p periods are distributed in d week days and h daily timeslots, i.e., $p = d * h$. 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.

For the solution representation, we choose a direct solution representation to make things as simple as possible. A candidate solution consists of $p * m$ 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 at period t_i and room r_j, then $x_{i,j}$ takes the value "*null*". 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 constant symbols and variable definitions are presented in tables 1 and 2.

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 cost for the four soft constraints are as follows:

Table 1
Table of symbols

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, $p = d * h$
s	the total number of curricula
C	the set of all courses, $\|C\| = n$
R	the set of all rooms, $\|R\| = m$
T	the set of all periods, $\|T\| = p$
CR	the set of all curricula, $\|CR\| = s$
l_i	the total number of lectures of course c_i
l	the total number of lectures, $l = \sum_1^n l_i$
std_i	the number of students attending course c_i
tc_i	the label of the teacher instructing course c_i
md_i	the number of minimum working days of course c_i
cap_j	the room capacity of room r_j
cr_k	the kth curriculum including a set of courses
$uav_{i,j}$	whether course c_i is unavailable at period t_j. $uav_{i,j} = 1$ if it is unavailable, $uav_{i,j} = 0$ otherwise

Table 2
Table of variables

Variables	Description
$x_{i,j}$	the label of the course assigned at period t_i and room r_j
$nr_i(X)$	the number of rooms occupied by course c_i for a candidate solution X
$nd_i(X)$	the number of working days that course c_i takes place at for a candidate solution X
$app_{k,i}(X)$	whether curriculum cr_k appears at period t_i for a candidate solution X, $app_{k,i}(X) = 1$ when any course in curriculum cr_k is scheduled at period t_i, $app_{k,i}(X) = 0$ otherwise.

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

  $$\sum_{i,j} sl_k(x_{i,j}) = l_k$$

  where

  $$sl_k(x_{i,j}) = \begin{cases} 1, & \text{if } x_{i,j} = c_k; \\ 0, & \text{otherwise.} \end{cases}$$

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

  $$(\forall cr_q, c_u \notin cr_q \vee c_v \notin cr_q) \wedge (tc_u \neq tc_v)$$

- H_4. **Availability**: $\forall x_{i,j} \in X, x_{i,j} = c_k$,

  $$uav_{k,i} = 0$$

- S_1: **Room Capacity**: $\forall x_{i,j} \in X, x_{i,j} = c_k$,

$$f_1(x_{i,j}) = \begin{cases} \alpha_1 \cdot (std_k - cap_j), & \text{if } std_k > cap_j; \\ 0, & \text{otherwise.} \end{cases}$$

- S_2: **Room Stability**: $\forall c_i \in C$,

$$f_2(c_i) = \alpha_2 \cdot (nr_i(X) - 1)$$

- S_3: **Minimum Working Days**: $\forall c_i \in C$,

$$f_3(c_i) = \begin{cases} \alpha_3 \cdot (md_i - nd_i(X)), & \text{if } nd_i(X) < md_i; \\ 0, & \text{otherwise.} \end{cases}$$

- S_4: **Curriculum Compactness**: $\forall x_{i,j} \in X$, $x_{i,j} = c_k$,

$$f_4(x_{i,j}) = \alpha_4 \cdot \sum_{cr_q \in CR} c\_cr_{k,q} \cdot iso_{q,i}(X)$$

where

$$c\_cr_{k,q} = \begin{cases} 1, & \text{if } c_k \in cr_q; \\ 0, & \text{otherwise.} \end{cases}$$

$$iso_{q,i}(X) = \begin{cases} 1, & \text{if } (i\%h = 1 \vee app_{q,i-1}(X) = 0) \\ & \wedge(i\%h = 0 \vee app_{q,i+1}(X) = 0); \\ 0, & \text{otherwise.} \end{cases}$$

One observes that in the S_4 soft constraint function, the calculation is only limited within the same day. $iso_{q,i}(X) = 1$ means that curriculum cr_q in the $[i/h]$th day is isolated, i.e., there is no any course in the curriculum cr_q scheduled adjacent (before or after) to the timeslot $i\%h$ in the $[i/h]$th day. More specifically, curriculum cr_q does not appear before (after) period t_i means that t_i is the first (last) timeslot of a working day or cr_q does not appear at t_{i-1} (t_{i+1}).

Note that α_1, α_2, α_3 and α_4 are the penalties associated to each of the soft constraints. In this problem formulation, they are set as:

$$\alpha_1 = 1, \alpha_2 = 1, \alpha_3 = 5, \alpha_4 = 2$$

It is obvious that the soft constraints S_1 and S_2 are uniquely room-related costs while S_3 and S_4 are period-related ones. This feature allows us to deal with the incremental cost of neighborhood moves in a more flexible way (as described in section 5.3 and 8.2).

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 $f(X^*) \leq f(X)$ for all X in the feasible search space.

$$f(X) = \sum_{x_{i,j} \in X} f_1(x_{i,j}) + \sum_{c_i \in C} f_2(c_i) + \sum_{c_i \in C} f_3(c_i) + \sum_{x_{i,j} \in X} f_4(x_{i,j}) \quad (1)$$

3. General Solution Framework

In this paper, we present a hybrid metaheuristic algorithm (Adaptive Tabu Search, denoted by ATS) for solving the curriculum-based course timetabling. The proposed algorithm follows a general framework composed of three phases: initialization, intensification and diversification. The initialization phase is primarily aimed to construct a feasible initial timetable using a fast greedy heuristic. When a feasible initial assignment is reached, the adaptively combined intensification and diversification phase is used in order to reduce the number of soft constraint violations without breaking hard constraints any more. The intensification phase employs a Tabu Search algorithm [20] while the diversification phase is based on a penalty-guided perturbation operator borrowed from Iterated Local Search [22].

In the initialization phase, a feasible solution is constructed from empty using a greedy heuristic. At each time, one appropriate lecture of a course is selected and assigned to a period and a room. For this purpose, we proposed two greedy heuristic rules for course selection and period-room assignment respectively. Notice that soft constraints are also considered in this procedure when deciding a period-room pair for a selected lecture.

For the intensification phase, the basic search engine is based on Tabu Search (TS) [20], in which we introduce two distinct neighborhood structures: one swaps two lectures or moves one lecture to a free position, the other is defined by single and double Kempe chain interchanges concerning two distinct periods. Our TS algorithm is implemented by combining these two neighborhoods in a token-ring way [19].

When the TS cannot improve the solution quality any more, a diversification phase base on Iterated Local Search [22] is triggered to help escape from local optimum solution. In order to not only inherit the essential parts of the current local optimum solution obtained but also move toward new promising regions of the search space, we propose a randomized penalty-guided perturbation operator to destruct the reached local optimum. Thus, a new TS phase starts then with the perturbed solution.

In order to provide the search with a continuous tradeoff between intensification and diversification, a mechanism is proposed to adaptively adjust the strength of TS and perturbation, which constitutes the main skeleton of our Adaptive Tabu Search (ATS) algorithm.

4. Initial Solution

The first phase of our algorithm aims to generate a feasible initial solution. This is achieved by a sequential greedy heuristic starting from an empty timetable, where assignments are constructed by inserting one appropriate lecture into the the timetable at each time. At each step, two distinct operations are carried out: one is to select a still unassigned lecture of a course, the other is to determine a period-room pair for this lecture. To this end, two heuristic rules are utilized, where lectures are selected and scheduled in a dynamic way primarily based on an idea of *least period availability*. It should be mentioned that, in our initial solution generator, we also take into account the soft constraints by introducing a weighted heuristic function involving all hard and soft constraint factors. Before describing the greedy heuristics, it is necessary to give some basic definitions as follows.

Definition 1. feasible timetable: a feasible timetable X is a complete timetable assignment which satisfies all the four hard constraints $H_1 \sim H_4$.

Definition 2. partial feasible timetable: A partial timetable, denoted by $\widetilde{X}$, is such that only a part of the lectures are scheduled to the timetable without violating any hard constraint.

Definition 3. feasible lecture insertion: Given a partial feasible timetable, a feasible lecture insertion consists of choosing one unassigned lecture of course c_i and scheduling it to a period t_j and a room r_k such that no hard constraint is violated, denoted by $< c_i, t_j, r_k >$.

Definition 4. available period: Given a partial feasible timetable $\widetilde{X}$, period t_j is available for course c_i means that there exists at least one room at period t_j such that course c_i can be assigned without violating any hard constraint.

In order to describe our algorithm more clearly, the following notations are presented under a partial feasible timetable $\widetilde{X}$:

- $apd_i(\widetilde{X})$: the total number of available periods for course c_i under $\widetilde{X}$;
- $aps_i(\widetilde{X})$: the total number of available positions (period-room pairs) for course c_i under $\widetilde{X}$;
- $nl_i(\widetilde{X})$: the number of unassigned lectures of course c_i under $\widetilde{X}$;
- $uac_{i,j}(\widetilde{X})$: the total number of lectures of unfinished courses who become unavailable at period t_j after assigning one lecture of course c_i at period t_j.

Under any partial feasible timetable $\widetilde{X}$, we attempt to choose one lecture of a course from all the unfinished courses, i.e. having unscheduled lectures, according to the following heuristic order (**HR1**):

(i) choose the course with the smallest value of $apd_i(\widetilde{X})/\sqrt{nl_i(\widetilde{X})}$;

(ii) if there are multiple courses with the same smallest values, then, choose the course with the smallest value of $aps_i(\widetilde{X})\sqrt{nl_i(\widetilde{X})}$;

(iii) if there are still more than one course with the same smallest values, then choose the course with the maximum number of $conf_i$, where $conf_i$ is the number of courses that share common students or teacher with course c_i. Ties are broken by following the label order.

In this heuristic, the courses with a small number of available periods and a large number of unassigned lectures have priority. The rationale for this heuristic is the following. On the one hand, one course which has a small number of available periods naturally has fewer choices to be assigned than courses having many available periods. On the other hand, it is reasonable to give priority to the course with a large number of left lectures. When this heuristic rule cannot distinguish two or more courses, the number of available positions (period-room pairs) and the number of conflicting courses are taken into account to distinguish them.

Based on a dynamic selection procedure, the proposed course selection heuristic **HR1** is similar to (but not exactly the same as) the least Saturation Degree first heuristic used in [5]. However, **HR1** is quite different from most of the previous fixed order heuristics [12].

Once we have chosen one lecture of a course to assign (suppose c_i^* is chosen), we want to select a period among all available ones that is least likely to be used by other unfinished courses at later steps.

To this end, we propose the following heuristic rule (**HR2**) for the period-room pair assignment: for each available period-room pair (t_j and r_k), we try to choose the pair with the smallest value of the following weighted function:

$$g(j,k) = k_1 \cdot uac_{i^*,j}(\widetilde{X}) + k_2 \cdot \Delta f_s(i^*, j, k) \qquad (2)$$

where $\Delta f_s(i^*, j, k)$ is the soft constraint penalties incurred by the feasible lecture insertion $< c_i^*, t_j, r_k >$; k_1 and k_2 are the coefficients related to hard and soft constraints, respectively. In practice, we have found that the following parameter values work well over a large class of instances: $k_1 = 1.0$, $k_2 = 0.5$. In this function, $uac_{i^*,j}$ denotes the total number of lectures of unfinished courses that become unavailable at period t_j after assigning one lecture of course c_i^* at period t_j, implying the influence of the feasible lecture insertion $< c_i^*, t_j, r_k >$ to the period availability of other unfinished courses. This means that there are $uac_{i^*,j}$ lectures of unfinished courses that originally can be assigned at period t_j, but will become impossible to assign if the feasible lecture insertion $< c_i^*, t_j, r_k >$ is executed. Therefore, feasible lecture insertions with small values of $uac_{i^*,j}$ are highly favored.

Given these definitions, notations and heuristic rules, our initial generation algorithm is described in algorithm 1.

Algorithm 1 Pseudo-code of the initial solution heuristic

- **Input**: I, an instance of CB-CTT
- **Output**: X_0, a feasible solution or a partial feasible solution
- set $\widetilde{X} = null$ and initialize the set LC of all unassigned lectures
- **repeat**
  - choose one unassigned lecture of a course (c_i) from LC according to **HR1**
  - assign the lecture to a period-room pair ($t_j - r_k$) based on **HR2**
  - implement the feasible lecture insertion $< c_i, t_j, r_k >$ and update $\widetilde{X}$
  - remove one lecture of course c_i from LC
- **until** (LC is empty or no any feasible lecture insertion is available)

In our initial solution generator, we dynamically determine the order of the lecture to be assigned according to the principle of *least period availability*, rather than using a fixed order as previous initial solution heuristics of the literature [12]. Under any feasible partial timetable with some lectures assigned, we use a heuristic function to examine every period-room pair by calculating its goodness. Among all possible period-room pairs, we select the lecture insertion with the smallest value in equation (2), in which the effectiveness of the proposed heuristic allows us to take into account the soft constraints as well. After assigning one lecture, the current partial feasible timetable is updated and the values of all possible lecture insertions are recalculated for all the unassigned lectures. Such a process continues until all the lectures are successfully assigned or the number of possible lecture insertions becomes zero.

5. Tabu Search Algorithm

In this section, we focus on the basic search engine of our ATS algorithm—Tabu Search. As a well-known metaheuritic that has proven to be successful in solving various combinatorial optimization problems [20], Tabu Search explores the search space by repeatedly replacing the current solution with a non-recently visited neighboring solution even if the later is worse than the current solution. It is based on the belief that intelligent searching should be systematically based on adaptive memory and learning. Comparing with the standard local search, TS introduces the notion of *tabu list* to forbid the previously visited moves in order to avoid possible cycling and to allow the search to go beyond local optima.

Our TS procedure exploits two neighborhoods (denoted by N_1 and N_2, see below) in a token-ring way [18]. More precisely, we start the TS procedure with one neighborhood. When the search ends with 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 a TS phase is achieved. In our case, the TS procedure begins from the basic neighborhood N_1 and then the advanced neighborhood N_2: $N_1 \rightarrow N_2 \rightarrow N_1 \rightarrow N_2$...

5.1. *Search Space and Evaluation Function*

Once a feasible timetable that satisfies all the hard constraints is reached, our intensification phase (TS algorithm) is aimed to optimize the soft constraint cost function without breaking hard constraints any more (formula (1)). 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).

5.2. *Neighborhood Structure*

It is widely believed that one of the most important features of a local search based 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 $X \bigoplus mv$. Let $M(X)$ be the set of all possible moves which can be applied to X and do not create any infeasibility, then the neighborhood of X is defined by: $N(X) = \{X \bigoplus mv | mv \in M(X)\}$. For the CB-CTT problem, we use two distinct *move*s denoted by *SimpleSwap* and *KempeSwap*. Respectively, two neighborhoods denoted by N_1 and N_2 are defined as follows, where only the moves producing those neighbors that do not incur any violation of the hard constraints are accepted.

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

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 form 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 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 $(t_i \backslash (K_1 \cup K_2)) \cup (t_j \cap (K_1 \cup K_2))$ and t_j with $(t_j \backslash (K_1 \cup K_2)) \cup (t_i \cap (K_1 \cup K_2))$. Note that in the definition of N_2 at least three courses are involved, i.e., $|K_1| + |K_2| \geq 3$. For instance, figure 1 depicts a subgraph deduced by two periods t_i and t_j and there are five Kempe chains: $K_a = \{c_{i1}, c_{i2}, c_{j1}, c_{j2}\}$, $K_b = \{c_{i5}\}$, $K_c = \{c_{j4}\}$, $K_d = \{c_{i3}, c_{i6}, c_{j3}\}$ and $K_e = \{c_{i4}, c_{j5}, c_{j6}\}$. In this example, each room at periods t_i and t_j has one lecture. If we swap two Kempe chains K_d and K_e, a *KempeSwap* produces an assignment by moving $\{c_{i3}, c_{i4}, c_{i6}\}$ to t_j and $\{c_{j3}, c_{j5}, c_{j6}\}$ to t_i.

Note that in our *KempeSwap*, one of the swapping Kempe chains can be empty, i.e., we add a new empty Kempe chain $K_f = \varnothing$. In this case, the move of *KempeSwap* degenerates into a single Kempe chain interchange. Formally, it means replacing t_i with $(t_i \backslash K) \cup (t_j \cap K)$ and t_j with $(t_j \backslash K) \cup (t_i \cap K)$ where K is the non-empty Kempe chain [12]. For example, in figure 1, if we exchange the courses of the Kempe chain K_a, it produces an assignment by moving $\{c_{i1}, c_{i2}\}$ to t_j and $\{c_{j1}, c_{j2}\}$ 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 [10,12,15,24,34].

Once courses are scheduled to periods, the room assignment can be done by solving a bipartite matching problem [27], where both heuristic and exact algorithms can be employed. In this paper, we implement an exact algorithm–the augmenting path algorithm introduced in [28], which runs in $O(|V||C|)$. In consideration of the high computational effort of this matching algorithm, we should try to use it as few as possible. For this purpose, we propose a special technique to estimate the goodness of a move without actually calling this matching algorithm, as shown in subsections 5.3 and 8.2.

Since *KempeSwap* can be considered as an ex-

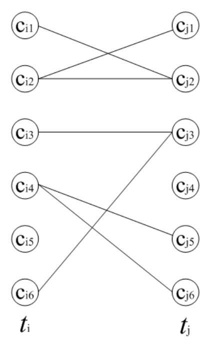

Fig. 1. Kempe chain illustrations

tended version of swapping two lectures (and afterward several other related lectures in the specified Kempe chain(s) being moved), the size of N_2 is bounded by $O(l*(l+p))$, where the size of double Kempe chains interchange is bounded by $O(l*l)$ and the size of single Kempe chain interchange is bounded by $O(l*p)$.

In order to maintain the feasibility of the Kempe chain neighborhood solution, another important factor also needs to be considered, i.e., the number of courses in each period (after Kempe chain exchange) cannot exceed the number of available rooms. For example, in figure 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 break the above-mentioned violation 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 taken into account, the *room allocation violation* is relaxed such that a large number of feasible moves can be generated. For instance, in figure 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_d and K_e. In a word, the introduction of the double Kempe chains interchanges allows us to consider more neighborhood moves in a flexible way than the previous single Kempe chain interchange.

It should be noted that our proposed neighborhoods N_1 and N_2 are quite different from the previous ones introduced in [11,18,19]. In their work, two basic neighborhood moves were defined: one is to simply change the period assigned to a lecture of a given course, while the other is to change the room assigned to a lecture of a given course. One observes that these two neighborhood moves are the subset of our basic neighborhood N_1. It should be mentioned that except the double Kempe chains interchange, other moves (one lecture move, two lectures swap and single Kempe chain interchange) are not completely new and have been proposed for solving other timetabling problems in the literature in recent years [12,21]. However, we will show in Section 8.4 that the proposed double Kempe chains move is much more powerful.

5.3. *Incremental Evaluation and Neighborhood Reduction*

Our basic search procedure is based on TS, which employs an aggressive search strategy to exploit its neighborhood, i.e., at each iteration, all the candidate neighbors of the current solution are examined and the best non-tabu one is chosen. In order to evaluate the neighborhood in an efficient way, we use an incremental evaluation technique. The main idea is to keep 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, as mentioned above, the move evaluation of the advanced neighborhood N_2 needs much more computational efforts than that of N_1. 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 period related and room related 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.

In fact, we use the period related sub cost Δf_p as a goodness estimation of the Kempe move. Specifically, if the period related cost Δf_p is promising (i.e., $\Delta f_p \leq \tau$, practically τ=2 would produce 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 neighborhood solutions are thoroughly evaluated, thus allowing us to save a considerable amount of CPU time. It should be noted that the successful employment of this technique must be based on the hypothesis that the period related sub cost Δf_p is proportional to the total cost function Δf (as shown in Section 8.2).

5.4. *Tabu List Management*

Within TS, a *tabu list* is introduced to forbid the previously visited moves in order to escape from local optima. At each iteration, a best non-tabu move mv is applied to the current solution X even if $X' = X \bigoplus mv$ does not improve the solution quality.

In our TS algorithm, when moving one lecture from one position (period-room pair) to another (N_1), or from one period to another (N_2), this lecture cannot be moved back to the original position (N_1) or period (N_2) for the next tt iterations (tt is called 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 one lecture of course c_i to the position (t_j, r_k) is declared tabu. Similarly, in neighborhood N_2, moving one lecture of course c_i to period t_j is declared tabu *iff* any lecture of course c_i is moved from period t_j to another one.

Tabu tenure tt of a course c_i is tuned adaptively according to the current solution quality f and the moving frequency of lectures of course c_i, denoted by $freq(c_i)$, i.e.,

$$tt(c_i) = f + \varphi \cdot freq(c_i)$$

where φ is a parameter that lies 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 in order to escape from the local optimum trap. On the other hand, the second part of the function is proportional to the moving frequency of course c_i. The basic idea is to penalize a move which repeats too often. The coefficient φ is instance-dependent and is defined as the ratio of the number of conflicting courses of c_i over the total number of courses. It is reasonable that a course involved in a large number of conflicts has more risk to be moved than a course having fewer conflicts. This tabu tenure function is intended to explicitly guide the search to new regions of the solution space. Notice that $freq(c_i)$ is the essential part of the above tabu tenure function and the frequency-based tabu tenure technique has been widely used in previous literature, see e.g. [35].

5.5. *Aspiration Criteria and Stop condition*

Since attributes of a solution instead of solutions themselves are recorded in tabu list, sometimes a candidate solution in the tabu list would lead to a solution better than the best found so far. In this case an aspiration criterion is used to accept this solution regardless of its tabu status. Our aspiration criterion accepts a tabu move if it improves the current best solution or the set of non-tabu moves is empty in the current neighborhood.

Many stop conditions are possible for the TS algorithm, such as the fixed numbers of iterations, the maximum number of iterations without improvement in cost function and the total amount of CPU time. Since our TS is a basic search procedure and will be adaptively integrated with perturbation operators, our TS algorithm stops when the best solution cannot be improved within a given number of moves (denoted by θ) and we call this number the *depth of TS*.

5.6. *TS Algorithm Description*

Given all the components of TS, the algorithm is described in algorithm 2.

Algorithm 2 Tabu Search procedure: TS(X_0,θ)

```
Input: X_0:a feasible initial solution
           θ:the depth of TS
Output: X_best:best solution found so far
X_best ← X_0
repeat
   X* ← TS_{N_1}(X_0, θ)
   X*' ← TS_{N_2}(X*, θ/3)
   if f(X*') < f(X_best) then
      X_best ← X*'
      X_0 ← X*'
   end if
until (no improvement is obtained)
```

Because of the high computational effort in evaluating the moves of neighborhood N_2, TS uses a much smaller depth ($\theta/3$) when N_2 is used. The TS procedure described here constitutes the basic search engine of our whole ATS algorithm, where the parameter θ will be dynamically tuned according to the search history (see Section 6.2).

6. Adaptive TS: Combining TS with Perturbation

In recent decades, TS and ILS have alone proved their efficiency for solving a large number of constraint satisfaction and optimization problems [20,22]. In this paper, we consider the possibility of combining these two powerful metaheuristics in an informative way. Following the basic idea of combining the advantageous features of TS and ILS exposed in [25], we devise in this work an ATS algorithm for the CB-CTT problem whose components and mechanisms are described in the following subsections.

TS can be used with both long and short computing budgets. In general, long computing budgets would lead to better results. However, if the total computation time is limited, it would be preferred to combine short TS runs with some robust diversification operators. Interestingly, Iterated Local Search provides such diversification mechanisms to guide the search to escape from the current local optimum and move toward new promising regions in the solution space [22].

6.1. *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 destruct 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 not only inherit the essential parts of the current local optimum solution obtained but also move towards a new region of the search space, we employ a *penalty-guided* perturbation operator to perturb the reached local optimum solution. Our perturbation 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 set $q = 30$). 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 decreasing order according to their soft penalties involved. Then, totally 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 and in this paper we empirically set $\phi = 4.0$. After that, η feasible moves of *SimpleSwap* or *KempeSwap* are randomly and sequentially produced, each involving at least one of the selected η lectures.

Notice 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). In addition, the most important elements of a local minimum solution without contributing anything to the soft constraint violations will remain unchanged, which inherit the essential parts of the current local minimum solution.

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 $\eta_{min} = 4, \eta_{max} = 15$).

6.2. *ATS and Two Self-Adaptive Mechanisms*

Our ATS algorithm is based on an integration of intensification (TS) and diversification (ILS's Perturbation). 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 historical search records.

At the beginning of the search, we take a basic TS where the *depth of TS* θ is a small positive number, 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., when it is close to the current best solution ($f \leq f_{best} + 2$), the *depth of TS* 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

Algorithm 3 Adaptive Tabu Search

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

(μ=0.6). On the other hand, *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 basic TS before each perturbation, while the perturbation strength reset to η_{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 solutions are accepted as the best solution found so far.

Different stop conditions are possible for the whole ATS algorithm, such as the fixed number of TS phases, the maximum number of perturbations without improvement in the cost function, the total CPU time and so on. In this paper, we use two stop conditions as described in Section 7.

Finally, our Adaptive Tabu Search algorithm is described in algorithm 3.

Table 3
Features of the 14 competition instances

Instance	n	m	p	l	occupancy	conflicts
test1	46	12	20	207	86.25%	5.41%
test2	52	12	20	223	92.92%	5.05%
test3	56	13	20	252	96.92%	4.74%
test4	55	10	25	250	100%	4.98%
comp01	30	6	30	160	88.89%	11.49%
comp02	82	16	25	283	70.75%	7.50%
comp03	76	12	25	251	62.75%	8.33%
comp04	79	18	25	286	63.56%	5.06%
comp05	54	9	36	152	46.91%	21.10%
comp06	108	18	25	361	80.22%	5.37%
comp07	131	20	25	434	86.80%	4.46%
comp08	86	18	25	324	72.00%	4.35%
comp09	76	18	25	279	62.00%	5.75%
comp10	115	18	25	370	82.22%	5.32%
comp11	30	5	45	162	72.00%	13.10%
comp12	88	11	36	218	55.05%	14.29%
comp13	82	19	25	308	64.84%	4.76%
comp14	85	17	25	275	64.71%	7.31%

7. Experimental Results

7.1. *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 set (4 instances) was previously used in the literature for the old version of the CB-CTT problem [18,19]. The second set (14 instances) is from the Second International Timetabling Competition **??**. The main features of these instances are listed in table 3. 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 $2 \cdot n_e/n \cdot (n-1)$ where n_e represents the total number of edges connecting two conflicting courses), respectively. Other symbols are described in table 1. Except for 2 instances, neither optimal solution nor tight lower bound is known. The only available (probably very bad) lower bound is *zero* which implies the satisfaction of all the soft constraints.

Our algorithm is programmed in C and compiled using Dev C++ on a PC running Windows XP with 3.4GHz CPU and 2.0G RAM. To obtain our computational results, each instance is solved 100 times with different random seeds. In this paper, we use two stop conditions for our ATS algorithm. The first one is the timeout condition required by the ITC–2007 competition rules. The second is the fixed number of iteration moves.

Note that all the following results are obtained without tuning any parameter for different in-

Table 4
Settings of important parameters

Param.	Description	Values or Updating
θ_0	basic depth of TS	10
μ	increase speed of θ	0.6
θ	depth of TS	$\theta = (1 + \mu) \cdot \theta$
ξ	non-improvement TS phases	$\xi = \xi + 1$
η_{min}	basic perturbation strength	4
η_{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	candidate number of perturbation lectures	30
ϕ	importance factor for perturbation lecture selection	4.0
τ	reduction cutoff for N_2	2

stances, i.e., all the parameters used in our algorithm are fixed or dynamically tuned during the problem solving. 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 4 gives the descriptions and settings of the important parameters used in our ATS algorithm.

7.2. *Results Using ITC–2007 Rules*

Our first experiment aims to evaluate the ATS algorithm on the 4 previous instances and 14 public competition instances of the ITC–2007, by comparing its performance with its two basic components (TS and ILS) and another reference algorithm in [11]. 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. The stop condition is just the timeout condition required by the ITC–2007 competition rules. On our PC, this corresponds to 390 seconds. The algorithm in [11] employs a dynamic TS technique, which uses a quite different neighborhood structure and whose search space also includes unfeasible assignments as well.

Table 5 shows the computational results of these four algorithms run under the ITC–2007 competition rules. First six columns give the results of our ATS algorithm, showing the following performance indicators: the best score (f_{min}), the average score (f_{ave}) and the standard deviation (σ) over 100 independent runs, as well as the total number of iteration moves ($Iter$), the total number of perturbations ($Pert$) and the total CPU time on our computer needed for finding the best solution f_{min} (Sec). If there exist multiple hits on the best solution in the 100 independent runs, the values listed in table 5 are the average over these multiple best hits. The last three columns in table 5 indicate the best results obtained by our TS and ILS, as well as those from [11].

Table 5
Computational results and comparison under the ITC–2007 competition stop conditions

	ATS						TS	ILS	best in [11]
Instance	f_{min}	f_{ave}	σ	$Iter$	$Pert$	Sec	f_{min}	f_{min}	f_{min}
test1	**224**	229.5	1.8	15586	208	189	230	226	234
test2	**16**	17.1	1.0	35271	406	182	16	16	17
test3	**74**	82.9	4.1	20549	369	160	82	79	86
test4	**74**	89.4	6.1	37346	735	208	92	83	132
comp01	**5**	5.0	0.0	321	5	5	**5**	**5**	**5**
comp02	**34**	60.6	7.5	15647	545	370	55	48	75
comp03	**70**	86.6	6.3	8246	102	257	90	76	93
comp04	**38**	47.9	4.0	5684	68	124	45	41	45
comp05	**298**	328.5	11.7	35435	54	191	315	303	326
comp06	**47**	69.9	7.4	13457	245	116	58	54	62
comp07	**19**	28.2	5.6	15646	368	383	33	25	38
comp08	**43**	51.4	4.6	17404	190	380	49	47	50
comp09	**99**	113.2	6.9	20379	238	370	109	106	119
comp10	**16**	38.0	10.8	16026	160	389	23	23	27
comp11	**0**	0.0	0.0	236	3	3	**0**	**0**	**0**
comp12	**320**	365.0	17.5	40760	590	382	330	324	358
comp13	**65**	76.2	6.1	16779	182	300	71	68	77
comp14	**52**	62.9	6.4	24427	270	368	55	53	59

From table 5, one clearly observes that the ATS algorithm achieves always the best results (in bold), comparing with the other three algorithms. For the instances where the four algorithms reach the same results (*comp01* and *comp11*), they concern the optimal solutions and can be reached by our ATS algorithm within 5 seconds). For other instances, our ATS algorithm outperforms its two main components TS and ILS, which highlights the importance of the hybrid mechanism of adaptively integrating TS and ILS. When comparing with the reference algorithm in [11], one finds that even the results of our TS and ILS algorithms are better than that from [11].

Table 6
Computational results of ATS algorithm under the relaxed stop condition

Instance	ATS					
	f_{min}	f_{ave}	σ	*Iter*	*Pert*	*Sec*
test1	**224**	227.2	0.5	17845	234	216
test2	**16**	16.0	0	32416	351	167
test3	**73**	76.0	2.56	40849	667	2078
test4	**73**	86.4	4.23	109198	2054	1678
comp01	**5**	5.0	0.0	321	5	5
comp02	**29**	50.6	8.78	768334	1032	3845
comp03	**66**	78.6	6.07	160909	1903	2078
comp04	**35**	42.3	3.53	23113	266	566
comp05	**292**	328.5	11.7	35435	54	191
comp06	**38**	56.5	8.0	216848	1527	2973
comp07	**13**	29.7	6.48	390912	3508	4035
comp08	**39**	48.8	3.75	203982	2352	3069
comp09	**98**	109.2	5.7	70443	909	1454
comp10	**10**	28.8	9.0	33971	371	838
comp11	**0**	0.0	0.0	247	4	3
comp12	**310**	328.5	11.7	742316	10392	2513
comp13	**59**	69.9	7.4	793989	10078	4207
comp14	**52**	28.2	5.6	23754	260	378

7.3. *Results Using More Computing Budgets*

One observes that the best solutions for some instances are reached near the timeout (390 seconds) following the ITC–2007 rules. This might reveal the possibility of obtaining still better results if the rigorous stop condition required by ITC–2007 is relaxed. Therefore, in our second experiment, we aim to evaluate the search potential of our proposed ATS algorithm with a relaxed stop condition. For this purpose, we terminate our algorithm when a fixed number of iteration moves (800,000) is reached. Table 6 shows the computational results of our ATS algorithm run under this stop condition and indicates the following information: f_{min}, f_{ave}, σ, $Iter$, $Pert$ and Sec over 100 independent runs. The meaning of all these symbols are the same as in table 5.

From table 6, one finds that for most instances (except two instances of the first set and three of the second), better solutions are found under the relaxed stop condition. One observes that our ATS algorithm improves the results obtained under the competition timeout condition listed in table 5, in terms of the three criteria f_{min}, f_{ave} and σ. It should be noticed that the results in bold are the best solutions we found so far and we list these results for future comparisons.

Given the fact that neither previous best results nor good lower bounds are available for these instances (except for *comp01* and *comp11* whose lower bounds is easy to calculate and reached by our algorithm within several seconds), it is difficult to have an absolute assessment of these results for the moment. The competition results (we are one of the five finalists of ITC–2007) would give us a more reliable comparison basis, but more generally, tight lower bounds are necessary and remain to be developed.

8. Analysis and Discussion

Our second aim in this paper is 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 in the token-ring way? What is the impact of the neighborhood reduction technique on the performance of the algorithm? How about the importance of the randomized penalty-guided perturbation strategy? Whether the new proposed double Kempe chains neighborhood is a value-added one? In this section we carry out a series of experimental analysis and attempt to answer these questions.

8.1. *Neighborhood Combination*

One of the most important features of neighborhood-based meta-heuristic is surely the definition of its neighborhood. We propose in this paper two different neighborhoods: basic neighborhood N_1 and advanced neighborhood N_2. 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 [19]. For token-ring combination, we begin the search in two ways: from N_1 and N_2 respectively, denoted by $N_1 \to N_2$ and $N_2 \to N_1$.

To make the comparison as fair as possible, we employ a *steepest descent* (SD) algorithm where only better neighborhood solutions are accepted. 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.

Table 7
Average soft costs for different neighborhoods and their combinations

Instan.	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**(1.16)	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)
comp08	154(0.7)	129(3.4)	112(5.2)	**105**(2.5)	109(3.5)
comp09	231(0.5)	189(2.1)	182(2.1)	**174**(1.7)	175(2.1)
comp10	186(0.9)	147(5.3)	128(9.0)	127(3.0)	**116**(5.1)
comp11	11(0.1)	11(0.1)	6(0.2)	**4**(0.1)	5(0.2)
comp12	774(0.5)	743(0.5)	684(0.8)	667(0.6)	**654**(1.1)
comp13	186(0.8)	151(3.9)	134(7.6)	131(2.7)	**130**(3.7)
comp14	175(0.5)	156(1.3)	132(2.7)	**120**(1.6)	124(2.0)

We apply the 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 14 competition instances. The average soft cost and CPU time (seconds, in brackets) over 50 independent SD runs are given in table 7. Note that the average soft costs have been rounded up and the best average soft costs are indicated in bold for each instance. From table 7, 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.

Moreover, we have carried out the same experiments using other advanced local search methods (such as Tabu Search and Iterated Local Search) under various stop conditions. As expected, the token-ring way combination of N_1 and N_2 always produces the best solutions. Meanwhile, for the two ways of token-ring search, starting the search from the basic neighborhood N_1 costs less CPU time than from N_2 for reaching similar solution quality.

8.2. *Influence of Neighborhood Reduction*

In subsection 5.3, we presented a special reduction technique to estimate the goodness of a move of the advanced neighborhood N_2 without actually calling the matching algorithm. Here we show that the proposed neighborhood reduction technique 1) enables to reduce considerably the evaluation cost of N_2; 2) does not sacrifice the solution quality.

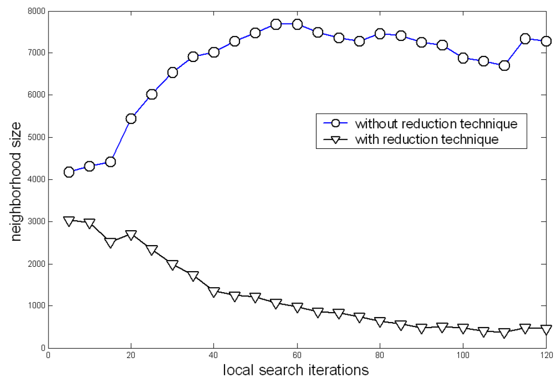

Fig. 2. Kempe chain neighborhood size with and without the reduction technique

In order to verify the first assumption, we compare the two neighborhoods with and without the reduction technique (denoted by N_2^* and N_2 respectively) in terms of their neighborhood size, which determines the computational efforts for evaluating the whole neighborhood solutions. Figure 2 shows the thoroughly evaluated neighborhood size of N_2 and N_2^* evolving with SD local search iterations for the largest instance *comp07* (very similar results are observed for other instances).

From figure 2, it is clear that with the reduction technique the neighborhood size (N_2^*) is becoming smaller and smaller along with the algorithm progressing, while the neighborhood size without reduction technique (N_2) remains the same or even becomes larger during the SD algorithm. One observes that by employing this reduction technique, at each iteration only a small subset of the neighborhood solutions are thoroughly evaluated and thus it allows the algorithm to save considerable CPU time.

On the other hand, we attempt to investigate whether the reduced neighborhood sacrifices the solution quality. For this purpose, we tested the SD algorithm on the 14 competition instances with and without the reduction technique technique. Figure 3 presents the average soft cost of N_2 and N_2^* over 50 independent runs for each instance. It is easily seen that the average soft costs with and without the reduction technique are almost the same. This means that the employment of this technique does not sacrifice the solution quality.

In order to make out why the solution quality of the modified neighborhood N_2^* is not sacrificed, we observe the distributions of the local minimum solutions of the original neighborhood N_2. Figure 4 shows the relationship of the period related sub-cost Δf_p with the total incremental cost Δf during a SD procedure (for the same instance *comp07*). Each point in the graph represents a local minimum solution (x-axis denotes its period related sub cost Δf_p and y-axis denotes its total incremental cost Δf) during the SD algorithm based on N_2. One can easily observe that almost all the local minimum solutions lies in the left side of the threshold line $\Delta f_p = 2$, i.e., we can use the threshold $\tau = 2$ to cutoff the neighborhood without missing the majority of the local minimum solutions. It is also interesting to observe that the period related sub-cost Δf_p is approximately proportional to the total incremental cost Δf. This is the basic reason why the period-related sub-cost Δf_p can be used to estimate the goodness of the total incremental cost Δf.

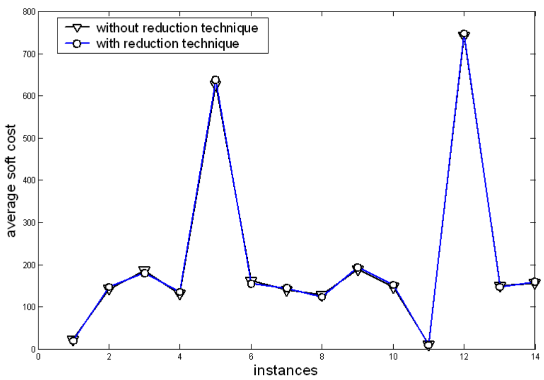

Fig. 3. Average soft cost comparison for N_2 with and without the reduction technique

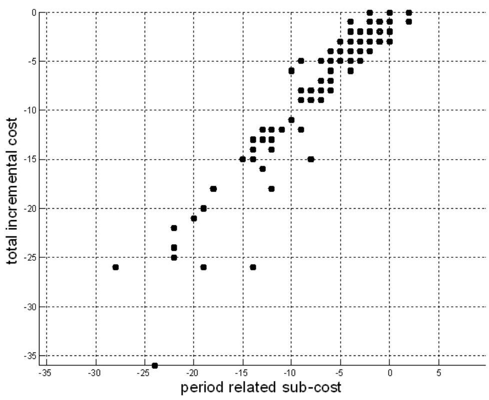

Fig. 4. Relationship between the period-related sub-cost Δf_p with the total incremental cost Δf for N_2

8.3. *Analysis of Penalty-Guided Perturbation Strategy*

In subsection 6.1, 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 6.1, 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*).

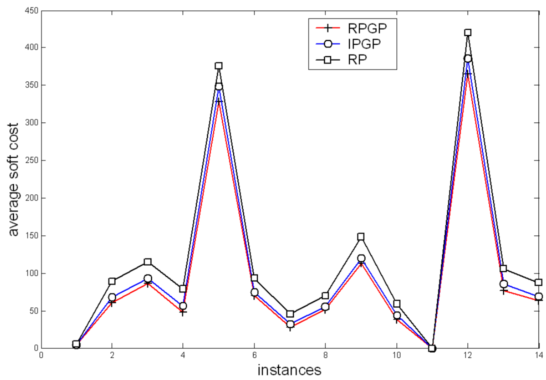

Fig. 5. Average soft costs for perturbation strategies *RPGP*, *IPGP* and *RS*

Keeping other ingredients unchanged in our ATS algorithm, we tested the above three lecture selection strategies with the 14 instances under the competition timeout stop condition. Figure 5 shows the average soft costs of these three strategies over 50 independent runs. One can easily find that the randomized and intensive penalty-guided strategies outperforms the random strategy, which highlights the importance of the penalty-guided perturbation strategy. In addition, the randomized penalty-guided strategy (*RPGP*) is also slightly better than the intensive penalty-guided strategy (*IPGP*), which implies that always restricting moved lectures to the highest penalized ones is too intensive such that the search may fall easily into a previous local optimum.

On the other hand, from the computational results of TS and ILS reported in table 5, we can clearly find that ILS with the penalty-guided strategy even outperforms TS (without perturbation) for almost all the 14 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.

8.4. *Interests of the Double Kempe Chains Move*

In subsection 5.2, we have proposed a new neighborhood move–double Kempe chains interchange, where two connected components of a subgraph concerning two periods are involved. In order to evaluate whether the newly proposed double Kempe chains move is a value-added one, our 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, each of which concerns only one kind of move.

Neighborhood $N_1^{(a)}$: $N_1^{(a)}$ is defined as all the feasible moves of *OneMove*. Each *OneMove* consists of moving one lecture from one position to another free position.

Neighborhood $N_1^{(b)}$: $N_1^{(b)}$ is defined as all the feasible moves of *TwoSwap*. Each *TwoSwap* move consists in exchanging the hosting periods and rooms assigned to two lectures of different courses. Note that *TwoSwap* move does not include any move of *OneMove*.

Neighborhood $N_2^{(a)}$: $N_2^{(a)}$ is defined as all the feasible moves of *SingleKChain*. Each *SingleKChain* move consists in exchanging the hosting periods assigned to the lectures in a single Kempe chain concerning two distinct periods, see subsection 5.2.

Table 8
Average soft costs for $N_1^{(a)}$ to $N_2^{(b)}$ over 50 independent runs

Instan.	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)
comp08	154(0.7)	205(0.3)	185(0.6)	**139**(3.2)	129(3.4)
comp09	238(0.4)	273(0.2)	244(0.4)	**193**(2.0)	189(2.1)
comp10	195(0.8)	250(0.4)	249(0.9)	**145**(5.1)	147(5.3)
comp11	16(0.1)	16(0.1)	25(0.0)	**9**(0.1)	11(0.1)
comp12	807(0.5)	874(0.3)	885(1.6)	**746**(0.5)	743(0.5)
comp13	197(0.7)	233(0.4)	224(0.7)	**151**(3.7)	151(3.9)
comp14	180(0.5)	213(0.2)	206(0.3)	**151**(1.2)	156(1.3)

Neighborhood $N_2^{(b)}$: $N_2^{(b)}$ is defined as all the feasible moves of *DoubleKChain*. Each *DoubleKChain* move consists in exchanging the hosting periods assigned to the lectures in two distinct Kempe chains concerning two distinct periods, see subsection 5.2. It should be noticed that *DoubleKChain* here does not include any move of *SingleKChain*, i.e., none of the two Kempe chains can be empty.

Note that except *DoubleKChain* move, the first three moves have been proposed in the previous literature [12]. It is easy to see that our neighborhoods N_1 and N_2 defined in subsection 5.2 are the neighborhood union of these four neighborhoods, i.e., $N_1 = N_1^{(a)} \cup N_1^{(b)}$, $N_2 = N_2^{(a)} \cup N_2^{(b)}$.

Table 8 shows the average cost functions for the SD algorithm based on $N_1^{(a)} \sim N_2^{(b)}$ over 50 independent runs. The averaged running times are given in parenthesis. From table 8, it is observed that the new proposed double Kempe chain neighborhood $N_2^{(b)}$ dominates the other three neighborhoods in terms of the solution quality, but needs more CPU time than others. However, we believe that its power to find high quality solutions deserves the additional CPU cost.

When comparing with the results of neighborhood N_2 (given in the last column and taken from table 7), one can easily find that neighborhood $N_2^{(b)}$ and N_2 obtains quite similar results in terms of both solution quality and CPU time. Note that their results are much better than that of the single Kempe chain neighborhood $N_2^{(a)}$, which emphasizes the importance of the proposed double Kempe chain move.

We have to mention that the same experiments have also been carried out on our TS, ILS and ATS algorithms. As was expected, the double Kempe chains move always obtains the best results in terms of solution quality. This further highlights the interest of the new *DoubleKChain* move.

9. Conclusions

To conclude, we have provided a mathematical formulation of the university curriculum-based course timetabling problem and presented a hybrid heuristic approach (Adaptive Tabu Search, ATS) to solving 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 proposed a new greedy heuristic for quickly producing initial feasible solutions. Second, 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. Third, 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 have assessed the performance of the proposed ATS algorithm on two sets of 18 problem instances. For these instances, we showed the advantageous merits of the proposed algorithm over TS and ILS alone, as well as another reference algorithm. We also present the best solutions found so far when the competition stop condition is relaxed. These results are reported for future comparisons. Tight lower bounds would have allowed a finer assessment, unfortunately, such bounds are not unavailable yet. Given the various constraints and the complexity of the problem, it is expected that tight lower bounds can be obtained only by advanced technique, which constitutes naturally another interesting search opportunity.

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, the effectiveness of the Kempe chain neighborhood reduction technique is carefully verified. Also, we have demonstrated that our randomized penalty-guided perturbation strategy is essential for our ATS algorithm. Finally, we carried out experiments to show that the proposed double Kempe chains move outperforms three other previous ones in the literature.

Let us comment that although the focus of this work is to propose a particular algorithm developed for solving a course timetabling problem, the basic ideas and fundamentals are quite general and would be applicable to other similar problems. At the same time, it should be clear that for a given problem, it is indispensable to realize specific adaptations by taking into account problem-specific knowledges in order to obtain an effective algorithm.

Acknowledgement

This work was partially supported by "*Angers Loire Metropole*".

References

[1] Second International Timetabling Competition (ITC–2007), Track 3: Curriculum-based Course Timetabling, http://www.cs.qub.ac.uk/itc2007/.

[2] R. Bai, E. K. Burke, G. Kendall, et al., A simulated annealing hyper-heuristic for university course timetabling, in: Proceedings of the 6th PATAT Conference, Brno, Czech Republic, 2006.

[3] U. Brannlund, P. O. Lindberg, A. Nou, J. E. Nilsson, Timetabling using lagrangian relaxation, Transportation Science 32 (1998) 358–369.

[4] E. K. Burke, B. L. MacCarthy, S. Petrovic, et al., Multiple-retrieval case-based reasoning for course timetabling problems, Journal of Operations Research Society 57(2) (2006) 148–162.

[5] E. K. Burke, B. McCollum, A. Meisels, et al., A graph-based hyper heuristic for timetabling problems, European Journal of Operational Research 176 (2007) 177–192.

[6] E. K. Burke, J. P. Newall, A multi-stage evolutionary algorithm for the timetable problem, IEEE Transactions on Evolutionary Computation 3(1) (1999) 63–74.

[7] E. K. Burke, J. P. Newall, Solving examination timetabling problems through adaptation of heuristic orderings, Annals of Operations Research 129 (2004) 107–134.

[8] E. K. Burke, S. Petrovic, Recent research directions in automated timetabling, European Journal of Operational Research 140(2) (2002) 266–280.

[9] M. W. Carter, G. Laporte, S. Y. Lee, Examination timetabling: Algorithmic strategies and applications, Journal of the Operational Research Society 47(3) (1996) 373–383.

[10] S. Casey, J. Thompson, Grasping the examination scheduling problem, in: E. K. Burke, P. D. Causmaecker (eds.), Proceedings of the 4th PATAT Conference, Springer-Verlag, 2003.

[11] F. D. Cesco, L. D. Gaspero, A. Schaerf, Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, and results, Tech. Rep. http://tabu.diegm.uniud.it/ctt/DDS2008.pdf, University of Udine (Feb 2008).

[12] M. Chiarandini, M. Birattari, K. Socha, et al., An effective hybrid algorithm for university course timetabling, Journal of Scheduling 9 (2006) 403–432.

[13] M. Chiarandini, A. Schaerf, F. Tiozzo, Solving employee timetabling problems with flexible workload using tabu search, in: Proceedings of the 3rd PATAT Conference, 2000.

[14] T. B. Cooper, J. H. Kingston, The complexity of timetable construction problems, in: E. K. Burke, P. Ross (eds.), Proceedings of the 1st PATAT Conference, Springer-Verlag, 1996.

[15] P. Côté, T. Wong, R. Sabourin, Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem, in: E. K. Burke, M. Trick (eds.), Proceedings of the 5th PATAT Conference, Lecture Notes in Computer Science 3616, Springer-Verlag, 2005.

[16] P. David, A constraint-based approach for examination timetabling using local repair techniques, in: E. K. Burke, M. W. Carter (eds.), Proceedings of the 2nd PATAT Conference, Lecture Notes in Computer Science 1408, Springer-Verlag, 1998.

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

[18] L. D. Gaspero, A. Schaerf, Multi-neighbourhood local search with application to course timetabling, in: E. K. Burke, P. D. Causmaecker (eds.), Proceedings of the 4th PATAT Conference, Springer-Verlag, 2003.

[19] L. D. Gaspero, A. Schaerf, Neighborhood portfolio approach for local search applied to timetabling problems, Journal of Mathematical Modeling and Algorithms 5(1) (2006) 65–89.

[20] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, 1997.

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

[22] H. R. Lourenco, O. Martin, T. Stützle, Iterated local search, Handbook of Meta-heuristics, Springer-Verlag, Berlin Heidelberg, 2003.

[23] B. McCollum, A perspective on bridging the gap between theory and practice in university timetabling, in: Proceedings of the 6th PATAT Conference, Brno, Czech Republic, 2006.

[24] L. T. G. Merlot, N. Boland, B. D. Hughes, et al., A hybrid algorithm for the examination timetabling problem, in: E. K. Burke, P. D. Causmaecker (eds.), Proceedings of the 4th PATAT Conference, Lecture Notes in Computer Science 2740, Springer-Verlag, 2003.

[25] A. Misevicius, A. Lenkevicius, D. Rubliauskas, Iterated tabu search: an improvement to standard tabu search, Information Technology and Control 35(3) (2006) 187–197.

[26] A. R. Mushi, Tabu search heuristic for university course timetabling problem, African Journal of Science and Technology 7(1) (2006) 34–40.

[27] C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Inc., 1982.

[28] O. Rossi-Doria, B. Paechter, C. Blum, et al., A local search for the timetabling problem, in: Proceedings of the 4th PATAT Conference, 2002.

[29] R. Santiago-Mozos, S. Salcedo-Sanz, M. DePrado-Cumplido, et al., 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.

[30] A. Schaerf, Tabu search techniques for large high-school timetabling problems, in: Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI Press/MIT Press, 1996.

[31] A. Schaerf, A survey of automated timetabling, Artificial Intelligence Review 13(2) (1998) 87–127.

[32] J. A. M. Schreuder, Constructing timetables for sport competitions, Mathematical Programming Study 13 (1980) 58–67.

[33] M. J. F. Souza, N. Maculan, L. S. Ochi, A GRASP-tabu search algorithm for solving school timetabling problems, Applied Optimization Metaheuristics: Computer Decision-Making, Kluwer Academic Publishers, Norwell, MA, USA, 2004, pp. 659–672.

[34] J. Thompson, K. Dowsland, A robust simulated annealing based examination timetabling system, Computer and Operations Research 25 (1998) 637–648.

[35] 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.

[36] 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.