Local Search for a Planning Problem with Resource Constraints in Health Simulation Center

Simon Caillard, Laure Brisoux Devendeville, Corinne Lucet

Applied Intelligence manuscript No.
(will be inserted by the editor)

Local Search for a Planning Problem with Resource Constraints in Health Simulation Center

Simon Caillard · Laure Brisoux Devendeville · Corinne Lucet

Received: date / Accepted: date

Abstract The Health Simulation Center SimUSanté performs training sessions for several different healthcare actors. This paper presents in detail the planning problem concerning time and resources encountered by SimUSanté, proposes 0-1 linear program modelization and the combination of greedy algorithm $SimUG$ with a local search algorithm $SimULS$, to solve it. New instances stemmed from the Curriculum-Based Courses Timetabling Problem (CB-CTT) are generated, in order to test $SimUG$ and $SimULS$ on representative instances. The computational results show respectively that $SimUG$ and $SimULS$ schedule on average 93.55% of the activities on the tested instances. Moreover, $SimULS$ obtains results with an average gap of 9.97% from the known optimal solution.

1 Introduction

In recent years, research into development of scheduling solutions in the healthcare sector has become increasingly important. These works discuss problems related to patient service quality, optimization of resource management and, as in our case, training time for health professionals. The training center SimUSanté located in Amiens, France, is one of the biggest multidisciplinary active pedagogy centers in Europe. This center is used by all kinds of health actors: professionals, students, patients and carers. The aim of this center is to provide a space where all of its actors can meet and train together by simulating medical acts in various fields of healthcare (such as surgical operations, blood sampling, cardiopulmonary resuscitation, etc.), but also attending regular courses. Thus, the number of different variables such as the number of activities, resources, and employees, their skills, and the number of operating rules might come to be important and become a problem when constructing a coherent and effective schedule.

The purpose of our collaboration with the SimUSanté center is to study possible solutions to management and planning problems they encounter in all their

S. Caillard · L. Brisoux Devendeville · C. Lucet
Laboratoire MIS (EA 4290), Université de Picardie Jules Verne
33 rue Saint-Leu, 80039 Amiens Cedex 1, France
E-mail: {simon.caillard, laure.devendeville, corinne.lucet}@u-picardie.fr

activities. First, we presented and formalized the SimUSanté problem and we proposed a mathematical model related to the model presented in [11]. A dedicated greedy algorithm $SimUG$ is proposed in order to provide a first suitable and compact solution in a short time. Then, a local search algorithm $SimULS$ was added to improve solutions obtained by $SimUG$. It is suited to being used over a short search time (less than 2 hours) and then uses strong diversification operators combined with an isolated tabu system when the regular diversification phase occurs. During intensification phases, it generates several neighborhood solutions from a selected operator and uses random proportionality rules to accept continuing the search from one of these neighbors. Because there is no realistic benchmark to our knowledge of the SimUSanté problems, we have generated adequate instances [13] inspired by those used in the Curriculum-Based Courses Timetabling problem [47] to train our algorithms. We compared $SimUG$ and $SimULS$ with the results worked out by our mathematical model implemented in CPLEX.

The paper is organized as follows. Section 2 presents a state-of-the-art related to issues close to the SimUSanté problem. In section 3, we describe and formalize the scheduling problem encountered by SimUSanté. The corresponding mathematical model is presented in section 4. In section 5 we present our greedy algorithm $SimUG$ and give the different selection criteria of the construction process. In section 6, we introduce our local search algorithm $SimULS$ and describe the movement operators used. Section 7 explains the instance generation and provides computational results. Section 8 concludes this paper with some final remarks.

2 State of the art

Timetabling is an important field of research which has many applications in real life problems, like hospitals, universities, airports, etc. Most of the time, scheduling is a difficult process that needs to be done in several stages and requires coordination and communication between the different groups involved: students, teachers, administrative department and heads of services for university timetables for example. In this paper, we are concerned with educational timetabling. Historically, these kinds of problems have been grouped into three categories: examination timetabling problems (ETP), school timetabling problems (STP) [42] and university course timetabling problems (UCTP) [5]. All these problems are NP-Hard [17]. They are defined as a classroom assignment [15] or a course assignment problem [14]. In recent years, a wide variety of articles related to educational scheduling problems have been published, as shown in many surveys [43, 29, 33]. The UCTP has been mostly studied [7] and considers many constraints [4] that can be classified in two sets. The constraints that must be satisfied are called hard constraints and make the schedule feasible. The soft constraints are optional and their optimization ensures timetables of high quality [3]. Although similar in many ways, UCTP can vary from one institution to another, with the addition of specific constraints. With the growing interest in this area, two international competitions have been organized: International Timetabling Competition (ITC-2002 [40] and ITC-2007 [34]). The objective of ITC-2002 was to formalize problem definition for the university course timetabling problem, and to generate some instances related to it. In ITC-2007, university course timetabling problems have been split into two subproblems [36]: Post-Enrolment-based Course Timetabling (PE-CTT)

and Curriculum-Based Course Timetabling (CB-CTT) and one of the aims of the competition was to reduce the differences between practice and research in educational scheduling problems [35]. The SimUSanté problem is an extension of CB-CTT [18]. A curriculum is the set of lectures that students follow and a course is composed by one or many lectures. The problem consists of finding the best weekly assignment for lectures, available rooms, time periods, etc. for a set of curricula while respecting a set of constraints. Even though the hard constraints are roughly the same, the case of SimUSanté differs in some points from the CB-CTT: in SimUSanté, there are no periodic schedules and we do not consider many soft constraints. However we need to consider lectures with room, time and precedence constraints. We also need to take into account different types of resources (rooms have specific characteristics like surgical block, pharmacy, etc.), different types of skills needed to perform lectures, resources with multi-types, scheduling of lunch breaks, etc.

Due to its importance, many approaches have been studied to solve UCTP and its subproblems. Most of them are optimization methods and give optimal or near optimal results. The ITC-2007 instances are often used as UTCP benchmarks to compare proposed methods. These approaches are split into two categories: exact method and metaheuristics. Only a few exact methods exist in the literature to solve UTCP. The Integer Linear Programming (ILP) approach is an exact method which uses a mathematical model to solve an objective function [41, 44, 37]. The ILP given in [12] is adapted to the CB-CTT and can be used with a generic solver. In [28], a two-step ILP is used to solve the CB-CTT. The first step assigns lectures to time periods, without taking into consideration rooms, minimizing penalties for curriculum compactness and working days. The second one assigns lectures to rooms. A generalized ILP method for the CB-CTT, which is able to handle most of the ITC-2007 instances is presented in [25]. The graph coloring approach has also been used. In [6], they use an edge coloring graph algorithm to first distribute resources to classes and then set time slots for each of them.

Metaheuristics used to solve UTCP can be divided into two groups. The first one concerns local search-based methods [2, 19]. They are characterized by the definition of several different neighborhood operators allowing switching from one solution to another (changing the time period assigned to a lecture, or the room assignment for a lecture, etc.). They are often composed of several phases. The first one builds an initial solution and the others explore the neighborhoods to improve it, as shown in [39]. They use different mechanisms to avoid being trapped in a local minimum. In these kinds of methods, we found simulated annealing (SA) [8, 20], tabu search (TS) [16, 23, 38], great deluge (GD) [26], hill climbing (HC) [48], etc. In [2], authors proposed a local search algorithm that starts from an initial solution and generates a set of neighborhood solutions with the aid of eight classic neighborhood operators stemmed from the literature [1, 46]. Each of these solutions is compared to the current best solution in order to determine the more promising search space. From this, the best solution is selected and the process is iterated until a new best solution is found or if there is no further improvement. In [19], a multi-neighbourhood local search is used. The method starts with the first neighbourhood operator and when it has found its best local optimum, starts the search again with the second neighbourhood operator from this new best local optimum. This procedure is then repeated over many iterations. With limited running time, it seems to be better to use local search with robust diversification

operators [27] like those proposed in [30]. In [38], a tabu search hybridized with a variable neighborhood search is used. It starts from an initial solution obtained by a greedy algorithm and generates several neighborhood solutions using shaking operators. A solution is accepted or set in tabu list. A solution is accepted if it is better than the current best one or if it satisfied an acceptance criteria. Then the search continues from here, generating new neighborhood solutions.

The second group is that of population-based methods, which are often bio-inspired approaches. They consider many solutions at the same time which are combined to obtain better ones. In these kinds of methods, we found: Ants Colony Optimization (ACO) [32], Genetic Agorithms (GA) [21], Particle Swarm Optimization (PSO) [22], etc. Local search based and population based algorithms have two disadvantages [9,45]: early convergence and being stuck in a local optimum, so hybrid approaches which combine the two procedures have often been used [10,24,31].

We also propose a local search algorithm to solve the SimUSanté problem. Indeed, this kind of algorithm is adapted to real problems and gives good results in a short time. Nevertheless the quality of the solutions mainly relies on the quality of the neighborhood operators. In this sense, we have developped several operators dedicated to the SimUSanté problem, whose principles nevertheless remain applicable to similar planning problems.

3 SimUSanté: a planning problem with resource constraints

In such a large center of multidisciplinary active pedagogy as SimUSanté, there are a lot of activities, grouped into training sessions, with some precedence constraints, specific skill requirements and specific room equipment. Moreover, because SimUSanté is a simulation center, the room facilities are both varied and flexible. Planning all activities while respecting the various constraints is an issue of primary importance for the proper functioning and success of such a center. It is important to schedule as many activities as possible in sessions and that session timetables are compact. This minimizes the number of days for learners and teachers. First let's formally define the data problem:

Horizon: Horizon H corresponds to one week decomposed into working days. Let D be the set of these working days. So, $\forall d \in D$, we denote T_d the set of time slots of day d and $T = \bigcup_{d \in D} T_d$. $start_d = \min_{t \in T_d}\{t\}$ represents the first time slot of day d and $end_d = \max_{t \in T_d}\{t\}$, the last one. $\bar{d}$ is the last day of H. Each time slot represents one hour and a day is composed of 9 time slots. Let $break_d$ be a subset of slots identified as potential time breaks, for day d. One at least of these time slots should be idle to ensure the existence of a daily lunch break for any sessions and employees.

Resources: We have a finite set of resources $R = R^r \cup R^e$ with R^r the set of rooms and R^e the set of employees. To R^e is associated a set of types $\Lambda^e = \{\lambda_1, ..., \lambda_{|\Lambda^e|}\}$ which corresponds to the skills of employees. To R^r is also associated a set of types $\Lambda^r = \{\lambda_{|\Lambda^e|+1}, ..., \lambda_{|\Lambda^e|+|\Lambda^r|}\}$ which corresponds to specific room equipment. We denote $\Lambda = \Lambda^r \cup \Lambda^e$. Each resource can have more than one associated type. For example, a room may be equipped with artificial arms for the simulation of taking blood, but also with artificial vertebral columns for the simulation of lumbar punctures. We denote R^{λ_i} = the

set of resources of type λ_i, and $qtav^t_{\lambda_i}$ the quantity of resource λ_i available at time slot t. All activities scheduled at time slot t cannot use more than the available resources. Each employee $e \in R^e$ is associated with a set of types Λ_e and their availability at each time slot is taken into account by $isavailable^t_e$ which is equal to 1 if employee e is available at time slot t and 0 otherwise.

Activities: Let A be the set of activities and s_a the session to which a belongs. Each activity $a \in A$ is characterized by a duration $duration_a$, an earliest starting date ES_a and latest starting date LS_a. $qtreq^a_{\lambda_i}$ is the quantity of resources of type λ_i, $\forall i = 1, .., |\Lambda|$ required by activity a, and $\Lambda_a = \{\lambda_i \in \Lambda \ / \ qtreq^a_{\lambda_i} \neq 0\}$, the set of resource types required by a. $iseligible^t_a$ is equal to 1 if there are enough resources for scheduling activity a at time slot t (i.e. $\forall \lambda_i \in \Lambda_a$, $qtreq^a_{\lambda_i} \leq qtav^t_{\lambda_i}$), 0 otherwise. A precedence relation is defined between the activities and we denote $pred_a$ the set of activities that must be planned before activity a.

Training session: Let S be the set of training sessions to schedule over horizon H. Each training session $s \in S$ is composed of a set of activities A_s and has a total duration $duration_s = \sum_{a \in A_s} duration_a$. We denote $\Lambda_s = \bigcup_{a \in A_s} \Lambda_a$, the set of resource types required per training session s.

4 Mathematical model

The model proposed for the SimUSanté problem is based on [11], that considers CB-CTT as an extension of the Resources Constrained Project Scheduling Problem (RCPSP). The two $0-1$ decision variables used in our model are:

- $x^t_a = 1$ if activity a starts at time slot t, 0 otherwise, $\forall a \in A$ and $\forall t \in T$.
- $y^{e,\lambda}_a = 1$ if activity a is assigned to employee e with type λ, 0 otherwise, $\forall a \in A$, $\forall e \in E$ and $\forall \lambda \in \Lambda_e$.

The objective is to plan as many activities as possible while minimizing the completion time (makespan) of each session. This objective is represented by function in equation (1) where the first term tends to minimize the session makespans and the second one tends to minimize the number of unscheduled activities. Coefficient α is penalty applied to unscheduled activities. $starts_s$ and end_s are two fictitious activities with duration zero, that respectively represent the beginning and the end of session s.

$$minimize \sum_{s \in S} \sum_{t \in T} (x^t_{end_s} \times t - x^t_{start_s} \times t) + \sum_{a \in A} (1 - \sum_{t \in T} x^t_a) \times \alpha \qquad (1)$$

This objective function (see equation (1)) must be minimized while respecting all time and resource constraints that are represented by the following linear constraints:

$$\sum_{t \in [ES_a, LS_a]} x_a^t \leq 1 \qquad \forall a \in A \tag{2}$$

$$\sum_{t \in T \setminus [ES_a, LS_a]} x_a^t = 0 \qquad \forall a \in A \tag{3}$$

$$\sum_{t \in T} x_a^t \leq 1 \qquad \forall s \in S,\ \forall a \in \{start_s,\ end_s\} \tag{4}$$

Equations (2) and (3) ensure that each activity a is scheduled once at most, and in $[ES_a, LS_a]$. Equation (4) checks that fictitious activities are scheduled once at most.

$$\sum_{t \in T} t.(x_a^t - x_b^t) \geq duration_b, \qquad \forall s \in S,\ \ \forall a \in A_s \cup \{end_s\},\ \ \forall b \in pred_a \tag{5}$$

$$\sum_{e \in R^\lambda} y_a^{e,\lambda} = qtreq_\lambda^a, \qquad \forall a \in A,\ \ \forall \lambda \in \Lambda^{\mathcal{E}} \tag{6}$$

$$\sum_{t'=end_d - duration_a + 1}^{t} x_a^{t'} + \sum_{t''=start_{d+1} - duration_a + 1}^{t} x_a^{t''} \leq 1,\ \ \forall a \in A,\ \ \forall d \in D \setminus \{\bar{d}\} \tag{7}$$

$$\sum_{a \in A} (qtreq_\lambda^a \times \sum_{t'=t - duration_a + 1}^{t} x_a^{t'}) \leq qtav_\lambda^t,\ \ \forall t \in T,\ \ \forall \lambda \in \Lambda^{\mathcal{R}} \tag{8}$$

Equations (5), (6), (7) and (8) correspond to constraints related to activities. Equation (5) ensures that activity a cannot start before all its predecessors. Equation (6) ensures, for each activity that the required quantity of employees with type λ is assigned to it. Equation (7) ensures that each activity is not split and scheduled on two distinct days. Finally, equation (8) verifies that the number of scheduled activities on time slot t, requiring rooms of type λ does not exceed the number of available rooms of this type.

$$\sum_{t'=t - duration_a + 1}^{t} x_a^{t'} + \sum_{b \in A_s \setminus \{a\}} \sum_{t'=t - duration_b + 1}^{t} x_b^{t'} \leq 1,\ \ \forall t \in T,\ \forall s \in S,\ \forall a \in A_s \tag{9}$$

$$\sum_{a \in A_s} \sum_{t \in break_d} \sum_{t'=t - duration_a + 1}^{t} x_a^{t'} \leq 1\ \ \forall s \in S,\ \ \forall d \in D \tag{10}$$

Equations (9) and (10) are related to session constraints. Equation (9) verifies that all activities in the session are serial scheduled and equation (10) ensures that each session has one lunch break per day if necessary.

$$duration_a . \sum_{\lambda \in \Lambda_e} y_a^{e,\lambda} - x_a^t . \sum_{q=t}^{t+duration_a-1} isavailable_e^q \leq (1 - x_a^t) \times \max_{a \in A}\{duration_a\}$$
$$\forall e \in E, \quad \forall a \in A, \quad \forall t \in T \tag{11}$$

$$\sum_{\lambda \in \Lambda_e} y_a^{e,\lambda} \leq 1 \quad \forall e \in E, \ \forall a \in A \tag{12}$$

$$\sum_{t'=t-duration_a+1}^{t} x_a^t + \sum_{t'=t-duration_b+1}^{t} x_b^t + (\sum_{\lambda \in \Lambda_e} y_a^{e,\lambda} + y_b^{e,\lambda}) \leq 3$$
$$\forall e \in E, \quad \forall t \in T, \ \forall a \in A, \ \forall b \in A \setminus \{a\} \tag{13}$$

$$\sum_{t'=t-duration_a+1}^{t} x_a^{t'} + \sum_{t'=t-duration_b+1}^{t} x_b^{t'} + \sum_{\lambda \in \Lambda_e} y_a^{e,\lambda} + \sum_{\lambda \in \Lambda_e} y_b^{e,\lambda} \leq 3$$
$$\forall e \in E, \quad \forall d \in D, \quad \forall t \in break_d, \quad a \in A, \quad b \in A \setminus \{a\} \tag{14}$$

All equations (11), (12), (13) and (14) concern the employee constraints. Equation (11) ensures that there is no activity assigned to employee e when they are not available. Equation (12) verifies that employees are assigned to one type at most per activity. Equation (13) ensures that employee e is assigned at most to one activity per time slot t. Equation (14) guarantees that each employee has a lunch break each day.

CPLEX solver (version 12.6) was used to solve this 0-1 linear program. It provides optimal solutions for small instances, but cannot process larger ones in a limited run time of 24 hours. However, thanks to the optimal solutions provided, we were able to measure the efficiency of our heuristic algorithms, *SimUG* and *SimULS* added in sections 5 and 6.

5 *SimUG*: a greedy algorithm for the SimUSanté problem

Because the real life instances are most often large and consequently unsolvable by exact methods, we propose a constructive greedy algorithm named *SimUG*. The strengths of a greedy algorithm are the simplicity and speed of execution. At each iteration of the construction process, it is necessary to make choices according to local optimality criteria, without guarantee of the global optimality. These selection criteria must therefore be relevant to the SimUSanté problem. In order to design them, we mainly focused on making a schedule compact. The training offered by the SimUSanté center must satisfy their customers, while monopolizing the learners as few as possible. We explain in this section how these criteria were constructed for *SimUG*.

Algorithm 1 *SimUG*

Require: S (set of unscheduled training sessions), T (set of time slots)
Ensure: Sol (a feasible solution), UA (set of unscheduled activities)
1: $Sol \leftarrow \emptyset$
2: $UA \leftarrow \emptyset$
3: **while** $S \neq \emptyset$ **do**
4: $s^* \leftarrow sessionChoice(S)$
5: $S \leftarrow S \setminus \{s^*\}$
6: $t^* \leftarrow betterStart(s^*, T)$
7: $t \leftarrow t^*$
8: $UA_{s^*} \leftarrow A_{s^*}$
9: **while** $(t \leq |T|)$ **and** $(UA_{s^*} \neq \emptyset)$ **do**
10: $EA_{s^*} \leftarrow eligibleActivities(UA_{s^*}, t)$
11: **if** $EA_{s^*} \neq \emptyset$ **then**
12: $(a^*, R_{a^*}) \leftarrow activityChoice(EA_{s^*}, t)$
13: $Sol \leftarrow Sol \cup (a^*, t, R_{a^*})$
14: $updateAvalaibility(a^*, t, R_{a^*})$
15: $UA_{s^*} \leftarrow UA_{s^*} \setminus \{a^*\}$
16: $t \leftarrow t + duration_{a^*}$
17: **else**
18: $t \leftarrow t + 1$
19: **end if**
20: **end while**
21: $UA \leftarrow UA \cup UA_{s^*}$
22: **end while**
23: **return** $(Sol,\ UA)$

5.1 Solution formulation

Constructing a solution is to assign a start date t_a (a time slot) to activity a, and a set of resources R_a, such that all the constraints of resources (number and type) of precedence and of operating rules are respected. A solution is represented by a set Sol of triplets (a, t_a, R_a), with $a \in A$, $ES_a \leq t_a \leq LS_a$ and $R_a \subseteq R^r \cup R^e$. R_a is a set of available resources assigned to a, which exactly matches the resources required to execute a. Let $SA \subseteq A$, $SA = \{a | (a, t_a, R_a) \in Sol\}$ be the set of scheduled activities and $UA = A \setminus SA$, the set of unscheduled ones. For session $s \in S$, $Sol_s \subseteq Sol$, represents the set of triplets of the solution related to s, with $Sol_s = \{(a, t_a, R_a) \in Sol | a \in A_s\}$. $SA_s = SA \cap A_s$, is the set of scheduled activities of s, and $UA_s = A_s \setminus SA_s$, the set of unscheduled activities of s.

For a given session $s \in S$, if at least one activity has been scheduled ($SA_s \neq \emptyset$), then the start date $t_{start_s} = \min_{(a,t_a,R_a) \in Sol_s}\{t_a\}$ and the end date $t_{end_s} = \max_{(a,t_a,R_a) \in Sol_s}\{t_a + duration_a\}$, otherwise $t_{start_s} = t_{end_s} = 0$. The makespan of session s is denoted $mk_s = t_{end_s} - t_{start_s}$.

The evaluation of Sol, denoted $Makespan(Sol)$, corresponds to the objective function presented by equation (1). It is the sum of the makespans of all sessions, plus the amount of unplanned activities multiplied by penalty α (see equation 15). The objective is to find a valid solution with a minimum $Makespan$.

$$Makespan(Sol) = \sum_{s \in S} mk_s + |UA| \times \alpha \tag{15}$$

5.2 Principle of SimUG

SimUG, described in algorithm 1, consists of scheduling one by one the sessions in S, using selection criteria that tend to minimize the objective function presented in equation (1). These selected criteria aim to compact the schedule obtained rather than planning as many activities as possible.

Two criteria are used to select the session s^* to plan - $sessionChoice()$ - and its better starting date t^* - $betterStart()$ -. Once s^* and t^* are fixed, the criterion $activityChoice()$ allows the sequential scheduling of activities of s^* from time slot t^*. In *SimUG*, $EligibleActivities()$ simply returns all unplanned activities of s^*, whose resource demands can all be satisfied at a given time slot t.

5.3 Criterion $sessionChoice(S)$

The aim of $sessionChoice()$ is to choose among unscheduled sessions in S the next to schedule. It selects the training session with the lowest difficulty score dif^s computed by equation (16). This score represents the difficulty of planning a session. dif^s estimates the total number of required resources over $duration_s$. The lower the dif^s of session s is, the more compact the schedule of s will be since s doesn't need many resources or not for a long time. In case of equal dif^s, a random choice is made.

$$\begin{aligned} dif^s &= \sum_{a \in A_s} \sum_{\lambda \in \Lambda_a} duration_a \times qtreq^a_\lambda \\ s^* &= \underset{s \in S}{\operatorname{argmin}} \ \{dif^s\} \end{aligned} \tag{16}$$

5.4 Criterion $betterStart(s^*)$

$betterStart(s^*)$ selects the start date t^* for previously chosen session s^*. This selection is made according to equation (17) which is based on three indicators that characterize each time slot $t \in H$: the resource deficiencies $D^t_{s^*}$, the resource availability $avail^t_{s^*}$ and the underestimated makespan $underspan^t_{s^*}$ of previously chosen session s^*. The chosen time slot is then the one with the lowest resource deficiencies and in case of equality between two time slots, those that have the higher resources availability and finally those that have the lowest underestimation of makespan. The computation of these indicators is described above.

$$t^* = [\ \underset{t \in T}{\operatorname{argmin}} \ \{\mathcal{D}^t_{s^*}\};\ \underset{t \in T}{\operatorname{argmax}} \ \{avail^t_{s^*}\};\ \underset{t \in T}{\operatorname{argmin}} \ \{underspan^t_{s^*}\}\ ] \tag{17}$$

To calculate these three indicators, we define for each time slot t, the earliest end of session s^* denoted $end^t_{s^*}$ described in equation (18). $end^t_{s^*}$ takes into account the total duration of the activities of s^* and the mandatory lunch breaks computed by the function $breaks$.

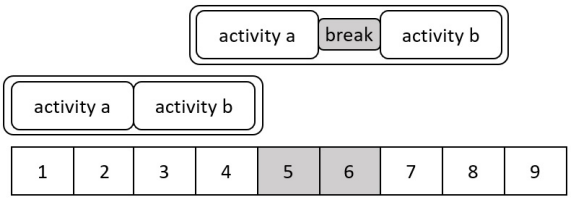

Fig. 1 Computation of the earliest end of session s^* for each time slot.

$$end^t_{s^*} = t + \sum_{a \in UA_{s^*}} duration_a + breaks(t, \sum_{a \in UA_{s^*}} duration_a) - 1 \qquad (18)$$

Figure 1 considers the set of activities $A_{s^*} = \{a, b\}$, with respective duration $duration_a = 2$ and $duration_b = 2$, and shows that if s^* starts at time slot 1 or at time slot 4, the earliest end is respectively $end^1_{s^*} = 4$ and $end^4_{s^*} = 8$, because a lunch break is required for the latter.

Lowerbound of chosen session $underspan^t_{s^*}$ is an underestimation of the makespan of s^* if it starts at t and is computed by equation (19).

$$underspan^t_{s^*} = end^t_{s^*} - t + 1 \qquad (19)$$

The previous example in figure 1 shows that the earliest ends for time slots 1 and 4 are respectively $end^1_{s^*} = 4$ and $end^4_{s^*} = 8$. The underestimation of corresponding makespans to these time slots is then $underspan^1_{s^*} = 4-1+1 = 4$ and $underspan^4_{s^*} = 8 - 4 + 1 = 5$.

Resource deficiencies $\mathcal{D}^t_{s^*}$ scores any time slot t by overestimating resource deficiencies over $[t, end^t_{s^*}]$ if s^* starts at t. We then consider that session s^* requires during its progress the maximum resource quantity over A_{s^*}, for all resource types $\lambda_i \in \Lambda_{s^*}$.

$$\begin{aligned} \mathcal{D}^t_{s^*} &= \sum_{t'=t}^{end^{s^*}_t} \sum_{\lambda_i \in \Lambda_s} \max(overeq^{s^*}_{\lambda_i} - qtav^{t'}_{\lambda_i}, 0) \\ overeq^{s^*}_{\lambda_i} &= \max_{\substack{a \in A_{s^*} \\ \lambda_i \in \Lambda_{s^*}}} \{qtreq^a_{\lambda_i}\} \end{aligned} \qquad (20)$$

So the overestimation of resource deficiencies for each time slot t is the sum of the differences between the over requirement of s^*, $overeq^{s^*}_{\lambda_i}$ and the available resource at time slot t, $qtav^t_{\lambda_i}$, for each resource type λ_i, as shown in equation (20).

Figure 2 illustrates the over requirement and considers the following requirements for the previous set of activities A_{s^*} :

activity a needs two rooms of type λ_1 ($qtreq^a_{\lambda_1} = 2$) and one employee of type λ_2 ($qtreq^a_{\lambda_2} = 1$).

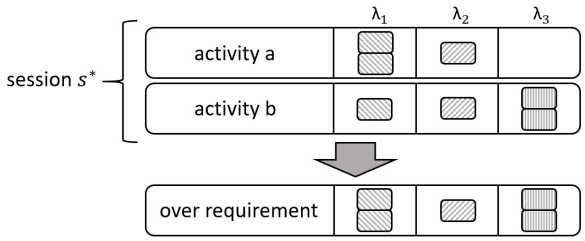

Fig. 2 Description of session s^* and its over requirement

activity b needs one room of type λ_1 ($qtreq^b_{\lambda_1} = 1$), one employee of type λ_2 ($qtreq^b_{\lambda_2} = 1$) and two employees of type λ_3 ($qtreq^b_{\lambda_3} = 2$).

The over requirements for session s^* is then $overeq^{s^*}_{\lambda_1} = \max(2,1) = 2$; $overeq^{s^*}_{\lambda_2} = \max(1,1) = 1$; $overeq^{s^*}_{\lambda_3} = \max(0,2) = 2$.

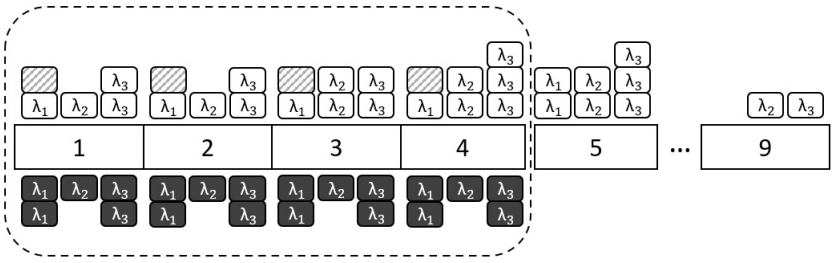

Fig. 3 Resource availabilities for time slots 1 to 9

In figure 3, the white blocks represent the quantity of resources available for time slot 1 to 9 and for types λ_1, λ_2 and λ_3. The black ones show the over requirements of session s^* presented in figure 2. The dashed rectangle represents the time slots considered to compute deficiencies for time slot 1 and the hatched blocks are the missing resources that occur in comparison with the over requirements.
As mentioned in previous examples, the earliest ending time slot, if session s^* starts at time slot 1, is $end^1_{s^*} = 4$. To compute the deficiencies for time slot 1, we then consider time slots in interval $[1;4]$ and the details of the computation is as follows : $\mathcal{D}^1_{s^*} = (\max((2-1),0)+\max((1-1),0)+\max((2-2),0)) + (\max((2-1),0)+\max((1-1),0)+\max((2-2),0)) + (\max((2-1),0)+\max((1-2),0)+\max((2-2),0)) + (\max((2-1),0)+\max((1-2),0)+\max((2-3),0)) = 4$.

Resource availability $avail^t_{s^*}$ is the average of available resources over $[t, end_t]$. The more resources left, the more opportunities to schedule remaining activities left. The available resources score of time slot t is given by equation (21).

$$avail^t_{s^*} = \frac{\sum_{t'=t}^{end^t_{s^*}} \sum_{\lambda_i \in \Lambda_s} (qtav^{t'}_{\lambda_i})}{underspan^t_{s^*}} \tag{21}$$

With our previous example, using s^* presented in figure 2 and the resource availabilities shown in figure 3, for time slot 1, the resource availability is the average of the white blocks inside the dashed rectangle, and then $avail^1_{s^*} = ((1+1+2)+(1+1+2)+(1+2+2)+(1+2+3))/4 = 19/4$.

5.5 Criterion $activityChoice(EA_{s^*}, t)$

Among the set of eligible activities criterion $activityChoice()$ chooses which activity $a^* \in EA_{s^*}$, with its pre-assigned resources R_{a^*}, will be scheduled. Its start date t_{a^*} is set to t and resource availabilities are updated by function $updateAvailability(a^*, t, R_{a^*})$.

a^* is activity a such that $(a, R_a) \in EA_{s^*}$ with the lowest $score_a$ (see equation (22)). $score_a$ overestimates the number of times a could be scheduled in $[t, end^t_{s^*}]$, thanks to $iseligible^t_a$ which is equal to 1 when a can start at t. The lower the score, the fewer possibilities there are to program a at a time slot greater than t. Note that if $score_a = 0$, there is no possibility of planning a after t.

Let $\Delta = \min_{a|(a,R_a)\in EA_{S^*}}\{duration_a\}$ be the smallest duration of the activities in EA_{s^*}. $activityChoice()$ selects a^* according to equation (22). If several activities have the same $score_a$, one is randomly chosen.

$$score_a = \sum_{d\in D} \sum_{t'\in T_d \cap [t+\Delta; end^t_{s^*}]} \left\lceil \frac{\sum_{t''=t'}^{t'+duration_a-1} iseligible^{t''}_a}{duration_a} \right\rceil \tag{22}$$

$$a^* = \underset{a|(a,R_a)\in EA_{s^*}}{\operatorname{argmin}} \{score_a\}$$

5.6 Function $updateAvailability(a^*, t, R_{a^*})$

When activity a^* is scheduled on t, all resources in R_{a^*} are set unavailable over the period $[t, t + duration_{a^*} - 1]$. Let us note that precedence constraints are also updated for all activities linked to a^*.

6 *SimULS*: a local search algorithm

As mentioned in 5, the strengths of a greedy algorithm like $SimUG$ are the simplicity of implementation and the very short run time. They also enable valid solutions to be built while not violating any constraints of the problem. However, despite the relevance of criteria exposed in section 5, the resulting solutions remain with some

unscheduled activities and/or with some idle time slots that damage the session makespans. To improve the $SimUG$ solutions, we have developed a local search algorithm $SimULS$ that explores the solution space by applying neighborhood operators, starting from a solution provided by greedy algorithm $SimUG$.

From a given valid solution Sol, $SimULS$ operator generates a set of possible movements. Each of them corresponds to a neighbor solution of Sol. The process consists of choosing one of these neighbor solutions as the current solution and continues, with the expectation of meeting a near optimal solution. The effectiveness of this method depends on the relevance of the operators and the strategy of their application. Movements and operators are respectively detailed further in sections 6.2 and 6.3.

6.1 Principle of $SimULS$

$SimULS$ proposes the following operators : $saturator$, $intra$, $extra$, $extra^+$ and $diversificator$. $saturator$ tries to complete the current solution by inserting unplanned activities into consecutive time slots with sufficient resources available. $intra$, $extra$ and $extra^+$ are applied when no movement stemmed from $saturator$ is possible. They try to insert unplanned activities by removing one or more scheduled activities from the current solution. $diversificator$ is applied as a diversification way in order to escape a local minimum.

$SimULS$ principle, described in algorithm 2, is the following: until a maximum preset $limitCounter$ iterations is reached, $SimuLS$ relies on $saturator$ to plan unscheduled activities. If $saturator$ fails to plan all the unscheduled activities, $SimULS$ calls function $selectOperator$ to choose one of the following operators: $intra$, $extra$ or $extra^+$, in order to plan a randomly selected unscheduled activity. Each of these operators removes triplets from the current solution according to some criteria and thus frees resources and unsets time slots associated with these triplets. Finally, if the best solution ever met is not improved after a preset $noImprov$ iterations, a dynamic $\beta\%$ of the solution is randomly destroyed by the $destructor$ operator.

6.2 Movement description

As mentioned in section 5.1, a solution is a set of triplets, so a movement m will be characterized by a couple $m =< (a, t_a, R_a) \ ; \ \Upsilon >$. (a, t_a, R_a) represents a triplet that will be added to the current solution, with $a \in UA$, an unscheduled activity, $t_a \in T$, a time slot from which a could be started, and R_a, the set of resources assigned to a that exactly matches its resource requirement. In order to plan a, we need to remove a set Υ of triplets from the solution. $\Upsilon = \{(b_1, t_{b_1}, R_{b_1}), \ldots, (b_n, t_{b_n}, R_{b_n})\}$, $n \in \{1, \ldots |Sol|\}$. The set of resources R_a can be composed of resources directly available over H, plus those released by cancelling all activities of Υ. A $movement$ respects all operational rules and resource constraints. Note that Υ can be an empty set if triplet (a, t_a, R_a) can be added to the solution without removing any activities. Movements and their scores are described in the following paragraph.

Algorithm 2 : SimuLS

Require: Sol (the current solution), S (set of sessions), $\forall s \in S, UA_s$ (set of unscheduled activities for session s), $UA = \bigcup_{s \in S} UA_s$ (the set of unscheduled activities), $noImprov$, $limitCounter$, β_{min}, $lim_{\beta_{max}}$
Ensure: $bestSol$ (the best solution met during the process)

```
noBest ← 0
counter ← 0
bestSol ← Sol
β ← β_min
β_max ← β_min
for all a ∈ UA do
    m ← saturator(a)
    if m ≠ m_NULL then
        (Sol, UA) ← applyMove(m, Sol, UA)
    end if
end for
bestSol ← Sol
while counter < limitCounter do
    if (noBest = noImprov) or (UA = ∅) then
        β ← randomInterval(β_min, β_max)
        (Sol, UA) ← diversificator(Sol, UA, β)
        noBest ← 0
        if β_max < lim_{β_max} then
            β_max ← β_max + 1
        end if
    end if
    if UA ≠ ∅ then
        a ← random(UA)
        m ← selectOperator({intra, extra, extra+}, a)
        if m ≠ m_NULL then
            (Sol, UA) ← applyMove(m, Sol, UA)
        end if
    end if
    for all a ∈ UA do
        m ← saturator(a)
        if m ≠ m_NULL then
            (Sol, UA) ← applyMove(mvt, Sol, UA)
        end if
    end for
    if Makespan(Sol) < Makespan(bestSol) then
        noBest ← 0
        bestSol ← Sol
        β_max ← β_min
    else
        noBest ← noBest + 1
    end if
    counter ← counter + 1
end while
return bestSol
```

Each movement m has a score, $score_m$, computed by equation 23. This score allows operators to determine which movement to apply, as show in equation 27. The higher the score associated with m, the greater the probability that m is applied. This score measures the potential improvement of the solution if this move is applied. It is composed of 3 indicators: the makespan variation Δ_m, the number of idle time slots $idle_m$ generated and the number of removed activities

$|\Upsilon_m|$. The last indicator is considered in order to limit the number of activities to plan next.

$$score_m = \frac{1}{(1+\Delta_m) \times (1+idle_m) \times (1+|\Upsilon_m|)} \tag{23}$$

In details:

Δ_m computes (see equation (24)) the difference between the new makespan Nmk_m of session s_a, if a is scheduled, and the estimated minimum makespan lb_m of session s_a.

$$\Delta_m = \begin{cases} 0 & \text{if } Nmk_m \leq lb_m \\ Nmk_m - lb_m & \text{otherwise} \end{cases} \tag{24}$$

This estimated makespan is obtained by relaxing all resource constraints and considering lunch-breaks computed by function $breaks()$. Nmk_m and lb_m are detailed in equations 25.

$$\begin{aligned} Nmk_m &= \max_{t' \in \{t_a + duration_a, t_{end_{s_a}}\}} \{t'\} - \min_{t'' \in \{t_a, t_{start_{s_a}}\}} \{t''\} \\ lb_m &= duration_{s_a} + breaks(\min_{t' \in \{t_a, t_{start_{s_a}}\}} \{t'\}, duration_{s_a}) \end{aligned} \tag{25}$$

$idle_m$ counts the number of free time slots unusable if movement m is applied. A set of contiguous free time slots is considered unusable if it is not possible to plan any unscheduled activities of s_a over it, because the duration of each of them exceeds the quantity of free slots in the set. $idle_m$ is computed at equation 26.

$$idle(m) = \sum_{t \in T_d} 1 - \left\lfloor \frac{|[t; t + duration_{min}[\cap used_d|}{duration_{min}} \right\rfloor \tag{26}$$

With d, the day to which t_a belongs, $duration_{min} = \min_{a' \in UA_{s_a}} \{duration_{a'}\}$, the smallest duration of the set of unscheduled activities of $s(a)$ and $used_d = T_d \cap \bigcup_{a' \in SA_{s_a}} [t_{a'}; t_{a'} + duration_a[$, the set of previously assigned time slots for day d and session s_a.

Υ_m is the set of activities that m envisages to remove from the solution, freeing up resources to plan a. The larger $|\Upsilon|_m$ is, the more the unplanned activities could weigh negatively on the objective function $Makespan(Sol)$.

Each of our operators, described in the next section, builds a set of movements according to different criteria, and selects one of them (see equation 27) to be applied to the current solution Sol.

6.3 Description of scheduling operators

As mentioned previously, operators generate a set of movements M. Each movement $m \in M$, with $m =< (a, t_a, R_a); \Upsilon >$, is composed of (a, t_a, R_a) the activity to plan at t_a with resource consumption R_a, and Υ the set of activities to remove from the current solution.The main difference between the four scheduling operators *saturator*, *intra*, *extra*, and $extra^+$ is the composition of Υ. Note that if there is no possible movement for an operator ($M = \emptyset$), it returns the empty movement m_{NULL}. This paragraph formally details these four operators.

$saturator_a$: This operator tends to place activity $a \in UA$ without changing the current solution. It builds a set of movements M=$\{< (a, t_a^1, R_a^1); \emptyset >, \ldots, < (a, t_a^k, R_a^k); \emptyset >\}$. Because all activities of a same session must be planned in series, for each movement $< (a, t_a^i, R_a^i)\ ;\ \emptyset >\in M$, with $i = 1, \ldots, k$, the following property must be respected:

- $[t_a^i; t_a^i + duration_a[\cap [t_b; t_b + duration_b[= \emptyset \quad \forall (b, t_b, R_b) \in Sol_{s(a)}$.

$intra_a$: This operator removes one or more scheduled activities from $s(a)$ in order to plan activity $a \in UA$. It builds a set of movements M=$\{< (a, t_a^1, R_a^1); \Upsilon^1 > , \ldots, < (a, t_a^k, R_a^k); \Upsilon^k >\}$, where Υ^i are the sets of activities belonging to session $s(a)$ to be removed from the solution. In addition to the seriality constraint, it is clear that only the activities of $s(a)$ scheduled on the time slots of $[t_a^i; t_a^i + duration_a[$ must be deleted. So, for each movement $< (a, t_a^i, R_a^i); \Upsilon^i > \ \in M$, with $i = 1, \ldots, k$, and $\Upsilon^i \subseteq SA_{s(a)}$, the following properties are verified:

- $[t_a^i; t_a^i + duration_a[\cap [t_b; t_b + duration_b[\neq \emptyset$, $\forall (b, t_b, R_b) \in \Upsilon^i$
- $[t_a^i; t_a^i + duration_a[\cap [t_b; t_b + duration_b[= \emptyset$, $\forall (b, t_b, R_b) \in \{Sol_{s(a)} \setminus \Upsilon^i\}$

$extra_a$: The principle of this operator is based on that of $intra(a)$. To schedule activity a, it builds a set of movements M for which each set Υ^i of removed activities belongs to the same randomly selected session $s' \neq s(a)$. For each movement $< (a, t_a^i, R_a^i); \Upsilon^i >\in M$, with $i \in [1; k]$ and $\Upsilon^i \subseteq SA_{s'}$, the following properties are verified:

- $[t_a^i; t_a^i + duration_a[\cap [t_b; t_b + duration_b[\neq \emptyset$, $\forall (b, t_b, R_b) \in \Upsilon^i$
- $[t_a^i; t_a^i + duration_a[\cap [t_b; t_b + duration_b[= \emptyset$, $\forall (b, t_b, R_b) \in Sol_{s(a)}$

$extra_a^+$: This operator differs from *intra* and *extra* because canceled activities come from different sessions. For each movement, the removed activities belong to S. It builds a set of movements M=$\{< (a, t_a^1, R_a^1); \Upsilon^1 >, \ldots, < (a, t_a^k, R_a^k); \Upsilon^k >$ $\}$ in order to plan activity $a \in UA$. For each movement $< (a, t_a^i, R_a^i); \Upsilon^i >\in M$, with $i \in [1; k]$ and $\Upsilon^i \subseteq SA$, the properties below are verified :

- $[t_a^i; t_a^i + duration_a[\cap [t_b; t_b + duration_b[\neq \emptyset$, $\forall (b, t_b, R_b) \in \Upsilon^i$
- $[t_a^i; t_a^i + duration_a[\cap [t_b; t_b + duration_b[= \emptyset$, $\forall (b, t_b, R_b) \in \{Sol_{s(a)} \setminus \Upsilon^i\}$

So, each operator defined above, first generates a set M of movements, then chooses one of them to apply to the current solution, according to the rule of random proportionality. This rule is built on probability p_m, associated with each movement and computed by equation 27.

$$p_m = \frac{score_m}{\sum\limits_{m' \in M} score_{m'}} \tag{27}$$

The strategy of scheduling operator selection during the iterative process of algorithm 2 still remains to be defined. This is provided by the function *selectOperator* described in section 6.5.

6.4 Diversification operator

$diversificator$ is the diversification operator. Its purpose is to partially modify the solution in order to escape from a local minimum. $diversificator$ has two operating modes depending on UA, because the latter could be empty when $diversificator$ is called. The first mode schedules all the activities in UA. The second one is used when $UA = \emptyset$ and moves some scheduled activities to other time slots. Both of these operating modes use $extra^+$ operator to schedule or move activities. Below, the details of these modes:

if $UA = \emptyset$: it selects a set $\{(a^1, t_{a^1}, R_{a^1}), \ldots, a^\beta, t_{a^\beta}, R_{a^\beta})\}$ of triplets to remove from the current solution. For each canceled triplet (a^i, t_{a^i}, R_{a^i}), with $i \in [1; \beta]$, it uses operator $extra^+$ to plan activity a^i.
Note that t_{a^i}, the previous time slot where a^i was scheduled, is considered as tabu and cannot be used by $extra^+$ at this step. This restriction was made to prevent $diversificator$ from scheduling a^i on the time slot t_{a^i} where it was previously planned.

if $UA \neq \emptyset$: in this mode, there is no need to remove triplets from the solution. For each unscheduled activity $a \in UA$, $extra^+$ operator is used to schedule it.

Note that whatever the mode used, each time $extra^+$ is used and then a movement m is applied, the canceled activities belonging to Υ_m are added to UA and are not yet considered by $diversificator$.

β represents the number of removed activities and is randomly generated between limits β_{min} and β_{max} which represent respectively the minimum and the maximum quantity of triplets canceled. β_{max} evolves according to the number of times *destructor* has been applied without an improvement of the solution. Each time the solution is improved, β_{max} is set to its initial value β_{min}, otherwise β_{max} increases by 1 each time *destructor* is used, until it reaches its preset limit $lim_{\beta_{max}}$.

6.5 Function : $selectOperator()$

Function *selectOperator* is used when *saturator* can't schedule all activities, as shown in algorithm 2. Its purpose is to determine which operator will be the most suitable to plan unscheduled activity a. In order to choose which operator to apply between $intra$, $extra$, $extra^+$, *SelectOperator* function uses two specific counters:

c^s_{extra} counts the number of times where $intra$ has been consecutively applied to session s. When it reaches its associated limit, it activates operator $extra$ to schedule a.

$c^a_{extra^+}$ determines how many iteration activities a remained consecutively unscheduled. When its limit is reached, it uses $extra^+$ to plan a.

By default, *selectOperator* uses $intra$, except if the counters reach their limits. Note that in case of equality between the two counters, $extra^+$ is always used first.

7 Instances & Computational results

7.1 Instances

In order to test our algorithms, we needed instances close to the SimUSanté problem. However, the current operation of the training center does not provide real instances. For this reason, we have generated new instances [13], from the classic instances of the CB-CTT problem [47]. These generated instances include characteristics inherent to the SimUSanté problem, and are accessible on the website [13]. Their construction process is now briefly described.

We generate an initial instance $D_0T_0C_0A_0$ which has as many employees (the teachers), rooms, time slots, sessions (the curricula) and activities (the lectures) as the original CB-CTT instance, and ensure that all the characteristics described in section 3 are present: An employee-type matches to a single employee. So there are as many employee-types as employees. Each employee is available over the horizon and their employee-type is associated with activities that correspond to the lectures the relative teacher/employee was performing on the original CB-CTT instance. All rooms have the same unique room-type. Because such an instance represents a 'too special' case of the SimUSanté instances, we have extended it and thus made it more general using the following criteria:

- Criterion $C1$ adds one random employee-type to a set of randomly chosen employees.
- Criterion $D1$ halves the availabilities to a set of randomly selected employees.
- Criterion $T1$ adds one or two new room-types, distributes the rooms into all these existing types and sets activity requirements accordingly.
- Criterion $T2$ allows the rooms to have more than one type and then adds to each randomly chosen room one existing room-type.
- Criterion $A1$ selects a set of randomly chosen activities and for each of them increases by 1 the quantity required in a randomly selected room-type. Some activities then require more than one room-type or two rooms of the same type.
- Criterion $A2$ uses the same set of activities considered by criterion A_1 and increases by 1 the quantity required in a randomly selected employee-type.

These criteria are combined to provide different instances. As an illustration, $D_0T_1C_0A_1 + D_1$ provides a new instance $D_1T_1C_0A_1$. Because the criteria involve random choices, it is possible to generate several different instances from a single variant criterion.

To test $SimUG$ and $SimULS$ we use 6 new instances derived from those of the CB-CTT: Brazil1, Italy1, Brazil2, Brazil6, Finland1 and StPaul. The characteristics of the new generated instances are presented in table 1:

7.2 Results

The mathemathical model presented in section 4 was implemented with the CPLEX solver 12.8. $SimUG$ and $SimULS$ were written in Java, and tested on an Intel I7 7500U processor. The tests of CPLEX, $SimUG$ and $SimULS$ are reported on tables 2 to 7. Column $Instancename$ corresponds to the label associated with

Table 1 Characteristics of the tested instances

Instance name	Sessions	Employees	Rooms	Activities	Time slots
Brazil1	3	8	3	27	36
Italy1	3	13	3	36	45
Brazil2	6	14	, 6	57	36
Finland1	10	18	13	139	45
Brazil6	14	30	14	126	36
StPaul	67	68	67	589	36

Table 2 Results for Brazil1 - average gap from CPLEX - $SimUG$: 37.52%, $SimULS$: 7.10%

Instance name	CPLEX	$SimUG$			$SimULS$			
		$Makespan$	$\|UA\|\%$	pen	$Makespan$	$\|UA\|\%$	pen	SD
$D_0T_0C_0A_0$	83	89	0	0	84.2	0	0	1.62
$D_0T_1C_0A_1$	84	124	3.70	36	91.1	0	0	1.67
$D_0T_1C_1A_1$	84	124	3.70	36	90.3	0	0	2.01
$D_0T_1C_1A_2$	84	124	3.70	36	91.3	0	0	1.71
$D_0T_2C_0A_1$	83	159	7.41	72	91.5	0	0	1.63
$D_1T_0C_0A_0$	83	86	0	0	84.3	0	0	1.68
$D_1T_1C_0A_1$	84	132	3.70	36	91.5	0	0	1.59
$D_1T_1C_1A_1$	84	168	7.41	72	90.5	0	0	1.73
$D_1T_1C_1A_2$	84	168	7.41	72	90.9	0	0	1.62
$D_1T_2C_0A_1$	83	164	7.41	72	91.4	0	0	1.67
average	83.6	133.8	4.44	43.2	89.99	0	0	1.69

Table 3 Results for Italy1 - average gap from CPLEX - $SimUG$: 63.09%, $SimULS$: 9.99%

Instance name	CPLEX	$SimUG$			$SimULS$			
		$Makespan$	$\|UA\|\%$	pen	$Makespan$	$\|UA\|\%$	pen	SD
$D_0T_0C_0A_0$	101	109	0	0	104.9	0	0	1.38
$D_0T_1C_0A_1$	103	199	5.56	90	113.8	0	0	1.65
$D_0T_1C_1A_1$	103	245	8.33	135	114.4	0	0	1.41
$D_0T_1C_1A_2$	103	244	8.33	135	114.3	0	0	1.64
$D_0T_2C_0A_1$	103	474	22.22	360	113.9	0	0	2.03
$D_1T_0C_0A_0$	101	107	0	0	105.1	0	0	1.87
$D_1T_1C_0A_1$	103	248	8.33	135	116.1	0	0	1.76
$D_1T_1C_1A_1$	103	294	11.11	180	117	0	0	1.73
$D_1T_1C_1A_2$	103	294	11.11	180	118.4	0	0	1.74
$D_1T_2C_0A_1$	103	566	27.78	450	118.2	0	0	1.69
average	102.6	278	10.28	177.4	113.99	0	0	1.69

the criteria used in the generating instance process, as explained before. Column $CPLEX$ gives the optimal solution when attainable in the running time limits of 24 hours. *na* specifies a non available result. For $SimUG$ and $SimULS$, the results are respectively decomposed into 3 and 4 parts: column $Makepsan$ is the value for $SimUG$ and the average over 20 runs for $SimULS$ of the objective function (equation 1), column $|UA|$ is the percentage ($SimUG$) and the average percentage ($SimULS$) of unscheduled activities, column *pen* is the penalization induced by

Table 4 Results for Brazil2 - average gap from CPLEX - $SimUG$: 46.88%, $SimULS$: 10.96%

Instance name	CPLEX	$SimUG$			$SimULS$			
		$Makespan$	$\|UA\|\%$	pen	$Makespan$	$\|UA\|\%$	pen	SD
$D_0T_0C_0A_0$	163	288	5.28	108	174.4	0	0	1.72
$D_0T_1C_0A_1$	166	395	10.53	216	188.1	0	0	1.70
$D_0T_1C_1A_1$	166	431	12.28	252	186.9	0	0	1.75
$D_0T_1C_1A_2$	166	431	12.28	252	188.3	0	0	1.12
$D_0T_2C_0A_1$	164	259	8.77	180	186.6	0	0	1.75
$D_1T_0C_0A_0$	163	220	1.75	36	175.8	0	0	1.33
$D_1T_1C_0A_1$	166	260	3.51	72	188.7	0	0	1.68
$D_1T_1C_1A_1$	166	264	3.51	72	188.1	0	0	1.56
$D_1T_1C_1A_2$	166	301	5.26	108	188.4	0	0	1.95
$D_1T_2C_0A_1$	164	257	3.51	72	187.8	0	0	1.64
average	165	310.6	6.67	136.8	185.31	0	0	1.71

Table 5 Results for Finland1

Instance name	CPLEX	$SimUG$			$SimULS$			
		$Makespan$	$\|UA\|\%$	pen	$Makespan$	$\|UA\|\%$	pen	SD
$D_0T_0C_0A_0$	na	525	2.88	180	345.9	0	0	1.62
$D_0T_1C_0A_1$	na	619	4.32	270	367.8	0	0	1.59
$D_0T_1C_1A_1$	na	575	3.60	225	368.4	0	0	0.98
$D_0T_1C_1A_2$	na	489	2.16	135	367.9	0	0	1.24
$D_0T_2C_0A_1$	na	619	4.32	270	366.5	0	0	1.29
$D_1T_0C_0A_0$	na	720	5.76	360	346.7	0	0	1.48
$D_1T_1C_0A_1$	na	770	6.47	405	368.3	0	0	1.41
$D_1T_1C_1A_1$	na	720	5.76	360	370.3	0	0	1.23
$D_1T_1C_1A_2$	na	715	5.76	360	370.1	0	0	1.31
$D_1T_2C_0A_1$	na	770	6.47	405	369.5	0	0	1.27
average	na	652.2	4.75	297	364.14	0	0	1.34

UA (with $pen = |UA| \times \alpha$ and $\alpha = |H|$ on equation (15)), and column SD represents the standard deviation over 20 runs from column $Makespan$ ($SimULS$).

CPLEX provided optimal solutions only for the smaller instance families such as $Italy1$, $Brazil1$ and $Brazil2$ (see tables 2, 3 and 4). Note that in the optimal solution, all activities are scheduled. For the others, the optimal solutions are not attained because of the prohibitive execution time of the solver.

The greedy algorithm $SimUG$ worked out valid but non optimal solutions in less than one second. $SimUG$ schedules all activities for only 4 of the 60 tested instances and schedules on average 93.55% of the activities with a standard deviation of 4.96 over all instances. The differences in number of scheduled activities between the tested instances are caused partly by criteria *betterStart* and *ActivityChoice*(). The first one returns a time slot t^* from which a session s^* can start to be planned and then time slots before t^* are never considered by $SimUG$ in order to plan s^*. As an example, in the solution proposed by $SimUG$

Table 6 Results for Brazil6

Instance name	CPLEX	*SimUG*			*SimULS*			
		Makespan	$\|UA\|\%$	*pen*	*Makespan*	$\|UA\|\%$	*pen*	*SD*
$D_0T_0C_0A_0$	383	710	6.35	288	428.5	0	0	1.03
$D_0T_1C_0A_1$	390	1030	13.49	612	435.3	0	0	0.86
$D_0T_1C_1A_1$	390	958	11.90	540	437.6	0	0	0.65
$D_0T_1C_1A_2$	na	1031	13.49	612	437.8	0	0	0.98
$D_0T_2C_0A_1$	na	886	10.32	468	437.3	0	0	0.91
$D_1T_0C_0A_0$	na	584	4.76	216	429.2	0	0	0.92
$D_1T_1C_0A_1$	na	841	8.73	396	438.2	0	0	0.89
$D_1T_1C_1A_1$	na	760	7.14	324	438.9	0	0	0.85
$D_1T_1C_1A_2$	na	873	9.52	432	438.5	0	0	0.94
$D_1T_2C_0A_1$	na	841	8.73	396	438.1	0	0	0.93
average	na	851.4	9.44	428.4	435.94	0	0	0.90

Table 7 Results for StPaul

Instance name	CPLEX	*SimUG*			*SimULS*			
		Makespan	$\|UA\|\%$	*pen*	*Makespan*	$\|UA\|\%$	*pen*	*SD*
$D_0T_0C_0A_0$	na	1591	1.02	216	1433.2	0	0	0.82
$D_0T_1C_0A_1$	na	1639	1.19	252	1458.7	0	0	0.89
$D_0T_1C_1A_1$	na	1587	1.02	216	1458.7	0	0	0.84
$D_0T_1C_1A_2$	na	2125	3.40	720	1462.5	0	0	0.92
$D_0T_2C_0A_1$	na	1639	1.19	252	1463.4	0	0	0.85
$D_1T_0C_0A_0$	na	2401	4.41	936	1438.1	0	0	0.92
$D_1T_1C_0A_1$	na	2331	4.07	864	1468.6	0	0	0.98
$D_1T_1C_1A_1$	na	2334	4.07	864	1468.6	0	0	0.95
$D_1T_1C_1A_2$	na	2785	6.11	1296	1470.2	0	0	1.01
$D_1T_2C_0A_1$	na	2331	4.07	864	1469.5	0	0	0.96
average	na	2076.3	3.06	648	1459.1	0	0	0.92

on $Brazil1-D_0T_1C_0A_1$ instance, each session starts at, or after, 1/6 of the horizon. The second one schedules the unplanned activities of s^* based on their remaining planning possibilities. These are computed from a restricted interval of time slots: the current time slot t, with $t^* \leq t$ and the earliest end $end^t_{s^*}$ of s^*. The time slots after $end^t_{s^*}$ are then not considered for these computations and lead *activityChoice* to misplace activities.

The restricted horizon mainly generates the differences in results observed between the instances. However, although the criteria *betterStart* and *activityChoice*() lead sometimes to solutions that have more unscheduled activities than others, they provide compact timetables. Indeed, without penalty *pen* added to the score of *SimUG* in equation 15 which have few idle time slots, than others with more scheduled activities but a lot of idle time slots. That is why we used *SimUG* solutions as initial solutions for *SimULS*.

The running time limit for *SimULS* has been set at two hours. *SimULS* always schedules all activities and always improves results of *SimUG*. We note an

average divergence of 9.97% between $SimULS$ and the optimal solution when it exists, with a standard deviation of 2.68. Moreover, considering the column SD, we observe for each instance that the standard deviation remains less than 2, which means that $SimULS$ always provides solutions of equivalent quality. Futhermore for each table, the results obtained are roughly the same (except for $D_0T_0C_0A_0$ and $D_1T_0C_0A_0$), which means that $SimULS$ is efficient in most cases.

For small instances, like $Brazil1$, $Italy1$, $Brazil2$ (cf. table 2, 3, 4), the main difference between the optimal solutions and that of $SimULS$ is that the number of lunch breaks is not minimized. Indeed, to obtain the optimal solution, it is necessary to start the sessions from specific time slots that allow lunch breaks to be minimized. For example: start a session on the first afternoon time slot and end it on the last morning time slot of the following day. Nevertheless the schedules obtained by $SimULS$, in all instances, are compact and quite revelant for SimUSanté.

Table 8 Operator use rate

Instance name	$diversificator$	*selectOperator*		
		$intra$	$extra$	$extra^+$
Brazil1	23.36	89.26	8.33	2.41
Italy1	16.37	86.09	8.10	5.81
Brazil2	17.74	88.17	8.55	3.28
Finland1	13.38	84.70	7.97	7.33
Brazil6	15.64	85.95	8.32	5.73
StPaul	10.12	82.84	8.81	8.55

Table 8 shows the average use rate of each operator compared to the total number of iterations. The results have been grouped into instances that have been generated from the same CB-CTT instance. We note nb_{it} the total number of iterations made by $SimULS$ in a running time of two hours and nb_{intra}, nb_{extra}, nb_{extra^+}, $nb_{diversificator}$ respectively the number of iterations where $intra$, $extra$, $extra^+$ and $diversificator$ have been used . Column *selectOperator* gives the percentage of use for the set of operators: $nb_{operator}/nb_{it} \times 100$. They have been grouped together because they are only called by function *selectOperator* which is called at each iteration. Column $diversificator$ is the percentage of use for $diversificator$ operator. Note that the rate of *saturator* is always 100% because it is called at each iteration.

We observe that the larger the size of the instance, the less the diversificator is used. Indeed, on the smaller and easier instances, once $diversificator$ has been used, $SimULS$ succeeds very quickly in planning all the activities and therefore in using $diversificator$ again. Likewise, we note that the larger and more difficult the instances, the more $extra^+$ is used to obtain a solution without unscheduled activities. This is mainly because $SimULS$ needs several iterations to successfully plan all the activities once $diversificator$ has been used, and therefore some activities remain unscheduled for quite a long time.

8 Conclusion

In this paper we have presented a study of a planning problem with resource constraints, for the health training center SimUSanté. We proposed a mathematical model and greedy algorithm $SimUG$, based on a set of choice criteria aimed at reducing the overall Makespan of training sessions, while respecting all resource and time constraints. $SimUG$ is interesting for SimuSanté because it allows immediat solutions to be obtained, in particular on easier instances. To improve the results obtained by $SimUG$, we have developed $SimULS$, a local search algorithm based on 5 movement operators, in order to plan all the activities, while minimizing the general Makespan. We experimented $SimUG$ and $SimULS$ on new instances, generated from those of CB-CTT, and integrating the SimUSanté problem characteristics. The results obtained were compared to the optimal solutions provided by the CPLEX solver. $SimUG$ schedules on average 93.55% of the activities and produced a suitable and compact basic solution in a very short time. $SimULS$ always schedules all activities, enhances results of $SimUG$ in all cases and has an average divergfence of 9.97% from optimal solutions. $SimULS$ is a good compromise because it obtains good results quickly thanks to our operators which efficiently explore the search space.

References

1. Abdullah, S.: Heuristic approaches for university timetabling problems. Ph.D. thesis, University of Nottingham (2006)
2. Abuhamdah, A., Ayob, M., Kendall, G., Sabar, N.: Population based local search for university course timetabling problems. Applied Intelligence **40**, 44–53 (2014)
3. Akkan, C., Gülcü, A.: A bi-criteria hybrid genetic algorithm with robustness objective for the course timetabling problem. Computers & Operations Research **90**, 22 – 32 (2018)
4. Al-Betar, M.A., Khader, A.T., Zaman, M.: University course timetabling using a hybrid harmony search metaheuristic algorithm. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **42**(5), 664–681 (2012)
5. Babaei, H., Karimpour, J., Hadidi, A.: A survey of approaches for university course timetabling problem. Computers & Industrial Engineering **86**, 43 – 59 (2015)
6. Badoni, R.P., Gupta, D.K.: A graph edge colouring approach for school timetabling problems. IJMOR **6**(1), 123–138 (2014)
7. Bashab, A., Ashraf, ., Ibrahim, O., Abedelgabar, E., Mohd, ., Ismail, M.A., Elsafi, A., Ahmed, A., Abraham, A.: A systematic mapping study on solving university timetabling problems using meta-heuristic algorithms (2020). DOI 10.1007/s00521-020-05110-3(
8. Bellio, R., Ceschia, S., Di Gaspero, L., Schaerf, A., Urli, T.: Feature-based Tuning of Simulated Annealing Applied to the Curriculum-Based Course Timetabling Problem. Computers & Operations Research **65** (2014)
9. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv. **35**, 268–308 (2001)
10. Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Comput. Surv. **35**, 268–308 (2001)
11. Brucker, P., Knust, S.: Resource-Constrained Project Scheduling and Timetabling. In: E. Burke, W. Erben (eds.) Practice and Theory of Automated Timetabling III, pp. 277–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
12. Burke, E.K., Mareček, J., Parkes, A.J., Rudová, H.: Penalising Patterns in Timetables: Novel Integer Programming Formulations. In: J. Kalcsics, S. Nickel (eds.) Operations Research Proceedings 2007, pp. 409–414. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
13. Caillard, S., Brisoux-Devendeville, L., Lucet, C.: Health Simulation Center Simusanté®'s Problem Benchmarks. https://mis.u-picardie.fr/en/Benchmarks-GOC/

14. Carter, M.W., Laporte, G.: Recent developments in practical course timetabling, *Lecture Notes in Computer Science*, vol. 1408. Springer, Berlin, Heidelberg. (1998)
15. Carter, M.W., Tovey, C.A.: When is the classroom assignment problem hard? Operations Research **40**, S28–S39 (1992). DOI 10.1287/opre.40.1.S28
16. Clark, M., Henz, M., Love, B.: QuikFix—A Repair-based Timetable Solver. In: Proceedings of the Seventh International Conference on the Practice and Theory of Automated Timetabling, http://www.comp.nus.edu.sg/ henz/publications/ps/PATAT2008.pdf (2008)
17. Cooper, T.B., Kingston, J.H.: The Complexity of Timetable Construction Problems. In: E. Burke, P. Ross (eds.) Practice and Theory of Automated Timetabling, pp. 281–295. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
18. Di Gaspero, L., McCollum, B., Schaerf, A.: The Second International Timetabling Competition (itc-2007): Curriculum-Based Course Timetabling (track 3). Tech. rep. (2007)
19. Gaspero, L.D., Schaerf, A.: Multi-neighbourhood local search with application to course timetabling. In: PATAT (2002)
20. Goh, S.L., Kendall, G., Sabar, N.R.: Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem. Journal of the Operational Research Society **70**(6), 873–888 (2019)
21. Gozali, A., Kurniawan, B., Weng, W., Fujimura, S.: Solving university course timetabling problem using localized island model genetic algorithm with dual dynamic migration policy. IEEJ Transactions on Electrical and Electronic Engineering **15** (2020)
22. Imran Hossain, S., Akhand, M., Shuvo, M., Siddique, N., Adeli, H.: Optimization of university course scheduling problem using particle swarm optimization with selective search. Expert Systems with Applications **127**, 9 – 24 (2019)
23. Islam, T., Perves, M., Shahriar, Z., Hasan, M.: University timetable generator using tabu search. Journal of Computer and Communications **4**, 28–37 (2016)
24. Kenekayoro, P., Zipamone, G.: Greedy Ants Colony Optimization Strategy for Solving the Curriculum Based University Course Timetabling Problem. British Journal of Mathematics & Computer Science **14(2)**, 1–10 (2016)
25. Kristiansen, S., Sørensen, M., Stidsen, T.: Integer programming for the generalized high school timetabling problem. Journal of Scheduling **18** (2015). DOI 10.1007/s10951-014-0405-x
26. Kuan, Y., Obit, J., Alfred, R.: Comparison of simulated annealing and great deluge algorithms for university course timetabling problems (uctp). Advanced Science Letters **23**, 11413–11417 (2017)
27. Lü, Z., Hao, J.K.: Adaptive tabu search for course timetabling. European Journal of Operational Research **200**, 235–244 (2010)
28. Lach, G., Lübbecke, M.: Curriculum Based Course Timetabling: New solutions to Udine benchmark instances. Annals of Operations Research **194**, 255–272 (2012)
29. Lewis, R.: A Survey of Metaheuristic-Based techniques for University Timetabling problems. OR Spectrum **30**(1), 167–190 (2008)
30. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated Local Search, pp. 320–353. Springer US (2003)
31. Matias, J., Fajardo, A., Medina, R.: Examining Genetic Algorithm with Guided Search and Self-Adaptive Neighborhood Strategies for Curriculum-Based Course Timetable Problem. In: 2018 Fourth International Conference on Advances in Computing, Communication Automation (ICACCA), pp. 1–6 (2018)
32. Mazlan, M., Makhtar, M., Khairi, A., Mohamed, M.A.: University course timetabling model using ant colony optimization algorithm approach. Indonesian Journal of Electrical Engineering and Computer Science **13**, 72–76 (2019)
33. Mazlan, M., Makhtar, M., Khairi, A., Mohamed, M.A., Rahman, M.: A Study on Optimization Methods for Solving Course Timetabling Problem in University. International Journal of Engineering and Technology(UAE) **7**, 196–200 (2018)
34. McCollum, B.: International Timetabling Competition 2007 (ITC2007). http://www.cs.qub.ac.uk/itc2007/index.htm
35. Mccollum, B.: A perspective on bridging the gap between theory and practice in university timetabling. pp. 3–23 (2006)
36. Mccollum, B., McMullan, P., Paechter, B., Lewis, R., Schaerf, A., Di Gaspero, L., Parkes, A., Qu, R., Burke, E.: Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS Journal on Computing **22**, 120–130 (2010)

37. MirHassani, S.: A computational approach to enhancing course timetabling with integer programming. Applied Mathematics and Computation **175**(1), 814–822 (2006)
38. Muklason, A., Irianti, R.G., Marom, A.: Automated course timetabling optimization using tabu-variable neighborhood search based hyper-heuristic algorithm. Procedia Computer Science **161**, 656 – 664 (2019)
39. Müller, T.: Itc2007 Solver Description: A Hybrid Approach. Annals of Operations Research **172**, 429–446 (2008)
40. Paechter, B., Gambardella, L.M., Rossi-Doria, O.: International Timetabling Competition 2002 (ITC2002). http://sferics.idsia.ch/Files/ttcomp2002/
41. Phillips, A., Walker, C., Ehrgott, M., Ryan, D.: Integer programming for minimal perturbation problems in university course timetabling. Annals of Operations Research **252** (2017)
42. Pillay, N.: A survey of school timetabling research. Annals of Operations Research **218**, 261–293 (2014)
43. Schaerf, A.: A Survey of Automated Timetabling. Artificial Intelligence Review **13**(2), 87–127 (1999)
44. Schimmelpfeng, K., Helber, S.: Application of a real-world university-course timetabling model solved by integer programming. Or Spectrum **29**(4), 783–803 (2007)
45. Stützle, T., Hoos, H.: Stochastic local search-foundations and applications (2005)
46. Thompson, J.M., Dowsland, K.A.: Variants of simulated annealing for the examination timetabling problem. Annals of Operations Research **63**, 105–128 (1996)
47. DMMP Group University of Twente, E.: High School Timetabling Project. https://www.utwente.nl/en/eemcs/dmmp/hstt/
48. Yusoff, M., Roslan, N.: Evaluation of genetic algorithm and hybrid genetic algorithm-hill climbing with elitist for lecturer university timetabling problem. In: Y. Tan, Y. Shi, B. Niu (eds.) Advances in Swarm Intelligence, pp. 363–373. Springer International Publishing (2019)