A Network Flow Based Construction for a GRASP+SA Algorithm to Solve the University Timetabling Problem

Edmar Hell Kampke(✉), Leonardo Moreli Scheideger(✉), Geraldo Regis Mauri(✉), and Maria Claudia Silva Boeres(✉)

Optimization Lab, Federal University of Espírito Santo, Vitória, ES, Brazil
edmar.kampke@ufes.br, leonardo.scheideger@aluno.ufes.br, geraldo.mauri@ufes.br, boeres@inf.ufes.br

Abstract. Educational timetabling is one of the most researched topics in the field of timetabling. This problem consists of allocating a set of lectures to available rooms and periods, considering students and teachers requests and constraints. Several mathematical models for this problem can be found in the literature. The model considered in this paper is based on courses curricula of a university, proposed in the second International Timetabling Competition (ITC-2007). A maximum flow partial solution is used together with the GRASP constructive algorithm to generate a local solution improved by Simulated Annealing. Computational experiments were performed in ITC-2007 instances, and the results were compared to the best solutions of ITC-2007 and to the literature.

Keywords: University timetabling problem · Hybrid algorithm · Network flow

1 Introduction

The timetabling problem consists of allocating activities in a timetable in order to address the constraints of the stakeholders as best as possible. This problem can be found in several areas such as organization of sporting events, transportation, work schedules of employees, and others. Specifically, educational timetabling problems allocate a sequence of lectures involving students and teachers in a fixed period of time, satisfying specific restrictions [9].

The University Timetabling Problem (UTP) has many variations. According to Burke et al. [1], timetables generation vary considerably between universities, since each of them has specific requirements. In many of them, the high number of events associated with teachers and resources, combined with a wide variety of requirements, makes the generation of timetables a difficult task.

Supported by CNPq (process 301725/2016-0) and FAPES (process 73290475/2016).

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11621, pp. 215–231, 2019.
https://doi.org/10.1007/978-3-030-24302-9_16

UTP is classified as NP-complete for most formulations [13]. In this way, the optimal solution can only be guaranteed for small instances, which is not the case of most universities.

In order to encourage the study of different approaches to timetabling problems, *The International Series of Conferences on the Practice and Theory of Automated Timetabling* (PATAT) promoted three competitions (2002, 2007 and 2011) of educational timetabling, called *International Timetabling Competition* (ITC). The first two competitions are dedicated to universities and the last, to high school timetabling. These competitions encouraged the research community to promote a broader debate in timetabling problems and also made novel instances close to the reality of universities.

The ITC-2007 addressed the UTP in three different formulations, and the competitors could participate independently in each one of them. Each formulation has its own set of instances. The first formulation is specific for the application of final term exams. The second and third formulations deal with the weekly allocation of lectures at a university. The difference between them is that the second formulation (called *post-enrollment*) addresses the problem by considering student enrollment data, while the third (called *curriculum-based*) takes into account the groups of courses with a common set of students and therefore the lectures can not be allocated at the same time.

Since many resolution techniques have been implemented and/or improved after ITC-2007, in this work we adopt the third formulation established by ITC-2007, the Curriculum-Based Course Timetabling (CB-CTT) [12], in order to facilitate the comparison with the results obtained by other algorithms from the literature.

First place of ITC-2007 went to Müller [11] that solved the CB-CTT formulation using *Conflict-based Statistics* to generate the initial solution and *Hill Climbing* combined with *Great Deluge* and *Simulated Annealing* for refinement of the solution. The second place was given to Lü and Hao [10] with an Adaptive Tabu Search that has been applied successfully at the same formulation.

A recent work (Kiefer et al. [7]) applies the metaheuristic *Adaptive Large Neighborhood Search*, obtaining the best results known so far for 5 of the 21 instances of the competition.

A prior work of Kampke et al. [6] includes the proposal of a GRASP algorithm with *Hill Climbing* and *Simulated Annealing* as local search methods and a *Path-Relinking* procedure to intensify the search for good solutions of the CB-CTT addressed problem. The authors also applied new movements for generating neighbors in the local search phase. They analyzed the movements in order to choose that generate the best results such as *Kempe Chain*, *Move* and *Swap* [5].

Several approaches to solve timetabling problems are presented in [13], among which we highlight the use of network flow to model the problem combined with algorithms, such as Ford and Fulkerson algorithm [4], to find a solution.

In this work, the CB-CTT formulation is modeled as a maximum network flow problem and two algorithms are employed to solve it during the construction of an initial solution for the GRASP proposed by Kampke et al. [6]. To the

best of our knowledge it is the first time that CB-CTT formulation for UTP is modeled as a network flow. This new proposal is the basis of an alternative GRASP constructive algorithm proposed in this work. The GRASP used here is the same of [6], except for the replacement of the constructive algorithm by that described in Sect. 4.2.

This paper is organized as follows. In Sect. 2 we describe the CB-CTT formulation for the university timetabling problem, adopted in ITC-2007. Sections 3 and 4 outlines, respectively, the modeling using network flow and the algorithms that we have implemented. Computational experiments results are reported in Sect. 5 and conclusions are drawn in Sect. 6.

2 Problem Description

The CB-CTT formulation for UTP considers a set of courses C where each course c have a number of lectures l_c to be taught by a teacher. This formulation handles the weekly assignment of all lectures at a university according to a timetable T divided into $|P|$ periods, where P is a set of periods and $|P| = d \times q$. d and q are respectively the number of weekly school days and the fixed number of periods each day is divided.

It is worth noting that for each course, a set of unavailable periods is given. In addition, each course is associated to a minimum number of days, among which its lectures should be spread and attended by a fixed number of students. As previously stated in this work, a course can belong to some university curricula and courses on the same curriculum have students in common. A set R of rooms is also given with each room characterized by its capacity.

Therefore, a solution of this problem represents the assignment of a pair period/room (timeslot) to all lectures of courses for a set of curricula so as to satisfy a set of *hard* constraints and to minimize the violations of *soft* constraints. The *hard* (denoted by **H1** to **H4**) and *soft* (denoted by **S1** to **S4**) constraints are summarized as follows:

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

- **S1.** *Room Capacity:* For each lecture, the number of students attending the course should not be greater than the capacity of the room hosting the lecture. Each surplus student is counted as a violation.
- **S2.** *Room Stability:* All lectures of a course should be scheduled at the same room. If this is impossible, the number of occupied rooms for these lectures should be as few as possible. Each distinct room is counted as a violation.

- **S3.** *Minimum Working Days:* The lectures of a course should be spread over a minimum number of days. Each day below the minimum is counted as a violation.
- **S4.** *Curriculum Compactness:* For a given curriculum, a violation is counted if there is one lecture not adjacent to any other lecture belonging to the same curriculum within the same day, which means that the weekly schedule of students should be as compact as possible.

The objective function $f(s)$ defines the quality of a feasible solution s and is given by $f(s) = \sum_{i=1}^{4} \alpha_i \cdot |S_i|$, where α_i and $|S_i|$ are respectively the penalty and total number of violations of each soft constraint i. The CB-CTT formulation for UTP, adopted in ITC-2007, defines the values of α_i: $\alpha_1 = 1$, $\alpha_2 = 1$, $\alpha_3 = 5$ and $\alpha_4 = 2$. We recall that in a feasible solution, the number of conflicts of hard constraints is zero, in other words, $\sum_{j=1}^{4} |H_j| = 0$, where $|H_j|$ are the total number of violations of each hard constraint j.

We briefly described some elements of the CB-CTT formulation for UTP. The whole formulation description can be found in [2].

3 A CB-CTT Network Flow Formulation

In this section we present a CB-CTT reformulation we proposed as a network flow model.

Maximum flow is a widely known problem in the field of Combinatorial Optimization, appearing in several important applications. This problem involves finding a maximum feasible flow through a single-source to a single-sink node of a flow network without exceeding the capacity of its arcs.

Therefore, let G be a flow network represented by the digraph $G = (V, A, \Phi)$ where V and A are respectively the set of vertices and arcs and $\Phi = \{\Phi_{ij}\}$ is a vector, with $|\Phi| = |A| + 1$. The arc arrows indicate the flow direction and the Φ flow vector terms Φ_{ij}, $\forall (i, j) \in A + (\tau, \mu)$, are the flow passing in the arc from vertice i to vertice j. The maximum flow passing over the network G is computed as the capacity $\Phi_{\tau\mu}$ of a dummy arc which links the network sink τ to its source μ. Thus, the flow crosses the network passing by transshipment nodes between μ and τ in such a way that it should be possible to reach every vertice of G from μ and each vertice of G reach τ.

In this work, we propose the CB-CTT reformulation as a maximum flow problem. In this case, the network G is composed of a source μ and a sink τ that link two layers of transshipment nodes. The first layer consists of $|C|$ vertices representing the courses of set C and the second represent the timeslots listed in the $P \times R$ set. A mathematical programming model for the CB-CTT reformulation is based on the classical model of the maximum flow problem and is given in Eqs. (1)–(6).

$$\text{Maximize} \quad \Phi_{\tau\mu} \tag{1}$$

subject to:

$$0 \leq \Phi_{\mu c} \leq l_c \qquad \forall c \in C, \tag{2}$$

$$0 \leq \Phi_{cj} \leq 1 \qquad \forall c \in C, j \in P \times R, \tag{3}$$

$$0 \leq \Phi_{j\tau} \leq 1 \qquad \forall j \in P \times R, \tag{4}$$

$$\sum_{\{r|(k,r)\in A\}} \Phi_{kr} - \sum_{\{r|(r,k)\in A\}} \Phi_{kr} = \begin{cases} \Phi_{\tau\mu}, & if\ k = \mu \\ 0, & \forall k \in V - \{\mu, \tau\} \\ -\Phi_{\tau\mu}, & if\ k = \tau \end{cases}, \tag{5}$$

$$\Phi_{ij} \in \mathbb{Z}^+, \quad \forall (i,j) \in A, \tag{6}$$

This model seeks to assign flows on the $|A|$ arcs of G such that its whole flow ($\Phi_{\tau\mu}$) is maximized (objective function (1)), satisfying the set of constraints (2) to (6).

Constraints (2) to (4) limit the minimum and maximum capacities for each arc of G, while constraints (5) impose flow conservation. The constraints (6) define the domain of the variables.

It is worth highlighting that the maximum capacities of the arcs mentioned in constraints (2), which emanate from source μ to the set of vertices representing C, refer to the number of lectures of each course $c \in C$. The remaining sets of capacity constraints (Eqs. (3) and (4)), related to the set of arcs between the intermediate network vertices layers (representing C and $P \times R$ of timeslots sets) and those linking the last layer to the sink τ, must be at most 1. These capacities limits occur since a course must have at most one lecture allocated in a timeslot, which can in turn, be only occupied by a single lecture.

Figure 1 shows an input file pattern defined by ITC-2007 for a toy instance and its corresponding representation as a network flow is presented in Fig. 2. This file pattern can be divided into five parts in this order: header, information of courses, rooms, curricula and unavailability constraints. The header consists of seven lines, each containing, respectively, the instance name and number of courses, rooms, days, periods per day, curricula and unavailability constraints. The courses section shows, by line, the information for each course: its and the teacher names, number of lectures per week, minimum number of lecture days and number of students. Next, a sequence of $|R|$ lines with all rooms names and capacities. On the way of the list of rooms, the curricula are organized by their names followed by the number and names of the courses attached to each of them. The last file pattern section contains the unavailability constraints, each represented by the name of a course followed by the day and period in which it can not be allocated.

Figure 2 illustrates the corresponding network flow representation. Besides its source and destination nodes, the network consists of two further layers of vertices, one representing the set of courses C and the other, the set $P \times R$ of timeslots. The network arcs labels indicate their capacity. Arcs linking the source μ to the first layer of nodes (set C) represent the number of lectures that should be allocated for each course. In this example, this value is two. The capacities

of the set of arcs between the intermediate vertices layers are at most 1. At last, all the arcs linking the $P \times R$ layer to τ have unitary capacities.

```
Name: Toy3
Courses: 3
Rooms: 2
Days: 2
Periods_per_day: 2
Curricula: 2
Constraints: 2

COURSES:
DiscMath Edmar 2 2 40
CompProg Geraldo 2 2 35
GraphThe Edmar 2 3 20

ROOMS:
rA 38
rB 32

CURRICULA:
Cur1 2 DiscMath CompProg
Cur2 1 GraphThe

UNAVAILABILITY_CONSTRAINTS:
CompProg 0 0
CompProg 1 1
END.
```

Fig. 1. An input file pattern for a toy example.

In all cases, a flow equal to one indicates the allocation of a lecture of a course to a timeslot. For instance, the arc linking the vertice *DiscMath* to $R_0Q_1D_0$ with capacity 1 means the allocation of a lecture of the *DiscMath* course to the timeslot representing the period 1 of day 0 in room 0. Null flow indicates that no lecture of *DiscMath* is allocated in this timeslot.

Therefore, this formulation ensures that the hard constraints **H2** (*Room Occupancy*) and **H4** (*Availability*) are satisfied but does not guarantee the same for **H1** (*Lectures*) and **H3** (*Conflicts*).

This formulation has no control over the soft constraints. The adopted strategy is the application of the maximum flow algorithm in order to allocate the lectures in different days and in the same room, assisting **S2** (*Room Stability*) and **S3** (*Minimum Working Days*), which are almost always not violated.

The order of courses in the network flow is defined randomly, whereas the order of timeslots is fixed, starting from room 0 at period 0 of day 0 and then varying the days, periods and rooms, in this order.

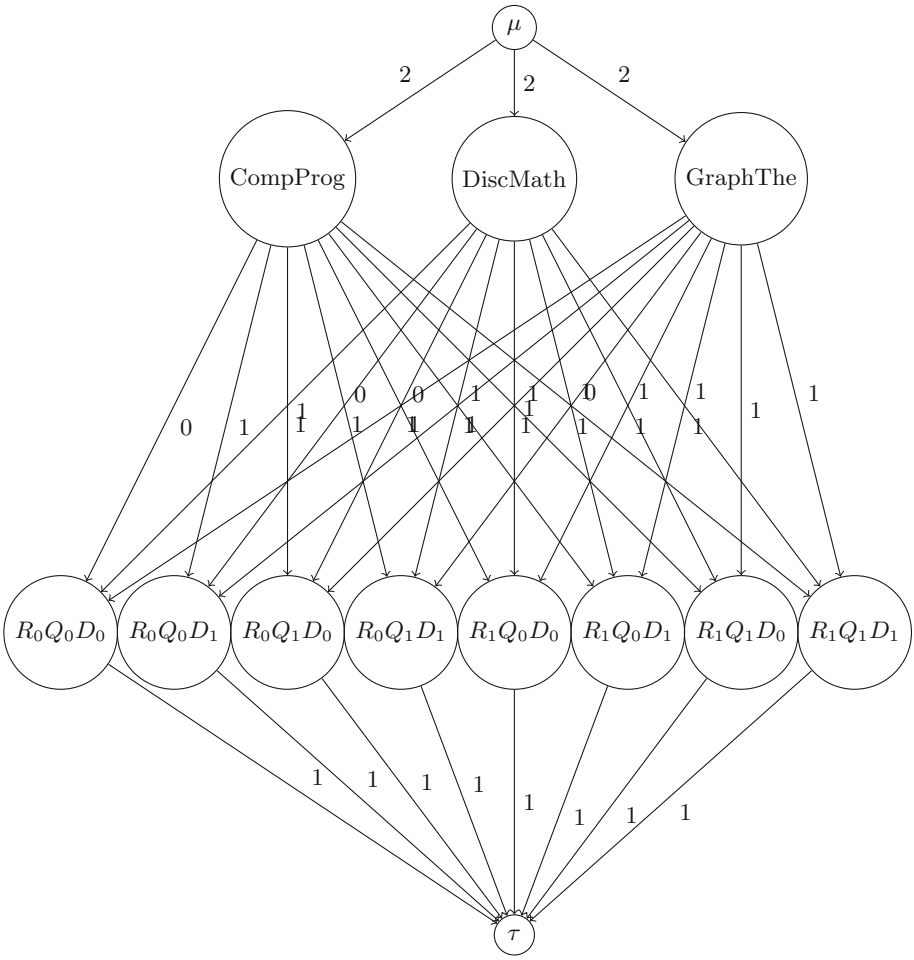

Fig. 2. Toy instance network flow representation.

4 Algorithms

In this work we used the CB-CTT reformulation as a network flow model (see Sect. 3) to apply maximum flow algorithms and thus to construct a partial solution, since there is no guarantee that all lectures will be allocated.

The GRASP algorithm proposed by Kampke et al. [6] is an iterative method in which two phases are applied in each iteration: construction of a complete initial solution (IS_{CBCTT} method) and local search (Simulated Annealing metaheuristic). In this work, we propose an adaptation in this GRASP algorithm, specifically in the construction of an initial solution phase. Here, we construct a partial solution by maximum flow algorithms that will then be completed by IS_{CBCTT} method. The implemented algorithms are presented below.

4.1 Maximum Flow

The Maximum Flow (MF) problem consists of finding the largest possible flow from one point (source) to another (sink) in a network flow. The algorithm proposed by Ford and Fulkerson [4] was the first to solve the problem and is based on finding, at each iteration, a flow path between source and sink, increasing the flow of the path arcs according to the smallest remaining capacity of them.

Finding paths which a flow can be sent through is the main routine of the Ford and Fulkerson algorithm. The way this path is computed determines the algorithm computational complexity. The Ford and Fulkerson algorithm implements a Depth-First Search (DFS) of the source to the sink in the network. The efficiency of the algorithm can be improved if DFS is replaced by a Breadth-First Search (BFS), trying firstly the paths with the least number of arcs. The algorithm with this modification was originally proposed by Edmonds and Karp [3].

Two versions of the MF algorithm are used and compared in this work, the original proposed by Ford and Fulkerson (MF-D) and the revised and improved version, by Edmonds and Karp (MF-B).

4.2 Initial Solution

In both versions of the MF algorithm, two modifications were made to improve the quality of the initial solution. The first modification consists in changing the order in which timeslots vertices (third level of the network flow) are visited: given a course c, timeslots with room capacity greater than the number of students enrolled in c should be visited first and in ascending order, according to the room capacity. In the following, timeslots with room capacities smaller than the number of students enrolled in c are visited in descending order by the capacity of the rooms.

The second modification consists of checking, for each visited timeslot vertice, if allocating a lecture in the timeslot violates the hard constraints **H1** (*Lectures*) and **H3** (*Conflicts*). This check prevents invalid allocations, but still does not guarantee the feasibility of initial solution as, in some cases, there will be no longer valid timeslots for the allocation. If this is the case, the lecture will not be allocated but will be added to the set of unallocated lectures $\overline{L}$, which is initially empty.

Both the MF algorithm versions implemented in this work visit the vertices of courses in sequential order, that is, the courses that are to the left (second level of the network flow) will be visited before those to the right. Thus, depending on the order in which courses are in network flow, eventually some lectures do not have more timeslots available and also are added in $\overline{L}$.

Even if, at the end of execution, the MF algorithm versions fail to obtain a solution using all lectures ($\overline{L} \neq \emptyset$), the partial solution and $\overline{L}$ are then sent to the GRASP constructive method IS_{CBCTT} of Kampke et al. [6].

Therefore, in this work both versions of the MF algorithm try to construct a solution with all available lectures. If it is not possible, IS_{CBCTT} is executed to allocate the remaining lectures and obtain a complete solution, which is then sent to local search method of GRASP.

4.3 Local Search

As the GRASP proposed by Kampke et al. [6] is used in this work with an alternative constructive method, we remember its local search procedure is the Simulated Annealing (SA) [8] algorithm that evolves over a neighborhood of a solution s. Let $N_m(s)$ be a neighborhood of s defined by a given movement m.

To generate a neighbor solution $s' = s \oplus m$, $s' \in N_m(s)$, a movement m is applied in the current solution s. The symbol $\oplus$ represents the transformation imposed over s by the movement m.

In this work we consider m the *Lecture Move* movement proposed by Müller [11] and selected as the best among nine tested in Kampke et al. [5]. For this movement, a lecture l_1 allocated in timeslot t_1 is randomly selected. Then a timeslot t_2 is also randomly selected. If t_2 is an empty timeslot, l_1 is moved to t_2. Otherwise, the movement consists of exchanging the lecture l_1 from timeslot t_1 to timeslot t_2, and the lecture l_2 from timeslot t_2 to timeslot t_1.

The SA algorithm has five main parameters: the initial solution s, the initial temperature T_i, the final temperature T_f, the cooling rate β and $|N(s)|$, where $N(s)$ is a subset of $N_m(s)$ defined at each temperature T_j, $j = i, \ldots, f$.

The algorithm starts from T_i which is continuously cooled to T_f. At each temperature T_j, $j = i, \ldots, f$, $|N(s)|$ neighbors are generated. For a given temperature T_j, if one of the neighbors generated is better than the incumbent solution, it is updated, otherwise, the neighbor solution can be accepted with probability equal to $e^{-\Delta f/T_j}$, where Δf is the difference in the objective functions values f of neighbor and current solutions.

4.4 GMF Proposed Algorithm

This section presents the hybrid algorithm GRASP+MF (MF-D or MF-B) + IS_{CBCTT}, denoted GMF-D and GMF-B respectively for MF-D and MF-B versions, proposed to solve the CB-CTT reformulation as a network flow.

The pseudocode of the whole method proposed in this work is presented in Algorithm 1.

Algorithm 1. GRASP with Maximum Flow algorithm and IS_{CBCTT}

Input: Maximum number of iterations $maxIter$
Output: Best solution found s^*

```
1  f* ← ∞
2  i ← 0
3  while i < maxIter do
4  |  s1 ← GMF()
5  |  L̄ ← LecturesNotAlocated(s1)
6  |  s2 ← IS_CBCTT(s1, L̄)
7  |  s ← SA(s2)
8  |  if f(s) < f* then
9  |  |  s* ← s
10 |  |  f* ← f(s)
11 |  end
12 |  i ← i+1
13 end
```

Note in Algorithm 1 that GMF as long as it does not satisfy the stop condition (line 3), repeatedly (lines 4–12) construct a solution (lines 4–6) and performs a local search phase (line 7). In construction phase, line 4 represents the construction of a partial solution s_1 by one of MF algorithm versions, line 5 presents the definition of set $\overline{L}$ of lectures not allocated in s_1 and line 6 refers to the execution of IS_{CBCTT} which is responsible for allocating the lectures of $\overline{L}$ in s_1 obtaining a complete solution s_2. Finally, lines 8–11 ensure that the best solution found by the GMF algorithm is stored, while line 12 ensures that the proposed algorithm ends.

5 Computational Experiments

The methods GMF-D and GMF-B described in the last section were submitted, as an alternative constructive algorithm for GRASP+SA, to computational experiments performed on the ITC-2007 instances *comp01–comp21*. The obtained results are presented, analysed and also compared with those found on Kampke et al. [6], Kiefer et al. [7] and ITC-2007 winners.

The code was written in C++ language, compiled using *g++* version 4.8.4 with the -O3 optimization flag and executed on a computer with Intel Core i5-3570 CPU 3.40 GHz × 4, 8 GB memory and Ubuntu 14.04 64 bit.

The next subsections present the parameters required for the GRASP+SA algorithm (Sect. 5.1) and discuss the results achieved (Sect. 5.2).

5.1 Parameters

GRASP has only two parameters to adjust: the *threshold* α, which is used in IS_{CBCTT} method to determine the size of the Restricted Candidates List (RCL),

and the maximum number of iterations *maxIter*. The SA metaheuristic, used as the GRASP+SA local search method, has four parameters: the initial temperature t_i, the final temperature t_f, the cooling rate β and the number of neighbors $|N(s)|$ generated in each temperature T_j, $j = i, \ldots, f$.

In this work, the *maxIter* parameter was replaced by unit of time (*timeout*), in order to be faithful to the rules imposed by the ITC-2007, that provided an executable tool, available in [12], for benchmarking all competitors machines. For the same reasons, Kampke et al. [6] and Kiefer et al. [7] also used this tool to define the execution time limits for their algorithms. In order to enable a fair comparison among all algorithms tested, the tool stipulated a timeout of 220 s for the machine used in this work for the computational tests.

The GRASP and SA parameter set is based on Kampke et al. [6] and afterwards empirically calibrated. Table 1 shows all parameters values used.

Table 1. GRASP+SA parameters values.

Parameter	Value
t_i	17.3
t_f	0.003
β	0.9999
$\|N(s)\|$	500
α	0.1
Timeout	220 s

5.2 Discussion

Computational tests were performed, for each instance and algorithm version, considering 10 runs with different seeds for generating random numbers. The results analysis is carried out into three main steps: (i) the solutions obtained by the algorithms GMF-D and GMF-B, together with their respective versions without **H1** (*Lectures*) and **H3** (*Conflicts*) hard constraints inspection (see Sect. 4.2), denoted as GMF-D^H and GMF-B^H, are all confronted with each other; (ii) from the conclusions of (i), we observed GMF-D outperformed the other algorithms. For this reason, the goal of this second step is to determine the number of times over all executions (success rate) that GMF-D succeed in achieving a complete solution of good quality, without using IS_{CBCTT}. The results of the runs considering or not the IS_{CBCTT} to construct a solution are respectively denoted as GMF-D$^+$ and GMF-D$^-$ in Table 3; and (iii) results comparison with those of the aforementioned literature works. In the following, these three steps are detailed.

Table 2 presents the results obtained for GMF-D, GMF-B, GMF-D^H and GMF-B^H. The first column of this table indicates the instance names. From the second to the fifth column, the f average value from all GRASP+SA executions, using each of the proposed algorithms, are presented. The remaining columns

Table 2. Results of GRASP with proposed algorithms to construct a partial solution.

Instances	f average				Time average ($\times 10^{-3}$)			
	GMF-D	GMF-B	GMF-D^H	GMF-B^H	GMF-D	GMF-B	GMF-D^H	GMF-B^H
comp01	**5.0**	**5.0**	**5.0**	**5.0**	0.012	0.105	**0.004**	0.007
comp02	61.5	60.3	62.9	**59.8**	0.685	2.235	**0.014**	0.037
comp03	85.2	**83.2**	86.0	86.7	0.363	1.359	**0.013**	0.033
comp04	**39.5**	39.7	39.7	39.9	0.066	1.025	**0.013**	0.035
comp05	**342.5**	344.9	347.7	351.1	12.365	20.125	**0.026**	0.050
comp06	**58.8**	59.5	59.8	61.1	0.166	1.523	**0.019**	0.052
comp07	28.8	**27.5**	28.4	31.0	0.323	2.390	**0.023**	0.064
comp08	44.6	45.5	**44.5**	45.4	0.065	1.252	**0.014**	0.040
comp09	**106.6**	109.8	106.8	108.9	0.109	1.101	**0.014**	0.039
comp10	25.1	25.5	**24.6**	28.0	0.237	1.607	**0.019**	0.052
comp11	**0.0**	**0.0**	**0.0**	**0.0**	0.009	0.162	**0.005**	0.010
comp12	**350.9**	354.2	360.0	355.1	0.828	1.468	**0.029**	0.059
comp13	75.6	76.2	76.1	**75.5**	0.079	1.206	**0.014**	0.041
comp14	**60.4**	61.4	60.9	61.5	0.085	0.793	**0.013**	0.036
comp15	86.2	84.8	**82.8**	84.7	0.352	0.965	**0.013**	0.033
comp16	44.1	45.3	44.2	**42.8**	0.371	2.024	**0.020**	0.056
comp17	85.3	85.8	**82.2**	84.7	0.143	1.272	**0.017**	0.045
comp18	86.0	**85.4**	87.7	86.7	0.016	0.192	**0.009**	0.019
comp19	72.3	**69.4**	69.6	70.1	0.706	1.298	**0.012**	0.031
comp20	43.5	**41.1**	44.3	44.1	0.325	1.941	**0.021**	0.058
comp21	111.3	111.2	**110.0**	113.2	0.690	2.178	**0.017**	0.045
Average	**86.34**	86.46	86.82	87.40	0.857	2.201	**0.016**	0.040

depict the respective average times. We can observe that GMF-D presents the best results on average. Nevertheless, the GMF-D mean was only 0.14% higher than that of GMF-B, even if, for 8 instances, the latter performed better than the former. Similar behaviour happens with GMF-D^H and GMF-B^H. In this case, the first algorithm mean was only 0.66% better than the second and it occurs just the opposite for 6 instances. In general, DFS-based algorithms presented better results, once this kind of search seems to facilitate the allocation of lectures of the same course into consecutive timeslots of the flow graph. Thus, the way these timeslots are organized in the flow graph promote the allocation of the lectures on different days in the same room, avoiding the violation of soft constraints **S2** (*Room Stability*) and **S3** (*Minimum Working Days*).

It is possible to notice that the algorithms with feasibility checking perform better. These results indicate that feasible solutions as input for SA favored the achievement of slightly better results, once the differences on GMF-D and GMF-B average results are respectively 0.55% and 1.08%. Therefore we can conclude that SA performed satisfactorily, since it was able to find feasible and quality solutions eventually from infeasible solutions.

Table 3. Analysis of GMF-D algorithm in construction of a partial solution.

Instances	% Sucess	f average		Time average ($\times 10^{-3}$)	
		GMF-D$^-$	GMF-D$^+$	GMF-D$^-$	GMF-D$^+$
comp01	54.55	298.32	357.46	0.101	0.195
comp02	0.00	-	716.71	-	6.328
comp03	0.00	-	667.05	-	3.702
comp04	22.22	565.55	540.93	0.277	0.594
comp05	0.00	-	1615.88	-	72.026
comp06	2.50	798.00	754.50	0.440	1.307
comp07	0.00	-	828.00	-	2.646
comp08	3.33	604.67	594.51	0.312	0.787
comp09	2.50	683.50	676.42	0.276	1.031
comp10	0.00	-	707.40	-	1.874
comp11	100.00	144.10	-	0.090	-
comp12	0.00	-	1545.90	-	5.753
comp13	36.67	582.39	592.26	0.362	0.727
comp14	6.67	621.17	607.13	0.280	0.760
comp15	0.00	-	677.66	-	3.369
comp16	0.00	-	713.79	-	2.554
comp17	0.00	-	734.65	-	1.924
comp18	83.75	531.93	540.38	0.123	0.181
comp19	0.00	-	603.25	-	5.009
comp20	0.00	-	931.00	-	2.176
comp21	0.00	-	790.88	-	4.146

Considering the CPU time required to construct a solution, we can observe in Table 2 that GMF-B and GMF-B^H run approximately 2.5 times, on average, slower than their corresponding GMF-D and GMF-D^H versions. The difference in performance can be explained by the structure of the network flow. In this network, the distance between the source μ and the sink τ is fixed and equal to exactly four levels, driving the network growth only in breadth, thus the number of vertices DFS needs to visit is much smaller than BFS.

Since the overall results of GMF-D were better, the second results analysis step is focused in the information pointed out in Table 3. According to the Table 3, the first columns is for the instances denomination and the second column shows the percentage of times that GMF-D method has built a complete solution, that is, without using IS_{CBCTT}. In the third and fourth columns are presented the f average of the solutions constructed respectively without or with IS_{CBCTT}. The respective CPU times are presented in the fifth and last columns.

Table 4. Comparison between GMF-D results and Kampke et al. [6] previous work.

Instances	Kampke[a]	GMF-D	Instances	Kampke[a]	GMF-D
comp01	**5**	**5**	comp12	375	**341**
comp02	73	**54**	comp13	97	**71**
comp03	98	**76**	comp14	72	**58**
comp04	48	**35**	comp15	101	**80**
comp05	409	**326**	comp16	69	**41**
comp06	75	**52**	comp17	105	**80**
comp07	36	**25**	comp18	102	**82**
comp08	58	**42**	comp19	87	**68**
comp09	119	**103**	comp20	88	**38**
comp10	41	**20**	comp21	136	**100**
comp11	**0**	**0**	Average	104.48	**80.81**

[a] *GRASP+SA+Path-Relinking*, [6]

We observed that the IS_{CBCTT} procedure run for instance *comp11* was not really necessary since GMF-D succeeded in generating a complete solution on all GRASP+SA executions. However, in the 12 other instances, the opposite occurred, that is, GMF-D algorithm was not able to construct a complete solution. In the remaining 8 instances, GMF-D achieved this result only a few times and in these cases, the quality of the solution constructed was, on average, 0.47% worse than those obtained combining GMF-D with IS_{CBCTT}. Nevertheless, the GMF-D algorithm, when constructing a complete solution, is on average 61.11% faster than when it needs IS_{CBCTT} to complete the solution.

The last analysis step is related to Tables 4 and 5. These tables compare the GMF-D results with those of Kampke et al. [6], Müller [11], Lü & Hao [10], and Kiefer et al. [7]. In the work of Kampke et al. [6], only the best results obtained from the 10 runs of its GRASP+SA+Path-Relinking algorithm are presented. In order to promote a fair comparison, the best solutions achieved in all executions of GMF-D are also presented in Table 4. However, in the works of Müller [11], Lü and Hao [10] and Kiefer et al. [7] are reported the f average over the 10 runs performed and consequently, in Table 5, these values are compared with the correspondent GMF-D results.

From Table 4 we can see that the results of the proposed algorithm improved on average 22.66% the results of Kampke et al. [6]. It is also noted that the GMF-D algorithm outperform results in all instances. The quality of results of GMF-D algorithm are already expected, since the proposed method constructs a partial solution without violations of hard constraints and also avoiding the violation of soft constraints **S2** and **S3**, since the algorithm proposed by Kampke et al. [6] uses the IS_{CBCTT} method to construct a complete solution also without violations of hard constraints, but does not verify the violation of soft constraints in a specific way.

Table 5. Comparison between GMF-D and some of the best results in the literature.

Instances	Müller[a]	Lü& Hao[a]	Kiefer[b]	GMF-D
comp01	**5.0**	**5.0**	**5.0**	**5.0**
comp02	61.3	60.6	**41.5**	61.5
comp03	94.8	86.6	**71.7**	85.2
comp04	42.8	47.9	**35.1**	39.5
comp05	343.5	328.5	**305.2**	342.5
comp06	56.8	69.9	**47.8**	58.8
comp07	33.9	28.2	**14.5**	28.8
comp08	46.5	51.4	**41.0**	44.6
comp09	113.1	113.2	**102.8**	106.6
comp10	21.3	38.0	**14.3**	25.1
comp11	**0.0**	**0.0**	**0.0**	**0.0**
comp12	351.6	365.0	**319.4**	350.9
comp13	73.9	76.2	**60.7**	75.6
comp14	61.8	62.9	**54.1**	60.4
comp15	94.8	87.8	**72.1**	86.2
comp16	41.2	53.7	**33.8**	44.1
comp17	86.6	100.5	**75.7**	85.3
comp18	91.7	82.6	**66.9**	86.0
comp19	68.8	75.0	**62.6**	72.3
comp20	34.3	58.2	**27.2**	43.5
comp21	108.0	125.3	**97.0**	111.3
Average	87.22	91.26	**73.73**	86.34

[a] http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm
[b] *Adaptive Large Neighborhood Search* (ALNS), [7]

In addition, the metaheuristic SA used as local search method in this work employs the *Lecture Move* movement to generate neighbor solutions, while only the classical *Move* and *Swap* movements are used in [6]. As reported by Kampke et al. [5], the *Lecture Move* is the best movement to be applied to CB-CTT formulation for UTP.

Observing the results presented in Table 5 and comparing them to each other, it can be concluded that the results presented by GMF-D algorithm surpassed on average the algorithms proposed by Müller [11] and Lü and Hao [10], respectively presented in the second and third columns, which are the winners of the ITC-2007. Specifically in comparison to Müller results, the GMF-D algorithm outperformed the results in 11 instances, in addition to the two instances (*comp01* and *comp11*) which the average are identical. In summary, the GMF-D algorithm found results with average improvements of 1.01% and 5.39% when compared to those found by Müller and Lü & Hao, respectively.

Finally, the results obtained by the ALNS [7] were 14.61% better than those from the GMF-D, on average. The ALNS algorithm proposed by Kiefer et al. [7] employs operators to destroy and reconstruct solutions allowing a greater diversification than that provided by the algorithm proposed in this work.

6 Conclusions

In this work, we presented a CB-CTT reformulation as a network flow model and the application of GMF-D and GMF-B algorithms to find the maximum flow which represents a partial solution in the original CB-CTT formulation. Then, we incorporated this algorithm to IS_{CBCTT} procedure, previously proposed by Kampke et al. [6], in order to conclude the construction of a solution, submitted to a SA metaheuristic, which performs the GRASP local search phase.

Considering the official results from ITC-2007 we observed that the proposed GRASP+SA combined with the maximum network flow algorithm was able to present very competitive solutions, and it could be ranked in the first place when compared to the results obtained by the other competitors. On average, the GMF-D results were equal or better than those found by Müller and Lü & Hao in 13 and 17 of the 21 instances, respectively.

The input data and the source codes implemented in this work are available in https://bitbucket.org/Moreli04/timetabling-moreli.

References

1. Burke, E., Jackson, K., Kingston, J.H., Weare, R.: Automated university timetabling: the state of the art. Comput. J. **40**(9), 565–571 (1997)
2. Di Gaspero, L., Schaerf, A., McCollum, B.: The second international timetabling competition (ITC-2007): curriculum-based course timetabling (track 3). Technical report (2007)
3. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM **19**(2), 248–264 (1972)
4. Ford Jr., L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press, Princeton (1962)
5. Kampke, E.H., Segatto, E.A., Boeres, M.C.S., Rangel, M.C., Mauri, G.R.: Neighborhood analysis on the university timetabling problem. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10406, pp. 148–164. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62398-6_11
6. Kampke, E.H., de Souza Rocha, W., Boeres, M.C.S., Rangel, M.C.: A GRASP algorithm with path relinking for the university courses timetabling problem. In: Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, vol. 3, pp. 1081–1087 (2015)
7. Kiefer, A., Hartl, R.F., Schnell, A.: Adaptive large neighborhood search for the curriculum-based course timetabling problem. Ann. Oper. Res. **252**(2), 255–282 (2017)
8. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)

9. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum **30**(1), 167–190 (2008)
10. Lü, Z., Hao, J.K.: Adaptive tabu search for course timetabling. Eur. J. Oper. Res. **200**(1), 235–244 (2010)
11. Müller, T.: ITC2007 solver description: a hybrid approach. Ann. Oper. Res. **172**(1), 429 (2009)
12. PATAT: International timetabling competition (2008). http://www.cs.qub.ac.uk/itc2007
13. Schaerf, A.: A survey of automated timetabling. Artif. Intell. Rev. **13**(2), 87–127 (1999)