Computers & Operations Research 52 (2014) 29–38

Contents lists available at ScienceDirect

Computers & Operations Research

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

A fix-and-optimize heuristic for the high school timetabling problem

Árton P. Dorneles [a], Olinto C.B. de Araújo [b], Luciana S. Buriol [a,*]

[a] Instituto de Informática, Universidade Federal do Rio Grande do Sul, 91501-970 Porto Alegre, RS, Brazil
[b] Colégio Técnico Industrial de Santa Maria, Universidade Federal de Santa Maria, 97105-900 Santa Maria, RS, Brazil

ARTICLE INFO

Available online 5 July 2014

Keywords:
High school timetabling
Mixed integer linear programming
Fix-and-optimize
Variable neighborhood descent
Matheuristics
ITC-2011

ABSTRACT

The high school timetabling is a classical combinatorial optimization problem that takes a large number of variables and constraints into account. Due to its combinatorial nature, solving medium and large instances to optimality is a challenging task. When resources are tight, it is often difficult to find even a feasible solution. Among the different requirements that are considered in Brazilian schools, two compactness requirements must be met on a teacher's schedule: the minimization of working days and the avoidance of idle timeslots. In this paper, we present a mixed integer linear programming model and a fix-and-optimize heuristic combined with a variable neighborhood descent method. Our method uses three different types of decompositions - class, teacher and day – in order to solve the high school timetabling problem. The method is able to find new best known solutions for seven instances, including three optimal ones. A comparison with results reported in the literature shows that the proposed fix-and-optimize heuristic outperforms state-of-the-art techniques for the resolution of the problem at hand.

1. Introduction

A common task to all educational institutions is the provision of a class assignment that combines teachers, students, rooms and timeslots (periods) that, in practice, constitute a solution to the high school timetabling problem. High quality solutions are relevant for financial and pedagogical reasons. In addition to feasibility, such a solution must be improved as much as possible, satisfying requirements and constraints of different nature. The quality of a solution to the high school timetabling problem leads to better use of resources and economic gains, thus directly influencing the quality of teaching and learning, as well as working conditions for teachers.

In spite of it's relevance, the problem is often manually solved in small institutions. However, the automation of this task is becoming common, and nowadays it is widely adopted (or even mandatory) in large institutions.

Timetabling requirements are separated into *hard* and *soft* ones. By hard requirements we mean those that must be satisfied, while soft requirements are those that may be violated, but should be satisfied whenever possible. Soft requirements have different levels of importance and are oftentimes conflicting with each other. Thus, it may be impossible to satisfy all of them. Typically, the quality of a solution is associated directly to the satisfaction of soft requirements. The more soft requirements are satisfied, the better a solution is considered.

The timetabling problem first appeared in the scientific literature in the 1960's [1] and since then it has been subject of intense research. The most basic problem is to schedule a set of class-teacher events (or meetings) such that no teacher (nor class) is assigned to more than one lesson at any timepoint. This basic problem can be solved in polynomial time by a min-cost network flow algorithm [2]. However, in a real-world application, teachers can be unavailable in some periods. Therefore, when this constraint takes place, the resulting timetabling problem is NP-complete [3].

In fact, this problem has several NP-complete variants proposed in the literature and the set of objectives and requirements depends mostly on the context of the application, the school and the place where it is located [4–6]. For instance, in certain countries, including Brazil, it is common that a teacher works in more than one school, holding several jobs. In order to allow for this possibility, it is important to compact the lessons in each school to the minimal number of days. Furthermore, the solution also requires the reduction or elimination of *idle periods* between lessons in a teacher's schedule in compliance with pedagogical demands or personal preferences. Teachers can also request that his/her lessons be taught in two consecutive periods (*double lessons*). This set of requirements

* Corresponding author. Tel.: +55 51 3308 6827; fax: +55 51 3308 7308.
E-mail addresses: arton.dorneles@inf.ufrgs.br (Á.P. Dorneles), olinto@ctism.ufsm.br (O.C.B. de Araújo), buriol@inf.ufrgs.br (L.S. Buriol).

http://dx.doi.org/10.1016/j.cor.2014.06.023
0305-0548/

define a problem called Class-Teacher Timetabling Problem with Compactness Requirements (CTTPCR).

This work extends previous studies about the CTTPCR. We propose a fix-and-optimize heuristic combined with a variable neighborhood descent (VND) method [7] using three different types of decompositions (class, teacher and day). We carry out extensive experiments to support the conclusions about the performance of our method in finding high quality solutions.

The remainder of the paper is organized as follows. Section 2 presents a review on the timetabling problem resolution. Section 3 presents the problem investigated in this study, the notation used to represent it, and a mixed integer linear programming formulation. Section 4 presents the proposed method for solving the problem. Section 5 presents experimental results considering both synthetic and real-world instances. Finally, Section 6 concludes the paper.

2. Related work

The timetabling problem has been intensively investigated since the 1960s [1]. We refer the reader to the excellent survey by Schaerf [6], which presents the main variants of the timetabling problem, its formulations and solution approaches.

For a few decades variants of the timetabling problem were considered intractable by exact methods, since only small instances, or simplified variants of the problem, were shown to be solved in reasonable time. However, in the last years several improvements have been made in mixed integer linear programming (MIP) solvers [8], which motived new research studies using this approach [9].

The CTTPCR problem is originated from the Brazilian High School System and it was first defined by Souza and Maculan [10]. In Souza [11] a MIP formulation for CTTPCR was presented, as well as a set of instances, which became a basic testbed used until today. Recently, Santos et al. [12] proposed an extended formulation and applied a column generation algorithm, providing the best known lower bounds for the testbed CTTPCR instances.

In a previous work [13], we empirically showed that the minimization of the teachers' idle periods turns the resolution of the problem considerably more difficult through general purpose solvers. In addition, in that work we proposed a new MIP model by reformulating the idle periods requirement previously proposed by Santos et al. [12]. Even though the reformulation allows one to solve slightly larger instances of CTTPCR, it is still impracticable to solve large real-world instances from the testbed. In general, scheduling problems, which include the timetabling problem, are considered hard to solve directly via general purpose solvers [8].

One technique that is useful in this context is the use of matheuristics, which combine mathematical programming and heuristic methods [14]. Currently, matheuristics have been successfully applied to solve several optimization problems across a range of applications. Fix-and-optimize is a matheuristic that iteratively decomposes a problem into smaller subproblems. In each iteration of the algorithm, a decomposition process is applied aiming at fixing most of the decision variables at their value in the current solution. Since the resulting subproblem is composed only by a small group of "free" variables to be optimized, each subproblem can be solved fairly quickly by a MIP solver, when compared with the full model. The solution obtained in each iteration becomes the current solution when it improves the objective value. In further iterations of the algorithm, a different group of variables is selected to be optimized. This process is repeated until a termination condition is satisfied. The fix-and-optimize heuristic was proposed independently by Gintner et al. [15] and Pochet and Wolsey [16]. In the latter, the method was called *exchange*, designed to improve the relax-and-fix heuristic [17]. However, the name fix-and-optimize used by the former was adopted in the literature.

Regarding the educational timetabling problem, there are few studies in the literature exploring matheuristics. To the best of our knowledge, there is a limited number of publications related to course timetabling, including, e.g. [18,19]. Further, perhaps only one reference that makes use of matheuristics applied to high school timetabling has been published until now. In this work, Avella et al. [20] proposed a two-phase algorithm applied to a problem similar to CTTPCR. The first phase is a simulated annealing (SA) algorithm. The second phase consists in a large-scale neighborhood search that decomposes the problem into subproblems which are solved independently by a MIP solver. In each subproblem all teachers remain fixed, with the exception of a randomly chosen one.

3. Problem definition and modeling

The goal of the Class-Teacher Timetabling Problem with Compactness Requirements (CTTPCR) is to build a weekly timetable. The week is organized as a set of days D, and each day is split into a set of periods P. Let C be a set of classes and T a set of teachers. A class $c \in C$ is a group of students that follow the same course and have full availability. A *timeslot* is a pair, composed of a day and a class period (d,p), with $d \in D$ and $p \in P$, wherein all periods have the same duration. Teachers $t \in T$ may be unavailable in some timeslots.

The main input for the problem is a set of events E that should be scheduled. Typically, an event is a meeting between class and teacher to address a particular subject in a given number of lessons (*workload*) in a given room. Particularly in the Brazilian context, a class, a teacher and a room are pre-assigned to each event $e \in E$. In addition, each event defines how lessons are distributed over a week by requesting an amount of double lessons, restricting the daily limit of lessons, and defining whether lessons taught on the same day are consecutive or not.

A *feasible* timetable has a timeslot assigned to each lesson of events satisfying the hard requirements H1–H6 below:

H1 The workload defined in each event must be satisfied.
H2 A teacher cannot be scheduled to more than one lesson in a given period.
H3 Lessons cannot be taught to the same class in the same period.
H4 A teacher cannot be scheduled to a period in which she/he is unavailable.
H5 The maximum number of daily lessons of each event must be respected.
H6 Two lessons from the same event must be consecutive when scheduled for the same day, in case it is required by the event.

Besides feasibility regarding hard constraints, as many as possible of the soft requirements S1–S3 stated below should be satisfied:

S1 Avoid teachers' idle periods.
S2 Minimize the number of *working days* for teachers. In this context, working day means a day that the teacher has at least one lesson assigned to her/him.
S3 Provide the number of double lessons requested by each event.

3.1. Problem formulation

In this subsection we present a MIP formulation for the CTTPCR problem considering all the hard and soft requirements mentioned above. The notation used in the problem formulation is presented in Table 1.

Table 1
Notation used for the CTTPCR model.

Symbol	Definition
Sets	
$d \in D$	days of week
$p \in P$	periods of day
P'	P without the last two periods of a day
$t \in T$	set of teachers
$c \in C$	set of classes
$e \in E$	set of events
E_t	set of events assigned to teacher t
E_c	set of events assigned to class c
U	set of tuples (m,n) for $m \in P', n \in P : n \geq m+2$
Q	set of tuples (m,n) for $m \in P', n \in P : n \geq m$
SG_e	set of timeslots on which event e can start a double lesson ($SG_e = \{(d,p) : d \in D, p \in P$ and $p < \|P\|, V_{edp} + V_{ed,p+1} = 2\}$). The parameter V_{edp} is defined below
Parameters	
ω_t	cost of each idle period of teacher t
γ_t	cost of each working day of teacher t
δ_e	cost of each double lesson of event e not taught sequentially
R_e	workload of event e
L_e	maximum daily number of lessons of event e
V_{edp}	binary parameter that indicates whether the teacher assigned to event e is available in the timeslot (d,p)
MG_e	minimum amount of double lessons required by event e
Variables	
x_{edp}	binary variable that indicates whether event e is scheduled to timeslot (d,p)
y_{td}	binary variable that indicates whether at least one lesson is assigned to teacher t on day d
g_{edp}	binary variable that indicates whether event e has a double lesson starting at timeslot (d,p)
G_e	integer variable that indicates the number of double lessons remaining to reach MG_e
b_{edp}	binary variable that indicates whether event e has a lesson at timeslot (d,p) and not at timeslot $(d,p-1)$
z_{tdmn}	binary variable that indicates whether the teacher t has idle periods on day d between periods m and n

Our formulation is novel in two aspects. Firstly, it includes the hard requirement H6, which had not been considered in previous studies on CTTPCR. Secondly, following Dorneles et al. [13] we analysed results considering different sets of constraints to model the requirement S1. We showed that this soft requirement considerably affects the solution quality; thus we proposed a new formulation that attends this requirement and that is faster when considering the testbed instances for this problem.

$$\textbf{Min} \quad \sum_{t \in T} \sum_{d \in D} \sum_{(m,n) \in U} \omega_t (n-m-1) z_{tdmn} + \sum_{t \in T} \sum_{d \in D} \gamma_t y_{td} + \sum_{e \in E} \delta_e G_e \qquad (1)$$

Subject to

$$\sum_{d \in D, p \in P} x_{edp} = R_e \quad \forall e \qquad (2)$$

$$\sum_{p \in P} x_{edp} \leq L_e \quad \forall e, d \qquad (3)$$

$$x_{edp} \leq V_{edp} \quad \forall e, d, p \qquad (4)$$

$$\sum_{e \in E_t} x_{edp} \leq y_{td} \quad \forall t, d, p \qquad (5)$$

$$\sum_{e \in E_t} \sum_{p \in P} x_{edp} \geq y_{td} \quad \forall t, d \qquad (6)$$

$$\sum_{e \in E_c} x_{edp} \leq 1 \quad \forall c, d, p \qquad (7)$$

$$b_{edp} \geq x_{edp} - x_{edp-1} \quad \forall e, d, p : p > 1 \qquad (8)$$

$$\sum_{p \in P : p > 1} b_{edp} + x_{ed1} \leq 1 \quad \forall e, d \qquad (9)$$

$$g_{edp} \leq x_{edp} \quad \forall e, (d,p) \in SG_e \qquad (10)$$

$$g_{edp} \leq x_{edp+1} \quad \forall e, (d,p) \in SG_e \qquad (11)$$

$$G_e \geq MG_e - \sum_{(d,p) \in SG_e} g_{edp} \quad \forall e \qquad (12)$$

$$\sum_{d \in D} y_{td} \geq \max\left\{ \left\lceil \frac{\sum_{e \in E_t} R_e}{|P|} \right\rceil, \max_{e \in E_t} \left\{ \left\lceil \frac{R_e}{L_e} \right\rceil \right\} \right\} \quad \forall t \qquad (13)$$

$$\sum_{(m,n) \in Q} z_{tdmn} = y_{td} \quad \forall t, d, m \in P : m \leq 3 \qquad (14)$$

$$\sum_{(m,n) \in Q} z_{tdmn} \leq y_{td} \quad \forall t, d, n \in P : n \geq 3 \qquad (15)$$

$$z_{tdpp} \leq 1 + \sum_{e \in E_t} (x_{edp+1} - x_{edp}) \quad \forall t, d, p \in P' \qquad (16)$$

$$z_{tdmm+1} \leq 1 - \sum_{e \in E_t} x_{edn} \quad \forall t, d, (m,n) \in U \qquad (17)$$

$$z_{tdmn} \leq \sum_{e \in E_t} x_{edn} \quad \forall t, d, (m,n) \in U \qquad (18)$$

$$x_{edp}, b_{edp} \in \{0,1\}, g_{edp}, G_e \geq 0 \quad \forall e, d, p \qquad (19)$$

$$y_{td} \geq 0, z_{tdmn} \in \{0,1\} \quad \forall t, d, (m,n) \in Q \qquad (20)$$

The objective function of the problem formulation consists of three weighted parts related to the soft requirements S1, S2 and S3, respectively. Regarding the soft requirement S1, the penalization is proportional to the number of idle periods.

Constraint set (2) ensures that the workload of each event is fully scheduled. Constraint set (3) provides a daily limit of lessons for each event. Constraint set (4) ensures that the lessons of an event are scheduled in available periods. Constraint sets (5) and (7) ensure that teacher and class are scheduled to only one lesson at a time, respectively. Constraint sets (5) and (6) identify the working days of teachers. Constraint sets (8) and (9) ensure that the lessons of an event are scheduled sequentially according to requirement H6. Constraint sets (10) and (11) enforce double lessons when the variable g_{edp} is equal to one. Constraint set (12) determines G_e, the number of double lessons remaining to

reach MG_e. Since G_e accounts for the objective function, the sum in the right side of the inequality tends to increase, and thus the establishment of double lessons is promoted.

Constraint set (13) is a cut proposed by Souza [11] that defines a minimum number of working days for each teacher and makes the formulation stronger.

Constraint sets (14)–(18) determine the number of idle periods in a solution. To explain these constraints, it is useful to consider an *idle periods graph* as shown in Fig. 1. In this graph the vertices p_1, p_2, p_3, p_4 and p_5 are periods on a day d of a teacher t. There are two types of arcs: *idle period arcs*, with $(m,n) \in U$ that are penalized in the objective function, and *auxiliary arcs*, with $(m,n) \in Q \backslash U$. In Fig. 1 the idle period arcs are drawn as solid lines and the auxiliary arcs are drawn as dashed lines. Note that each arc corresponds directly to a binary variable z_{tdmn} such that m is the tail and n is the head node of the arc. For example, variable z_{td13} corresponds to the arc (p_1, p_3). The underlying idea is to ensure that the idle period arcs are properly activated to compute the cost of idle periods. Constraint sets (14) and (15) ensure that there is exactly one arc leaving and reaching each period that can be the beginning or end of an idle period, respectively. Constraint set (16) states that an auxiliary arc (m,m) must be active only in two situations: when the teacher has no lesson in the period m, or when the teacher has a lesson in periods m and $m+1$. Constraint set (17) states that an auxiliary arc $(m, m+1)$ must be active only when the last lesson on a working day of the teacher occurs at period m. Constraint set (18) states that an idle period arc (m,n) can be active only when the teacher has a lesson in period n.

Fig. 2 presents only the activated arcs in the idle periods graph considering two allocation scenarios for a teacher. In the scenario (a) there are two idle periods, p_2 and p_3, that are identified by the arc (p_1, p_4). In the scenario (b) there are no idle periods. Thus, only auxiliary arcs are activated.

4. A fix-and-optimize heuristic combined with a variable neighborhood descent strategy

The model presented in the previous section can be used to solve small instances of CTTPCR by using general purpose MIP solvers. Thus, specialized solution approaches are required for solving medium and large problem instances. In the proposed CTTPCR model, the set of variables x_{edp} is the most important one,

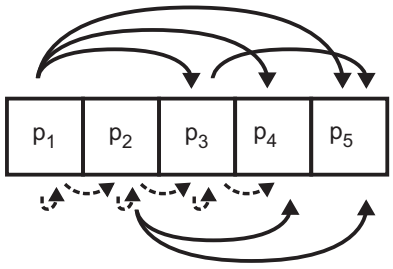

Fig. 1. Idle periods graph.

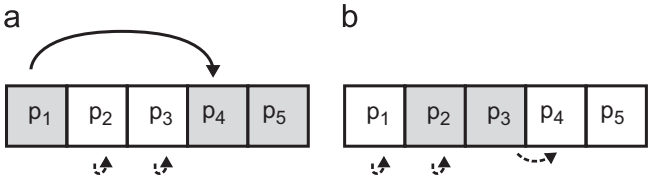

Fig. 2. Examples of allocation scenarios. The gray cells indicate periods in which a lesson occurs.

since other decision variables depend on this set. This means that if the values of x_{edp} variables are fixed, then the values of the remaining ones are easily inferred. This property indicates that the fix-and-optimize heuristic could succeed in solving the problem, since fixing binary variables to integer values would result in only two possibilities.

In the fix-and-optimize heuristic, the way we choose the variables to be fixed, as well as the number of variables to be fixed, directly impacts on the performance of the algorithm and on the quality of the final solution, i.e., on the level of satisfaction of the soft constraints. Thus, the decomposition operation must vary in *type* and *size*. In this paper, we propose three types of decomposition:

- *Class Decomposition* (CD): a certain number of classes are free to be optimized.
- *Teacher Decomposition* (TD): a certain number of teachers are free to be optimized.
- *Day Decomposition* (DD): a certain number of days are free to be optimized.

Note that when a class, teacher or day is free to be optimized it means that all variables for the events of this class, teacher or day are not fixed.

For each type of decomposition τ we can define a parameter k which defines the cardinality of the subset of variables that are free to be optimized. For instance, considering the decomposition CD, we can free 1,2,3,..., k classes, such that $k \leq |C|$.

The combination of a decomposition type $\tau \in \{\text{CD}, \text{TD}, \text{DD}\}$ and a size k defines different neighborhoods. The tuple (τ,k) can be used to represent a specific neighborhood. For instance, a neighborhood (DD, 2) of a solution x consists of all solutions that can be obtained by solving subproblems such that $|D|-2$ days are fixed exactly as in x, but 2 days are free to be optimized.

Since there are many possible neighborhoods, we explore them through a variable neighborhood descent (VND) approach [7]. The VND process implies an iteration over a sequence of neighborhoods $\mathcal{N}$ while better solutions are found using a *first improvement* selection strategy. Typically the neighborhoods, $(\tau, k) \in \mathcal{N}$, are explored by a general purpose MIP solver from the smaller to the larger ones.

Fig. 3 presents the behavior of the fix-and-optimize heuristic applied on a toy instance of the problem. It is composed by three classes (c_1, c_2, c_3), six teachers, four periods (p_1, p_2, p_3, p_4) and only one day. The algorithm begins from a feasible solution and, at each step, solves a different subproblem. Note that the procedure begins with the neighborhood (CD, 1) and improves the current solution twice. At iteration 6, the neighborhood is changed to (CD, 2) and the algorithm is then able to improve the solution once again. In the following sections we present further details of this procedure.

4.1. Generating initial feasible solutions

Since the variable fixations described above are based on values of a previous solution, we need to provide an initial feasible solution to start the fix-and-optimize algorithm. In order to do so, we solve a feasibility version of the CTTPCR by disregarding the objective function of the MIP model presented in Section 3, i.e., all soft constraints of the problem. This approach allows the algorithm to quickly find an initial feasible solution.

4.2. The proposed algorithm

The overall algorithm is described in the pseudo-code of Fig. 4. Function `fixAndOptimize()` receives as input a sequence of

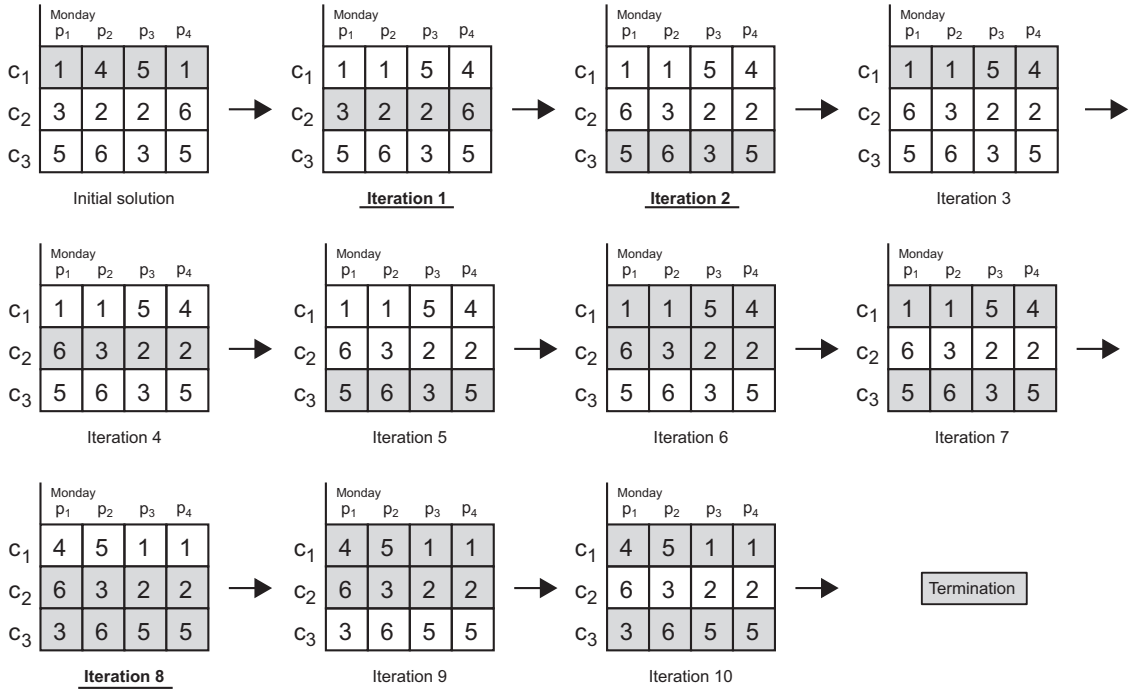

Fig. 3. Example of the proposed fix-and-optimize heuristic using $\mathcal{N}=((\mathrm{CD},1),(\mathrm{CD},2))$. Each table shows an iteration of the algorithm, and each cell shows a teacher assignment. The shaded cells denote the group of variables that are free to be optimized, while the remaining ones are fixed with values from the previous solution. Underlined iterations denote that the current solution was improved in the previous iteration. Note that teachers 1, 3, 5 and 6 have idle periods in the initial solution which are gradually removed along the iterations.

Algorithm `fixAndOptimize` $(\mathcal{N}, \mathrm{TL}, \mathrm{STL})$

1: $x^* \leftarrow$ `GenerateInitialSolution()`;
2: **if** $x^* = \emptyset$ **then**
3: **return** $\emptyset$;
4: **end if**
5: **for all** $(\tau,k) \in \mathcal{N}$ **do**
6: `count` $\leftarrow$ `subproblemCount`(τ,k);
7: `s` $\leftarrow 1$;
8: `noImprov` $\leftarrow 0$;
9: **repeat**
10: $\mathcal{R} \leftarrow$ `decompose`(τ,k,s);
11: $x \leftarrow$ `solve`$(x^*,\mathcal{R},\mathrm{STL})$;
12: **if** x is better than x^* **then**
13: $x^* \leftarrow x$;
14: `noImprov` $\leftarrow 0$;
15: **else**
16: `noImprov++`;
17: **end if**
18: **if** TL was reached **then**
19: **return** x^*;
20: **end if**
21: `s` $\leftarrow$ (`s` **mod** `count`)+1;
22: **until** `noImprov=count`;
23: **end for**
24: **return** x^*.

Fig. 4. Pseudo-code of the proposed fix-and-optimize heuristic.

Algorithm `subproblemCount` (τ,k)

1: **switch**(τ)
2: **case** CD
3: `count` $\leftarrow \binom{|C|}{k}$;
4: **case** TD
5: `count` $\leftarrow \binom{|T|}{k}$;
6: **case** DD
7: `count` $\leftarrow \binom{|D|}{k}$;
8: **end switch**
9: **return** `count`.

Fig. 5. Function that computes the number of subproblems of a neighborhood.

Algorithm `decompose` (τ,k,s)

1: **switch**(τ)
2: **case** CD
3: $\mathcal{R} \leftarrow \{x_{edp} : c \in$ `subsets`$(C,k,s), e \in E_c, d \in D, p \in P\}$;
4: **case** TD
5: $\mathcal{R} \leftarrow \{x_{edp} : t \in$ `subsets`$(T,k,s), e \in E_t, d \in D, p \in P\}$;
6: **case** DD
7: $\mathcal{R} \leftarrow \{x_{edp} : e \in E, d \in$ `subsets`$(D,k,s), p \in P\}$;
8: **end switch**
9: **return** $\mathcal{R}$.

Fig. 6. Decomposition function.

neighborhoods ($\mathcal{N}$), the overall time limit (`TL`), and the time limit for each subproblem (`STL`).

The algorithm begins by creating an initial feasible solution x^* (line 1) as described in Section 4.1. If the problem is infeasible it terminates returning no solution.

The outer loop (lines 5–23) iterates over a sequence of neighborhood structures $\mathcal{N}$ on the same fashion as a VND algorithm. Each neighborhood has a finite number of subproblems computed by function `countSubproblems()` (line 6) as described in the pseudo-code of Fig. 5. The number of subproblems, indicated by `s`, depends on the type of the decomposition τ and on its size k.

In the inner loop (lines 9–22) the subproblems of the current neighborhood are explored until `noImprov=count`, i.e., the algorithm evaluates each subproblem (within the subproblem time limit STL) of the neighborhood (τ,k) and is not able to improve the quality of the current solution, i.e., the level of satisfaction of the soft constraints.

The function `decompose()` (line 10) is used to compute the set of variables to be optimized ($\mathcal{R}$) in the current subproblem according to the pseudo-code presented in Fig. 6. The function

`subsets`(S,k,s) returns in lexicographical order the sth subset of all subsets of S containing exactly k elements.

After that, the subproblem is solved through function `solve()` which receives three parameters: the current solution (x^*), the set of variables to be optimized ($\mathcal{R}$), and the time limit of the subproblem (`STL`). This function fixes all variables x_{edp} which do not belong to $\mathcal{R}$ to their values in x^*, and starts the solver. If it is able to find a better solution than x^*, then it is returned. Otherwise, if no better solution is found within the time limit, or if the subproblem is infeasible, it returns the previous current solution x^*. Note that subproblems can be infeasible since we add a cutoff constraint that forces the solver to search only for solutions whose objective value is less than the objective value of x^*. After the function `solve()` returns a result, all variables previously fixed are released.

Whenever a better solution is found it becomes the current solution x^* (line 13), and variable `noImprov` is reset. Otherwise, `noImprov` is incremented. The algorithm terminates returning the best solution found when the time limit `TL` is reached (lines 18–20). In line 21 the variable `s` indexes the next subproblem. After all neighborhoods in the outer loop are explored, the algorithm terminates in line 23 returning the best visited solution x^*.

5. Computational experiments

In this section we present an experimental evaluation for the fix-and-optimize heuristic proposed in this paper. The goal of our experiments is to answer the following questions:

(i) Does the proposed algorithm outperform a general purpose MIP solver?
(ii) Which sequence of neighborhoods $\mathcal{N}$ provides the best results?
(iii) How do our results compare with results of the state-of-the-art methods for solving the problem?

The subproblems are solved by CPLEX 12.1 [21] with default settings and the algorithms were implemented in C++ using the compiler g++ 4.6.1. The experimental results were computed in a Desktop-PC equipped with an Intel Core i5-2300 processor clocked at 2.8 GHz, 4 GB of RAM, over a 64 bits Linux operating system. Along this section, we report results of one run for each tested algorithm, since they are deterministic. The mathematical model parameters γ_t, ω_t and δ_e were set to 9, 3 and 1, respectively. These values are the same used by all previous works on CTTPCR of our knowledge.

5.1. Dataset

To evaluate the algorithm, we used the instances presented in Table 2. In the table, the first two columns present the instance identifier name. Since their names are long, we use the identifiers for shortening reference along the text. Columns $|D|$ and $|P|$ show the number of days and periods, respectively, while columns $|T|$, $|C|$ and $|E|$ present the number of teachers, classes and events, respectively. Finally, columns $\sum_{e\in E} MG_e$ and $\sum_{e\in E} R_e$ present the total number of required double lessons and the total amount of workload, respectively.

5.2. Initial solutions

The instances are split into two sets. Instances 1–7 comprise set-1 and are available from the repository [22] and to the best of our knowledge they were used in all previous works on CTTPCR. Requirement H6 is not considered in this group of instances. Instances A, D, E, F, G, from set-2, are different versions of instances 1, 4, 5, 6, 7, respectively. They differ mainly in two aspects: in set-2, teachers are available in all periods, and requirement H6 is considered. These modifications made the instances of set-2 more challenging to be included in the first round of the Third International Timetabling Competition 2011 (ITC-2011) [23]. They are part of the XHSTT-2012 archive.[1]

Table 2
Main characteristics of the tested instances.

Id	Name	$\|D\|$	$\|P\|$	$\|T\|$	$\|C\|$	$\|E\|$	$\sum_{e\in E} MG_e$	$\sum_{e\in E} R_e$
1	Inst1	5	5	8	3	21	21	75
2	Inst2	5	5	14	6	63	29	150
3	Inst3	5	5	16	8	69	4	200
4	Inst4	5	5	23	12	127	66	300
5	Inst5	5	5	31	13	119	71	325
6	Inst6	5	5	30	14	140	63	350
7	Inst7	5	5	33	20	205	84	500
A	BrazilInstance1	5	5	8	3	21	21	75
D	BrazilInstance4	5	5	23	12	127	66	300
E	BrazilInstance5	5	5	31	13	119	71	325
F	BrazilInstance6	5	5	30	14	140	63	350
G	BrazilInstance7	5	5	33	20	205	84	500

Table 3
Initial feasible solutions.

Id	LB	obj	gap_L (%)	Time (s)
1	202	612	202.97	0.0
2	333	1089	227.03	0.2
3	423	1172	177.07	0.4
4	652	1598	145.09	0.9
5	762	2369	210.89	0.9
6	756	2299	204.10	1.1
7	1017	2758	171.19	2.9
A	189	567	200.00	0.2
D	621	1658	166.99	10.6
E	756	2109	178.97	10.8
F	738	2247	204.47	12.5
G	999	2776	177.88	31.6
Avg.	620.7	1771.2	188.89	6.0

Table 3 shows the initial feasible solution values obtained by CPLEX according to the procedure described in Section 4.1. Column *LB* presents the best known lower bounds computed for the instances. Lower bounds for instances 1–7 were provided by Santos et al. [12], while the remaining lower bounds were obtained by solving the linear relaxation of the model presented in Section 3.1. Column *obj* shows the value of the objective function. Column gap_L presents the percentage deviation from the best known lower bound (LB). It is computed by $100\times(ob-LB)/LB$. Column *time* shows the running times in seconds.

As one can notice, the proposed procedure provides feasible solutions in a short time, spending just 6 s on average. Despite their values begin far from the lower bound *LB*, it is important to note that the method provides feasible solutions quickly without burdening the overall total time. As expected, the generation of initial solutions for instances of set-2 is more time demanding, since the solution space is larger when all teachers have full availability.

5.3. Parameter setting

In this section we describe a set of experiments that supported us to define a standard parameter setting to be used by the

[1] http://www.utwente.nl/ctit/hstt/archives/XHSTT-2012/.

proposed heuristic. Basically, we aimed to define the sequence of neighborhoods $\mathcal{N}$, the order in which the different neighborhoods are visited, and a suitable subproblem time limit (STL).

Initially, we tested several neighborhoods composed by a single tuple (τ,k), i.e., $|\mathcal{N}|=1$, with $\tau \in \{CD, TD, DD\}$ and with several values for the decomposition size k. For the neighborhoods involving day decompositions we tested $k \in \{1,\ldots,5\}$ and for teacher and classes decompositions we used $k \in \{1,\ldots,12\}$. Obviously, the maximum value for k is set to five for the decomposition DD since all tested instances have $|D|=5$. Considering the decompositions CD and TD the maximum value for k was chosen in order to keep a good trade-off between performance and solution quality. These neighborhoods are combined with two different STL values: STL=30 and STL$=\infty$, meaning that the subproblem time limit is 30 s in the first case, and when set to ∞ the subproblem runs to optimality, or the overall time limit (TL) is reached.

Fig. 7 shows average results, considering all instances, for the gap (plot in the top), and running times (plot in the bottom) of the different combinations of neighborhoods and STL values. The gap value of each instance is calculated as the percentage deviation of the solution value found to the best known values. For each combination, the overall time limit (TL) of each run was set to 10 min.

Analyzing this figure, it can be observed that class and teacher decompositions provide better results than day decomposition. This occurs because the subproblems generated by day decompositions are too large and CPLEX spends a long time on each subproblem, leading to few solution improvements.

Another observation is related to the STL parameter. All runs with STL=30 spent less time than STL$=\infty$. A suitable setting of this parameter is fundamental for the overall heuristic performance. If the time available is too short, the solver rarely solves the subproblem. This can be observed in Fig. 7 in cases with STL=30 in class decomposition with $k \geq 8$, and for day decomposition with $k \geq 4$. This behavior is expected since the number of free variables increases with k. In these cases the algorithm often finishes prematurely. Finally, runs with teacher decomposition were almost not affected by STL since most of the subproblems were solved within the subproblem time limit.

Unexpectedly, in some situations in which the algorithm finished before the overall time limit, results using STL=30 were better than with STL$=\infty$ (for instance, for class decomposition $k \leq 7$). In this case, when STL$=\infty$ the algorithm performs large decreasing steps in the objective function value during the first iterations. As a side effect some parts of the solution are "frozen" in its local optima discouraging further interactions with other solution parts.

In summary, considering the overall results for single neighborhoods, we concluded that class and teacher decompositions with small size values (ranging from 1 to 4) might provide a good trade-off between running time and solution quality. Thus it is reasonable to consider them as strong candidates to take place the first positions in a sequence of neighborhoods for the VND method.

In order to answer which sequence of neighborhoods $\mathcal{N}$ provides the best results, we evaluated different sequences

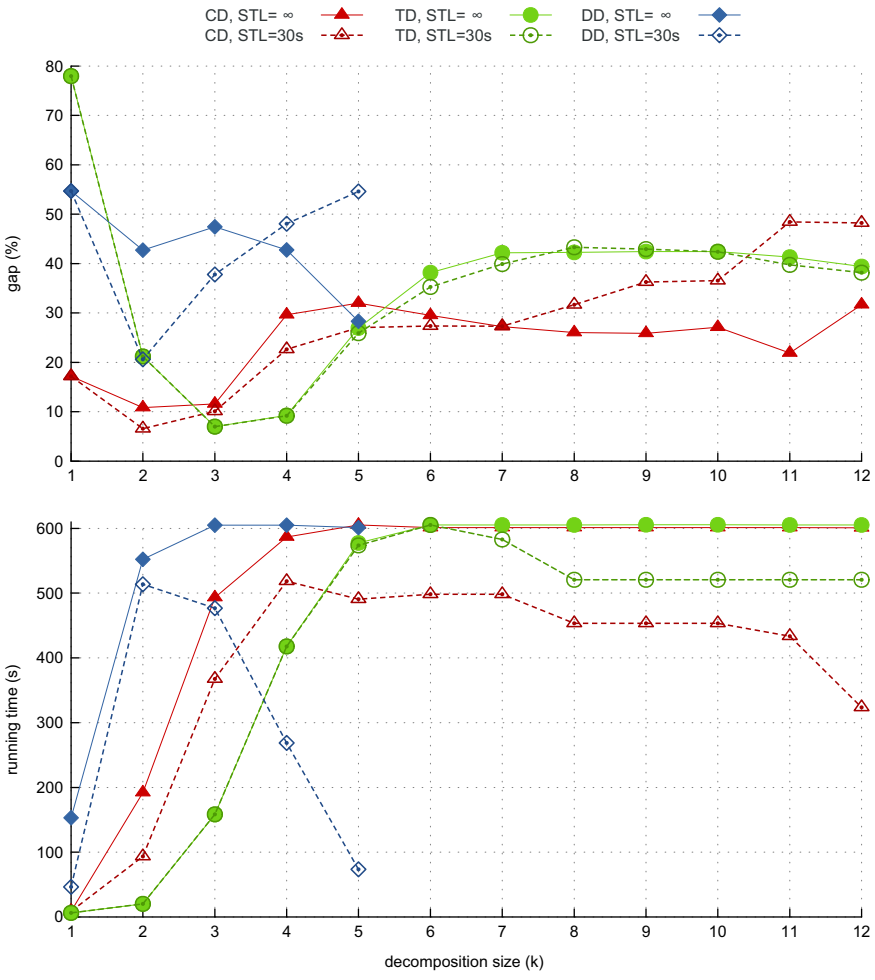

Fig. 7. Each line in the plots represents a combination of decomposition type and STL value. The solid lines show results from runs with STL$=\infty$, while dashed lines show results from runs for STL=30. The x-axis represents the size (k) of each evaluated decomposition.

Table 4
Results for variants of the fix-and-optimize heuristic.

Var	Sequence of neighborhoods ($\mathcal{N}$)	STL=10s		STL=30s		STL=50s	
		$\overline{gap_L}$ (%)	#opt	$\overline{gap_L}$ (%)	#opt	$\overline{gap_L}$ (%)	#opt
F1	$((TD,1),\ldots,(TD,\infty))$	3.91	1	3.82	2	3.85	2
F2	$((CD,1),\ldots,(CD,\infty))$	3.36	2	3.32	2	3.22	2
F3	$((TD,2),\ldots,(TD,\infty))$	3.95	1	3.86	2	3.86	2
F4	$((CD,2),\ldots,(CD,\infty))$	3.36	2	3.24	2	3.42	2
F5	$((TD,1),(CD,1),\ldots,(TD,\infty),(CD,\infty))$	3.04	3	3.19	3	3.19	3
F6	$((CD,1),(TD,1),\ldots,(CD,\infty),(TD,\infty))$	3.04	3	3.13	3	3.23	3
F7	$((TD,2),(CD,2),\ldots,(TD,\infty),(CD,\infty))$	2.90	3	2.84	3	3.11	3
F8	$((CD,2),(TD,2),\ldots,(CD,\infty),(TD,\infty))$	3.02	4	2.92	4	3.17	3
F9	$((TD,3),(CD,3),\ldots,(TD,\infty),(CD,\infty))$	3.42	2	3.68	2	3.61	2
F10	$((CD,3),(TD,3),\ldots,(CD,\infty),(TD,\infty))$	6.02	3	7.34	3	8.29	3

presented in Table 4, producing different variants (*Var*) of the algorithm (F1–F10). Each sequence is composed by different arrangements of teacher and class decompositions where neighborhoods with small values of k appear first since, as we concluded previously, they produce high quality solutions in a short time. All variants were tested considering three different values of parameter STL $= \{10, 30, 50\}$. For each test we report the average gap ($\overline{gap_L}$) and the number of optimal solutions found (*#opt*). The overall time limit (TL) of each run was set to 10 min.

According to the table, we observed that the algorithm was not significantly sensitive to changes in the parameter STL. The best results were obtained by variants F5–F8, which results are quite similar. While the variant F7 achieved the best average gap among them, the variant F8 found the largest number of optimal solutions. On the whole, the results indicated that variants mixing different types of decompositions (F5–F10) are better than variants with just one type of decomposition (F1–F4), except for variants F9 and F10. Particularly, these two performed poorly since the time limit imposed was too short to deal properly with sequences of neighborhoods starting with large decompositions ($k \geq 3$). In fact, according to our experience, when more time is available, in average, variants F5–F10 are strictly better than variants F1–F4. We chose the variant F8 using STL=30 as the standard setting used in the next experiments performed in this work.

5.4. Comparison with CPLEX

In Table 5 the results obtained by variant F8 of the proposed fix-and-optimize heuristic are compared with the results obtained by the general purpose solver CPLEX (CPX). The labels CPX and F8 are subscripted with the overall time limit used in the method. Column *BKV* shows the previous best known solution values. Whereas the values for instances 1, 2, 3, and 6 were obtained by Santos et al. [12], the values for instances 4, 5, and 7 were obtained by Dorneles et al. [13]. Finally, results for set-2 were the best generated solution reported in the first round of ITC-2011. By the competition rules, the best known solution could be obtained by any technique, using any resources, without any time limit. The teams had 5 months to produce these results. For each method we report the objective value (*obj*) and the percentage deviation (gap_B) from the best known value (*BKV*). Column gap_B is computed by $100 \times (obj - BKV)/\min(BKV, obj)$. Thus, a negative gap_B value represents an improvement over the best known solution value.

As the table shows, the proposed algorithm was able to find better solutions than CPLEX spending considerably less computational time. For example, $F8_{(10m)}$ found, on average, better or equal results than $CPX_{(10h)}$ for all instances, except for instance 3. In a separate experiment, the variant $F8_{(6h)}$ was also able to find the optimal result for this instance.

We can also observe that if more time is available, the proposed algorithm can improve the quality of the solutions even further. For set-1, for example, variant $F8_{(10m)}$ was able to obtain the best known value for three instances, and two new best known values (instances 5 and 7) were found. Variant $F8_{(30m)}$ found a new best known value for instance 6, and improved the previous best known value of instance 7. Variant $F8_{(1h)}$ improved the solution of instance 3. Regarding instances of set-2, considerable improvements are observed when comparing the proposed algorithm with the best known values and with CPLEX results. Both, our method and CPLEX, found the best known solution for instance A. For the remaining instances, $CPX_{(10h)}$ was unable to find the best known solution for any of them, and the gap ranges from 3.07% up to 80.64%. On the other hand, variant $F8_{(10m)}$ was able to improve the best known solutions for all of them. New improvements are obtained for variants $F8_{(30\ m)}$ and $F8_{(1h)}$.

In fact, along the development of the fix-and-optimize algorithm, several new best results were found while testing different configurations of the algorithm. The next section presents the new best known results for the instances.

5.5. New best known results

Table 6 presents the best results produced in this study.[2] Results shown in boldface are new best known values. Columns *LB* and *BKV* were previously presented. Column *obj* presents the objective value obtained by the proposed fix-and-optimize heuristic. Columns BKV_{itc} and obj_{itc} present the solution evaluation provided by HSEVal validator.[3] HSEVal checks the solution feasibility and also computes the solution value. It was used to evaluate the solutions of ITC-2011. Note that $obj_{itc} = obj - LB$, as well as $BKV_{itc} = BKV - LB$. Some cells are filled with "-" since instances 1–7 were not tested in ITC-2011. Columns gap_L and gap_B are computed as mentioned, respectively, in Sections 5.2 and 5.4. Results whose gap_L value is zero represent an optimal solution. Finally, column *Variant* presents which variant of fix-and-optimize heuristic produced the best result reported for each instance in column *obj*.

The solutions achieved by our approach are equal or better in quality when compared to best known results reported in the literature. Our method was able to find seven new best values out of the 12 instances analyzed. In addition to new optimal values achieved for instances 5, 6, and 7 we found the optimal results for all instances in set-1. Moreover, our algorithm was able to improve all solutions for CTTPCR instances of the first round of the Third

[2] The solution for each instance whose values are reported in Table 6 is available at www.inf.ufrgs.br/~apdorneles/timetabling/2013HSFOPTVND.

[3] http://sydney.edu.au/engineering/it/~jeff/hseval.cgi

Table 5
Comparison results between CPLEX and the proposed fix-and-optimize heuristic.

Id	BKV	$CPX_{(1h)}$		$CPX_{(10h)}$		$F8_{(10m)}$		$F8_{(30m)}$		$F8_{(1h)}$	
		obj	gap_B (%)	obj	gap_B (%)	obj	gap_B (%)	obj	gap_B (%)	obj	gap_B (%)
1	202	202	0.00	202	0.00	202	0.00	202	0.00	202	0.00
2	333	333	0.00	333	0.00	333	0.00	333	0.00	333	0.00
3	423	426	0.71	423	0.00	429	1.42	429	1.42	426	0.71
4	652	652	0.00	652	0.00	652	0.00	652	0.00	652	0.00
5	764	801	4.84	764	0.00	762	−0.26	762	−0.26	762	−0.26
6	760	778	2.37	765	0.66	761	0.13	759	−0.13	759	−0.13
7	1028	1259	22.47	1028	0.00	1019	−0.88	1017	−1.08	1017	−1.08
A	200	200	0.00	200	0.00	200	0.00	200	0.00	200	0.00
D	665	873	31.28	737	10.83	653	−1.84	648	−2.62	648	−2.62
E	799	1017	27.28	840	5.13	795	−0.50	784	−1.91	778	−2.70
F	815	1520	86.50	840	3.07	796	−2.39	787	−3.56	787	−3.56
G	1121	2025	80.64	2025	80.64	1087	−3.13	1085	−3.32	1084	−3.41
Avg.	646.8	840.5	21.34	734.1	8.36	640.8	−0.62	638.2	−0.96	637.3	−1.09

Table 6
New best known results.

Id	LB	BKV	obj	BKV_{itc}	obj_{itc}	gap_L (%)	gap_B (%)	Variant
1	202	202	202	–	–	0.00	0.00	$F8_{(10m)}$
2	333	333	333	–	–	0.00	0.00	$F8_{(10m)}$
3	423	423	423	–	–	0.00	0.00	$F10_{(1h)}$
4	652	652	652	–	–	0.00	0.00	$F8_{(10m)}$
5	762	764	**762**	–	–	**0.00**	−0.26	$F8_{(10m)}$
6	756	760	**756**	–	–	**0.00**	−0.53	$F10_{(1h)}$
7	1017	1028	**1017**	–	–	**0.00**	−1.08	$F8_{(30m)}$
A	189	200	200	11	11	5.82	0.00	$F8_{(10m)}$
D	621	665	**648**	44	**27**	**4.35**	−2.62	$F8_{(30m)}$
E	756	799	**776**	43	**20**	**2.65**	−2.96	$F10_{(1h)}$
F	738	815	**779**	77	**41**	**5.56**	−4.62	$F7_{(1h)}$
G	999	1121	**1066**	122	**67**	**6.71**	−5.16	$F10_{(2h)}$
Avg.	620.7	646.8	634.5			2.09	−1.44	

Table 7
Soft requirement satisfaction for the best solutions found.

Id	obj	gap_L (%)	S1	S2	S3	H6
1	202	0.00	4	0	1	1
2	333	0.00	0	0	0	0
3	423	0.00	3	0	0	20
4	652	0.00	3	0	0	12
5	762	0.00	2	0	0	11
6	756	0.00	6	0	0	26
7	1017	0.00	6	0	0	17
A	200	5.82	3	0	2	0
D	648	4.35	3	0	18	0
E	776	2.65	2	0	14	0
F	779	5.56	7	0	20	0
G	1066	6.71	7	0	46	0

International Timetabling Competition 2011, except for instance A, where the result matches the previous best known value. We would like to emphasize that the previous results were obtained by several techniques, and no time limit was imposed. This clearly illustrates the effectiveness of our method.

Table 7 presents individually the level of satisfaction regarding soft requirements for the best solutions produced in this study. Columns *obj* and gap_L present, respectively, the objective value and the optimality gap for each instance. Column *S1* presents the number of idle periods. Column *S2* presents the number of working days exceeding the minimum number of days defined by constraint set (13) from the problem formulation. Column *S3* presents the number of unsatisfied double lessons. Column *H6* presents the number of non-consecutive lessons. We recall that requirement H6 is not taken into account for instances of set-1.

From Table 7, it can be appreciated that virtually all solutions present some violations of soft requirements, but these are expected and occur even in the optimal solutions. Among the soft requirements, the minimization of working days was the only one thoroughly satisfied in all solutions. Regarding the requirement S3, we can observe a discrepancy when comparing solutions of set-1 and set-2. In set-1 there is only a single violation for instance 1, while for set-2 several violations are identified. This difference suggests that considering H6 as a hard requirement impacts directly in the satisfaction of double lessons. In other words, it is hard to satisfy simultaneously both the soft requirement S3 and the hard requirement H6.

6. Conclusions

In this paper, we presented a novel approach for solving a variant of the high school timetabling problem which explores class, teacher and day decompositions. We proposed a fix-and-optimize heuristic combined with a variable neighborhood descent method that produces solutions which satisfy all hard constraints, i.e., feasible solutions. In addition, we proposed a simple construction procedure that quickly generates feasible initial solutions. The experimental results show that our approach provides high quality feasible solutions in a smaller computational time when compared with results obtained with the general-purpose integer programming solver CPLEX. We have improved best known solutions in the case of 7 out of 12 instances quoted in the literature. Among these new solutions, three are new optimal solutions for classical instances that have been available since 2000. Further, our method was able to obtain better solutions for four out of five CTTPCR instances from the first round of the Third International Timetabling Competition (held in 2011), outperforming the state-of-the-art techniques. These results show that the proposed technique is very promising to solve the CTTPCR, motivating its use to variants of this problem, as well as to other general combinatorial optimization problems.

Acknowledgments

This work is partly supported by the CAPES Foundation and Petrobras, Brazil.

References

[1] Gotlieb C. The construction of class-teacher timetables. In: IFIP. Amsterdam; 1963. p. 73–7.
[2] de Werra D. Construction of school timetables by flow methods. INFOR 1971; (9):12–22.
[3] Even S, Itai A, Shamir A. On the complexity of time table and multi-commodity flow problems. In: Proceedings of the 16th annual symposium on foundations of computer science. SFCS '75; Washington, DC, USA: IEEE Computer Society; 1975. p. 184–93.
[4] Post G, Kingston J, Ahmadi S, Daskalaki S, Gogos C, Kyngas J, et al. XHSTT: an XML archive for high school timetabling problems in different countries. Ann Oper Res 2011;1–7.
[5] Drexl A, Salewski F. Distribution requirements and compactness constraints in school timetabling. Eur J Oper Res 1997;102(1):193–214.
[6] Schaerf A. A survey of automated timetabling. Artif Intell Rev 1999;13 (2):87–127.
[7] Hansen P, Mladenović N. Variable neighborhood search: principles and applications. Eur J Oper Res 2001;130(3):449–67.
[8] Lodi A. Mixed integer programming computation. In: 50 years of integer programming 1958–2008. Berlin, Germany: Springer; 2010. p. 619-45.
[9] Birbas T, Daskalaki S, Housos E. School timetabling for quality student and teacher schedules. J. Sched. 2008;12(2):177–97.
[10] Souza M, Maculan N. Melhorando quadros de horário de escolas através de caminhos mínimos. Tend. Mat. Apl. Comput. 2000;1(2):515–24.
[11] Souza M. Programação de horários em escolas: uma aproximação por metaheuristicas. [Ph.D. thesis], Universidade Federal do Rio de Janeiro; Rio de Janeiro, Brazil, 2000.
[12] Santos HG, Uchoa E, Ochi LS, Maculan N. Strong bounds with cut and column generation for class-teacher timetabling. Ann Oper Res 2012;194:399–412.
[13] Dorneles ÁP, Araújo OCB, Buriol LS. The impact of compactness requirements on the resolution of high school timetabling problem. In: Proceedings of XLIV Simpósio Brasileiro de Pesquisa Operacional (SBPO 2012); 2012. p. 3336–47.
[14] Maniezzo V, Stützle T, Voss S. Matheuristics: hybridizing metaheuristics and mathematical programming, vol. 10. New York, USA: Springer; 2009.
[15] Gintner V, Kliewer N, Suhl L. Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. OR Spectr 2005;27(4):507–23.
[16] Pochet Y, Wolsey L. Mixed integer programming algorithms. In: Production planning by mixed integer programming. New York, USA: Springer; 2006. p. 77–113.
[17] Wolsey LA. Integer programming. London: Wiley; 1998.
[18] Burke EK, Mareček J, Parkes AJ, Rudová H. Decomposition, reformulation, and diving in university course timetabling. Comput Oper Res 2010;37(3):582–97.
[19] Gunawan A, Ng KM, Poh KL. A hybridized lagrangian relaxation and simulated annealing method for the course timetabling problem. Comput Oper Res 2012;39(12):3074–88.
[20] Avella P, D'Auria B, Salerno S, Vasil'ev I. A computational study of local search algorithms for Italian high-school timetabling. J Heuristics 2007;13:543–56.
[21] IBM. ILOG CPLEX 12.1 User's manual. Mountain View, CA; 2009. URL ⟨http://www.ilog.com/products/cplex⟩.
[22] LABIC. Instances of school timetabling. 2008. URL ⟨http://labic.ic.uff.br/Instance/index.php?dir=SchoolTimetabling⟩.
[23] ITC2011. Third international timetabling competition. 2011. URL ⟨http://www.utwente.nl/ctit/hstt/itc2011⟩.