European Journal of Operational Research 265 (2018) 81–98

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Parallel local search algorithms for high school timetabling problems

Landir Saviniec[a,*], Maristela O. Santos[a], Alysson M. Costa[b]

[a] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Avenida Trabalhador São-carlense 400, 13566-590, São Carlos, São Paulo, Brazil

[b] School of Mathematics and Statistics, The University of Melbourne, 813 Swanston Street, Parkville VIC 3010, Australia

ARTICLE INFO

Article history:
Received 20 October 2016
Accepted 8 July 2017
Available online 13 July 2017

Keywords:
Timetabling
Parallel metaheuristics
Iterated local search
Tabu search
Simulated annealing

ABSTRACT

High school timetabling consists in assigning meetings between classes and teachers, with the goal of minimizing the violation of specific soft requirements. This family of problems has been frequently considered in the literature, but few strategies employing parallelism have been proposed. In this exploratory study, we consider two different parallel frameworks and present a thorough computational study in order to understand algorithmic decisions that are closely related to performance. Our best algorithm outperforms state-of-the-art algorithms for variants of the problem considered, indicating both the efficiency and the flexibility of the method.

1. Introduction

Educational timetabling problems consist in scheduling encounters between teachers (or exams) and students. Specific situations give origin to different problems, with a richness of characteristics. The scientific literature has branched this family of problems in three main sub-areas: University course timetabling Di Gaspero, McCollum, and Schaerf (2007); Lewis (2008); Lewis, Paechter, and McCollum (2007), examination timetabling (McCollum, McMullan, Burke, Parkes, & Qu, 2007; Qu, Burke, McCollum, Merlot, & Lee, 2009) and high school timetabling (De Werra, 1985; Pillay, 2014; Post et al., 2012; Schaerf, 1999). Each of these three families contains their own set of specific constraints.

In this article, we address the high school timetabling problem (HSTP). A restricted decision version of the HSTP was shown to be NP-complete by polynomial reduction of the 3-SAT problem (Even, Itai, & Shamir, 1975). Instances originating from practical contexts naturally extend the requirements of this idealized problem. Extra requirements are usually associated with pedagogical preferences and social/cultural particularities. Examples of these extra requirements are the need to assign double lessons (two consecutive lessons) for some classes subjects and the need to obtain compact schedules for teachers (Pillay, 2014).

The HSTP is usually modeled by mixed-integer programming (MIP) models (Al-Yakoob & Sherali, 2015; Dorneles, de Araújo, & Buriol, 2014; 2017; Kristiansen, Sørensen, & Stidsen, 2015; Santos, Uchoa, Ochi, & Maculan, 2012; Sørensen & Dahms, 2014). The resolution of these models for most medium and large practical instances are known to be still a challenge for actual black-box MIP solvers, which may be why most of the literature is concerned with metaheuristic techniques (Pillay, 2014). The main goal in these metaheuristic studies is to develop strategies to generate good quality solutions within reasonable computational efforts. Recently proposed metaheuristics can be found in Dorneles et al. (2014); Fonseca and Santos (2014); Fonseca, Santos, and Carrano (2016a); Fonseca, Santos, Toffolo, Brito, and Souza (2016c); Saviniec, Constantino, Romão, and Santos (2013). These heuristics are shown to be efficient in the sense that near-optimal solutions are consistently found for different input instances.

In this article, we aim to investigate parallel metaheuristics for the HSTP, in which different solution methods (agents) are run concurrently in different processor threads. The motivation is two-fold. On one hand, the availability of multi-processor machines and appropriate coding schemes make the use of parallel algorithms more accessible than ever. On the other hand, although the literature is abundant in parallel solution methods for similar complex problems (Subramanian, Drummond, Bentes, Ochi, & Farias, 2010; Bożejko, Pempera, & Smutnicki, 2013; Sánchez-Oro, Sevaux, Rossi, Martí, & Duarte, 2015; Luque & Alba, 2015, e.g.), very little attention has been given to the design of parallel methods for the HSTP (Abramson, 1991; Abramson & Abela, 1992; Srndic, Pandzo, Dervisevic, & Konjicija, 2009). The reader is referred to Section 3 for a closer look at these contributions.

As an exploratory study, our goal is to investigate a number of questions related to the design of parallel metaheuristics in the context of the HSTP. The main research questions can be summarized as: (1) What are good parallel strategies for the HSTP? (2) Can these strategies improve the performance of stand-alone

* Corresponding author.
E-mail addresses: landir.saviniec@gmail.com (L. Saviniec), mari@icmc.usp.br (M.O. Santos), alysson.costa@unimelb.edu.au (A.M. Costa).

http://dx.doi.org/10.1016/j.ejor.2017.07.029
0377-2217/

metaheuristics? (3) Are the proposed parallel metaheuristics competitive with state-of-the-art methods?

In order to provide some insight into these questions, we propose a thorough computational study that tests all parallelization schemes resulting from combinations of the following characteristics:

Agents cooperation: we wish to test the effect of allowing the agents to cooperate, by sharing good solutions among threads.

Diversification: we wish to test the effect of using all agents as search intensification mechanisms or allowing at least one agent to diversify the search.

Agents diversity: we wish to test homogeneous and heterogeneous algorithms. An algorithm is *homogeneous* when all threads execute the same metaheuristic, and *heterogeneous* otherwise.

As stand-alone metaheuristics, we use the iterated local search from Saviniec et al. (2013) and adapt versions of tabu search, simulated annealing, and late acceptance strategy. We compare the performance of the different parallel algorithm variants against each other, against the stand-alone metaheuristics and against two state-of-the-art algorithms for close variants of the problem Dorneles et al. (2014); Fonseca et al. (2016a).

The remainder of this paper is organized as follows. Section 2 describes the problem both in plain language and with the help of a formal mixed-integer programming formulation. Section 3 gives an introduction to different parallelization schemes and reviews the existing parallel algorithms for the HSTP. Section 4 explains our parallel metaheuristics. Section 5 presents our computational experiments and Section 6 concludes this paper with final remarks and suggestions for further investigations. An appendix completes this article with details on the used metaheuristics and their parameter setting procedures.

2. The high school timetabling problem

We focus on a real HSTP motivated by Brazilian high schools timetabling rules. In this context, *classes* are disjoint groups of students enrolled in the same set of subjects (mathematics, chemistry, e.g.) and with no free periods during the week. The school also has a set of teachers in its workforce and the goal of the problem is to obtain weekly timetable specifying the schedule of meetings for class/teacher pairs.

Definition 1 (HSTP instance)**.** A HSTP instance is the input data for the timetabling construction in a particular shift. It is represented by the following parameters:

- C: a set of classes.
- T: a set of teachers.
- D: a set of weekdays.
- H: a set of periods per day.
- H_{td}: the set of periods in day $d \in D$ for which teacher $t \in T$ is available.
- RL_{ct}: the number of weekly required lessons to be taught by teacher $t \in T$ to class $c \in C$.
- LM_{ct}: the daily limit for the number of lessons between teacher $t \in T$ and class $c \in C$.
- RDL_{ct}: the number of consecutive double lessons required for a pair ($c \in C$, $t \in T$).

An output for the problem can be represented by a set of binary variables x_{ctdh} for each $(c, t, d, h) \in C \times T \times D \times H$. A feasible solution is an assignment of values to these variables that respects the following hard requirements:

1. *Meeting of weekly required lessons*: all required lessons must be assigned.

$$\sum_{d \in D} \sum_{h \in H} x_{ctdh} = RL_{ct} \qquad \forall\ c \in C;\ t \in T \tag{1}$$

2. *No clashes in classes' schedules*: each class $c \in C$ must attend exactly one lesson per period.

$$\sum_{t \in T} x_{ctdh} = 1 \qquad \forall\ c \in C;\ d \in D;\ h \in H \tag{2}$$

3. *No clashes in teachers' schedules*: each teacher $t \in T$ must teach at most one lesson per period.

$$\sum_{c \in C} x_{ctdh} \leq 1 \qquad \forall\ t \in T;\ d \in D;\ h \in H \tag{3}$$

4. *No assignment of teachers in their unavailable periods*: teachers must not be assigned to periods in which they are unavailable.

$$\sum_{c \in C} x_{ctdh} = 0 \qquad \forall\ t \in T;\ d \in D;\ h \in H \setminus H_{td} \tag{4}$$

An optimal solution is a feasible solution that minimizes the penalties associated with the following soft requirements:

5. *No daily workload violation for class/teacher pairs*: each class/teacher pair should meet no more than LM_{ct} times in a day. For each pair ($c \in C$, $t \in T$), daily workloads greater than LM_{ct} can be measured by auxiliary variables E_{ctd}.

$$E_{ctd} \geq \sum_{h \in H} x_{ctdh} - LM_{ct} \qquad \forall\ c \in C;\ t \in T;\ d \in D \tag{5}$$

$$E_{ctd} \geq 0 \qquad \forall\ c \in C;\ t \in T;\ d \in D \tag{6}$$

6. *No holes in schedules of class/teacher pairs*: Meetings for a pair class/teacher should be consecutive within days. A *hole* is a period that splits the lessons of a pair class/teacher in non-consecutive meetings within the same day. For each pair ($c \in C$, $t \in T$), a hole in period $h \in \{1, \ldots, |H| - 2\}$ is flagged by the auxiliary variables J_{ctdh}.

$$J_{ctdh} \geq x_{ctdi} - x_{ctdh} + x_{ctdj} - 1 \qquad \begin{array}{l} \forall\ c \in C;\ t \in T;\ d \in D; \\ h = 1, \ldots, |H| - 2; \\ i = 0, \ldots, h - 1; \\ j = h + 1, \ldots, |H| - 1 \end{array} \tag{7}$$

$$0 \leq J_{ctdh} \leq 1 \qquad \forall\ c \in C;\ t \in T;\ d \in D;\ h = 1, \ldots, |H| - 2 \tag{8}$$

7. *Meeting of double lessons for class/teacher pairs*: All weekly double lesson requirements should be met. For each pair ($c \in C$, $t \in T$), auxiliary variables U_{ct} count the number of unmet weekly double lessons.

$$U_{ct} \geq RDL_{ct} - \sum_{d \in D} \sum_{h=1}^{|H|-1} W_{ctdh} \qquad \forall\ c \in C;\ t \in T \tag{9}$$

$$U_{ct} \geq 0 \qquad \forall\ c \in C;\ t \in T \tag{10}$$

In these inequalities, additional auxiliary variables W_{ctdh} are equal to 1 if a double lesson for pair ($c \in C$, $t \in T$) ends in period $h \in \{1, \ldots, |H| - 1\}$ in day $d \in D$, and equal to 0 otherwise. These variables can be linked with the original assignment variables with the following linear constraints:

$$W_{ctdh} \leq x_{ctdh} \qquad \forall\ c \in C;\ t \in T;\ d \in D;\ h = 1, \ldots, |H| - 1 \tag{11}$$

$$W_{ctdh} \leq x_{c,t,d,h-1} \qquad \forall\ c \in C;\ t \in T;\ d \in D;\ h = 1, \ldots, |H| - 1 \tag{12}$$

$$W_{ctdh} \leq 1 - W_{c,t,d,h-1} \qquad \forall\ c \in C;\ t \in T;\ d \in D;\ h = 2, \ldots, |H| - 1 \tag{13}$$

$$0 \leq W_{ctdh} \leq 1 \qquad \forall\ c \in C;\ t \in T;\ d \in D;\ h = 1, \ldots, |H| - 1 \tag{14}$$

8. *No idle periods in teachers' schedules*: teachers should not have idle periods while in the school. An *idle period* is a period during which the teacher is not busy, but is busy in earlier and later periods within the same day. For each teacher $t \in T$, an idle period in period $h_k \in H_{td}$ $(k = 1, \ldots, |H_{td}| - 2)$ in day $d \in D$ is flagged by variables J'_{tdk}.

$$J'_{tdk} \geq \sum_{c \in C} x_{c,t,d,h_i} - \sum_{c \in C} x_{c,t,d,h_k} + \sum_{c \in C} x_{c,t,d,h_j} - 1 \qquad \forall\ t \in T;\ d \in D;\ k = 1, \ldots, |H_{td}| - 2;\ i = 0, \ldots, k-1;\ j = k+1, \ldots, |H_{td}| - 1;\ h_i, h_k, h_j \in H_{td} \tag{15}$$

$$0 \leq J'_{tdk} \leq 1 \qquad \forall\ t \in T;\ d \in D;\ k = 1, \ldots, |H_{td}| - 2 \tag{16}$$

In these inequalities, unavailable periods are not computed as idle periods. Inequalities (15) are similar to inequalities (7), which identify holes for class/teacher pairs, except that we omit variable J' for periods in which the teachers are unavailable. For example, consider that the set of periods is $H = \{0, 1, 2, 3, 4\}$ and suppose that a teacher $t \in T$ is not available to teach at periods 0 and 1 within a day $d \in D$, then $H_{td} = \{2, 3, 4\}$. Now, only variable $J'_{t,d,3}$ and related constraints are considered.

9. *Minimum number of weekly working days for teachers*: Teachers should be scheduled to come to school the minimum possible number of days. For each teacher $t \in T$, auxiliary variables $\widehat{D}_{td}$ flag if he/she works in day $d \in D$.

$$\widehat{D}_{td} \geq \sum_{c \in C} x_{ctdh} \qquad \forall\ t \in T;\ d \in D;\ h \in H_{td} \tag{17}$$

$$0 \leq \widehat{D}_{td} \leq 1 \qquad \forall\ t \in T;\ d \in D \tag{18}$$

10. *Balancing on teachers' unnecessary working days*: the compactness on working days should be balanced among teachers. This requirement can be modeled by several strategies, as discussed in Burget and Rudová (2016). We consider the worst schedule among teachers. An auxiliary variable β measures the maximum number of extra days of work among all teachers' schedules. The extra days of work for teachers are the difference between the number of days they are assigned to work and the minimum number of days that they could be assigned and still meet their teaching requirements. We use a lower bound on the minimum number of working days for a teacher t, given by $MD_t = \left\lceil \frac{\sum_{c \in C} RL_{ct}}{|H|} \right\rceil$.

$$\beta \geq \sum_{d \in D} \widehat{D}_{td} - MD_t \qquad \forall\ t \in T \tag{19}$$

$$\beta \geq 0 \tag{20}$$

The goal of the problem is to minimize the weighted sum of the violations of soft requirements. A complete mixed-integer program can thus be written as

$$\begin{aligned} \textit{Minimize}\ & \alpha_5 \sum_{c \in C} \sum_{t \in T} \sum_{d \in D} E_{ctd} + \alpha_6 \sum_{c \in C} \sum_{t \in T} \sum_{d \in D} \sum_{h=1}^{|H|-2} J_{ctdh} + \alpha_7 \sum_{c \in C} \sum_{t \in T} U_{ct} \\ & + \alpha_8 \sum_{t \in T} \sum_{d \in D} \sum_{k=1}^{|H_{td}|-2} J'_{tdk} + \alpha_9 \sum_{t \in T} \sum_{d \in D} \widehat{D}_{td} + \alpha_{10} \cdot \beta \end{aligned} \tag{21}$$

Subject to

(1) – (20)

$$x_{ctdh} \in \{0, 1\} \qquad \forall\ c \in C;\ t \in T;\ d \in D;\ h \in H \tag{22}$$

in which weights α_5–α_{10} are adjusted to represent the school preferences.

The formulation is an adaptation of the one proposed by Souza (2000), with the addition of soft requirements 6 (no holes in class/teacher meetings) and 10 (balancing on teachers' unnecessary working days). Also, unlike in Souza (2000), the *daily workload limit for class/teacher pairs* is considered as a soft requirement.

We also use the additional cuts (23) proposed by Souza (2000), which specify that a teacher t cannot work less than the minimum number of working days MD_t.

$$\sum_{d \in D} \widehat{D}_{td} \geq MD_t \qquad \forall\ t \in T \tag{23}$$

3. Existing parallel algorithms for the HSTP

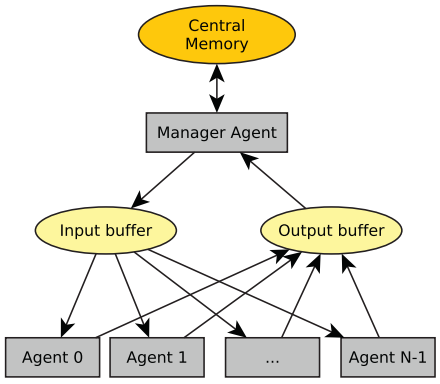
Metaheuristics can be classified into *population-based (PB)* and *trajectory-based (TB)* methods (Alba, Luque, & Nesmachnow, 2013). Population-based metaheuristics are characterized by keeping a pool of solutions (genetic algorithms, ant colony optimization, and particle swarm optimization, e.g.). The method starts with an initial population and employs, at each step, stochastic operators to evolve toward better quality populations. According to Alba et al. (2013), two classical parallel algorithm frameworks have been used for population-based metaheuristics

- *Parallel individuals evaluation (PB-PIE)*: each individual in the population can be evaluated in parallel.
- *Parallel islands (PB-PI)*: the initial population is split into a set of subpopulations (islands) in which a sequential algorithm can be employed. In this framework, the islands often exchange individuals among each other to diversify the subpopulations and prevent early convergence.

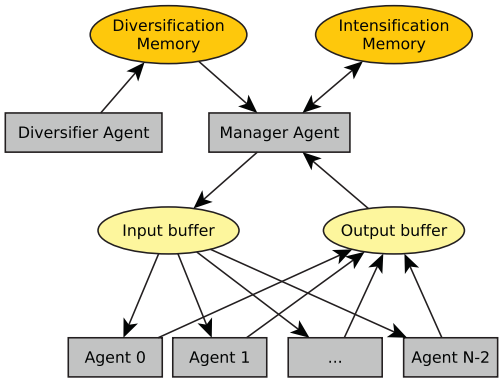
Trajectory-based metaheuristics are characterized by methods which employ a single current solution. The method starts with an initial solution and at each step, goes through neighborhoods of the current solution tracing trajectories in the search space of the problem. Some well-known metaheuristics in this class are: simulated annealing, tabu search, iterated local search, and variable neighborhood search. Three classical parallel algorithm frameworks have been used in the literature for trajectory-based metaheuristics according to Alba et al. (2013):

- *Parallel moves (TB-PM)*: the neighborhood of a current solution is explored in parallel. This is usually implemented by a manager/workers technique. At each step, the manager replicates the current solution to a set of parallel workers, each worker then computes the objective function of one of the neighboring solutions and returns it to the manager, who takes decisions.
- *Move acceleration (TB-MA)*: the objective function of a single solution is evaluated in parallel. This framework is an attractive approach when the sequential function evaluation is both time-consuming and can be decomposed into small parts to be computed in parallel.
- *Parallel multi-start (TB-PMS)*: several asynchronous threads of trajectory-based methods run simultaneously to compute high-quality solutions. These threads may be homogeneous (same method) or heterogeneous (distinct methods). They may or may not cooperate with each other. Also, they may start from the same solution seed or from different ones.

Luque and Alba (2015) state that parallel strategies for trajectory-based methods have been less studied than for



(a) Central memory based: a parallel multi-start framework using a manager/workers strategy and a central memory of solutions.



(b) Diversification-intensification memory based: a parallel multi-start framework with diversification and intensification memories.

Fig. 1. Parallel multi-start metaheuristic frameworks.

population-based ones. For HSTP, we have found only three studies related to parallel metaheuristics. Abramson (1991) proposed a simulated annealing which employs a series of locks to enable several processes to perform simultaneous moves in the same timetable data structure (TB-PM). Abramson and Abela (1992) proposed a genetic algorithm that, at each step, generates and evaluates individuals in parallel (PB-PIE). Srndic et al. (2009) proposed a genetic algorithm approach in which the global population is divided into small subpopulations that are managed by different processes (PB-PI).

In this paper, we focus on trajectory-based methods. Specifically, we propose parallel multi-start methods for the HSTP using as agents a sequential iterated local search (Lourenço, Martin, & Stützle, 2003), a tabu search (Glover, 1990), a simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983) and a late acceptance strategy (Burke & Bykov, 2008) metaheuristics. Details of these metaheuristics are presented in the Appendix. In the next section, we concentrate on the parallel algorithms.

4. Proposed parallel metaheuristics

The proposed algorithms follow the trajectory-based parallel multi-start (TB-PMS) scheme described in the previous section and employ manager/workers strategies with shared memories. We propose the use of two main strategies. The first is based on central memory (Crainic, Gendreau, Hansen, & Mladenović, 2004) and the second is based on diversification and intensification memories (Jin, Crainic, & Løkketangen, 2014).

4.1. Central memory based

The central memory based (CMB) framework, shown in Fig. 1(a), has a central memory which keeps up to X solutions, a manager procedure, N metaheuristic worker agents and two synchronization buffers, input and output. The main idea in this framework is that a group of metaheuristic agents can execute concurrently while possibly cooperating with each other by means of an exchange of current solutions. This exchange is done via the central memory, which keeps a set of elite solutions.

Each agent pulls a solution from the input buffer, operates on it for a parameter-defined amount of time, and then returns the best found solution to the output buffer. The manager is responsible for implementing the policies which will determine the exchange of solutions between the two intermediate buffers (input and output) and the central memory.

In our implementations with this framework, the following policies are adopted:

- *Memory initialization*: the central memory is initialized with solutions generated by the constructive heuristic described in Algorithm 1, see the Appendix.
- *Input selection*: a solution is randomly selected from the central memory and put into the input buffer whenever requested by an agent.
- *Output acceptance*: when the manager retrieves a solution Z_r from the output buffer, the solution Z_r is compared sequentially with solutions in the central memory. A first improvement strategy is adopted, i.e., the first visited solution that is worse than or equal to Z_r is replaced. If no solution meets this criterion, then Z_r is discarded.
- *Agents' execution time*: each of the N agents is allowed to run for Y seconds before pushing its best solution into the output buffer and requesting a new solution from the input buffer. Short execution times define high levels of cooperation among agents while long execution times define more independent agents.

4.2. Diversification-intensification memory based

The diversification-intensification memory based (DIMB) framework, shown in Fig. 1(b), is an extension of the CMB framework. The central memory is now divided into a 'diversification memory' and an 'intensification memory'. The first memory keeps a set of non-elite solutions to diversify the search while the second keeps a set of elite solutions.

One agent is responsible for providing diversified solutions to the diversification memory while all others aim at finding good quality solutions. The details of our implementation are as following:

- *Memory initialization*: the two memories are initialized with solutions generated by the constructive heuristic described in Algorithm 1.
- *Input selection*: whenever requested by an agent, a solution is selected from the diversification memory with probability ρ and from the intensification memory with probability $(1-\rho)$.
- *Output acceptance*: when the manager retrieves a solution Z_r from the output buffer, it accepts the solution into the intensification memory if it is better than or equal to a solution randomly chosen in the intensification memory, otherwise Z_r is discarded.

- *Agents' execution time*: each of the $N-1$ intensification agents is allowed to run for Y seconds before pushing its best solution into the output buffer and requesting a new solution from the input buffer.
- *Diversification memory update*: The solutions of this memory are continuously replaced by solutions with better or equivalent quality generated by the diversifier agent during run-time. The diversifier is, usually, a metaheuristic with slow convergence or that quickly converges to poor local optima and keeps doing plain moves (moving to other solutions with the same quality). In our implementation, the diversifier agent is a late acceptance strategy metaheuristic with plain moves and set with parameters that lead to premature convergences. Whenever the late acceptance strategy updates its current solution Z_c, the solution Z_c is compared with a random solution Z_d in the diversification memory. If Z_c is better or equal to Z_d in terms of objective value, the metaheuristic replaces Z_d by Z_c.

In the next Section, we test these two frameworks with several parameter configurations. We investigate cases in which the agents are homogeneous or heterogeneous and with independent or cooperative search.

5. Computational experiments

In this section, we conduct a thorough computational study in order to evaluate the performance of the proposed parallel methods. All algorithms were coded in C++ and compiled with GNU Compiler Collection 4.4.7. The experiments were made on a server running *Red Hat Enterprise Linux 6.5*. The hardware is composed of two CPU Intel Xeon E5-2680v2 (2.8 gigahertz) and 128 gigabytes of RAM. To implement parallelism we employed the POSIX Threads Library (Pthreads) available in the GNU Compiler Collection. The proposed MIP model was implemented with Concert Technology Libraries from IBM ILOG CPLEX 12.5. In all experiments reported, the metaheuristics are allowed to execute during a time limit of 625 seconds and 25 trials (different seeds) are used for each instance. The seeds were generated by standard C++ libraries and we used the same seeds for every metaheuristic configuration.

The experiments have been set up in two phases: in Section 5.1, we compare the different proposed methods against each other. The following tests are executed (the appropriated subsection where the results can be found is indicated):

5.1.1 Tests with the CMB parallel framework proposed in Section 4.1.
5.1.2 Tests with the DIMB parallel framework proposed in Section 4.2.
5.1.3 Tests comparing the best sequential and parallel algorithm configurations.

Then, in Section 5.2, we compare the best configuration found with results obtained by the MIP solver CPLEX applied to the formulation proposed in Section 2. The best configuration is also adapted and compared with state-of-the-art literature methods for two close variants of the problem proposed here:

5.2.1 Tests comparing our best algorithm against CPLEX.
5.2.2 We compare our best algorithm with the solver winner of the Third International Timetabling Competition (ITC2011).
5.2.3 We compare our best algorithm with approaches reported to the HSTP proposed by Souza (2000).

The interested reader is referred to the Appendix, where the used stand-alone metaheuristics are described along with the procedure we used for setting the best parameters.

5.1. Comparisons among proposed methods

For tests 5.1.1–5.1.2, we use the reduced set of six instances from Table A.4, which were used to calibrate the sequential metaheuristics in Section A.6 in the Appendix. For tests 5.1.3, we use all 34 instances described in Table A.4.

5.1.1. Analysis of results for CMB parallel metaheuristics

The two parallel frameworks CMB and DIMB described in Section 4 generate different algorithms for different combinations of metaheuristic agents. Also, agents with short execution times (parameter Y) generate cooperative searches while agents with large execution times lead to independent searches. In this section, we analyze algorithms which include homogeneous, heterogeneous, cooperative and independent agents. We assess how much these parallel algorithms are better than the sequential ones and how cooperation can play a role in the quality of solutions.

We first analyze four cases of CMB parallel metaheuristics. In each case, the agents are homogeneous threads of one of the metaheuristics ILS, TS, SA or LAS. For each case, we tested configurations in which the central memory contains $X = (1, 4, 8, 16, 32)$ solutions and agent time $Y = (1, 2, 5, 10, 30, 90, 315, 625)$ seconds. Cases with homogeneous (HM) ILS showed the overall best results.

Fig. 2 plots the best configurations with homogeneous ILS. In this figure and all the remaining figures in this paper, the y-axis represent normalized objective values obtained as follows. Let Z_j be a timetable solution for the j-th instance, then its normalized objective value is given by $NOV(Z_j) = f(Z_j)/f(Z_j^*)$. Where $f(Z_j^*)$ is a benchmark objective value for instance j, which can be a lower bound or a best known solution. As at each point x, each metaheuristic collects 25 samples for each instance, the final y-value in a point x is the median of the instances' median sample. In these charts, the reference line $(y = 1)$ represents the benchmark objective value $f(Z_j^*)$. In other words, the charts summarizes the results for all instances and indicates, for each time point, how close or far the algorithm median solutions are from the benchmark value, providing insights on the algorithm convergence rates.

The figure plots the two best configurations with cooperative agents (*thin lines*) which were HM-ILS-1-30 with $(X, Y) = (1, 30)$ and HM-ILS-4-30 with $(X, Y) = (4, 30)$, the best configuration with independent agents (*blue-thick line*) which is HM-ILS-4-625 with $(X, Y) = (4, 625)$ and the best sequential metaheuristic, the stand-alone ILS (*red-thick line*). These results indicate that parallel versions with homogeneous agents provide better quality solutions in less computational time than sequential versions. We also observed this fact in the experiments using TS, SA and LAS as agents. In Fig. 2, we also observe that after 400 seconds, configurations with cooperative ILS agents perform slightly better than their independent counterparts.

Fig. 3 plots the best configurations with homogeneous TS (*black lines*) and LAS (*blue lines*). We also plot the results of the stand-alone ILS in order to provide a common benchmark among different charts. The figure shows that configurations with cooperative agents (*thin lines*) perform significantly better than configurations with independent agents (*thick lines*).

On the other hand, for SA, cooperative agents performed worse than independent agents. This is shown in Fig. 4, where the *thin line* plots the best configuration with cooperative agents, the *blue-thick line* represents the best configuration with independent agents and the *black-thick line* is the best sequential SA. These results probably indicate that more sophisticated tuning procedures need to be used for the parallel SA, in order to balance the diversification provided by the parallel contribu-

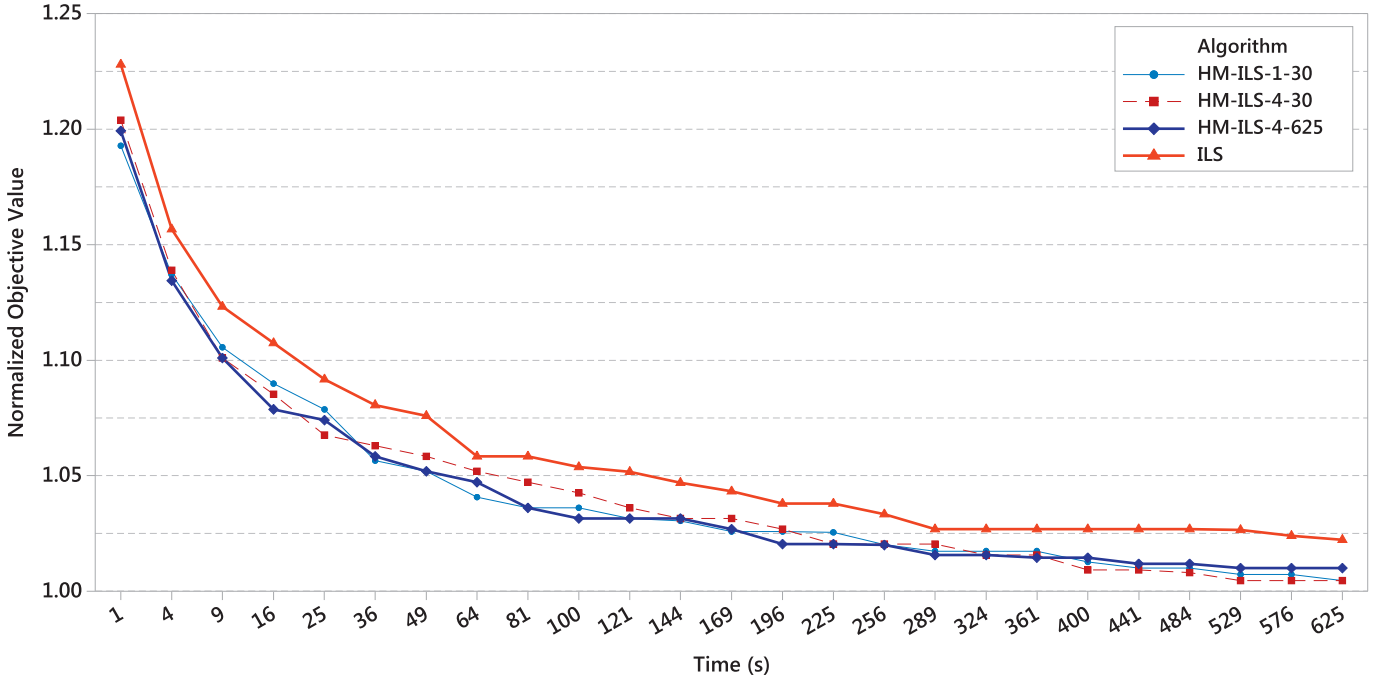


Fig. 2. Best results for CMB with homogeneous ILS agents (4 threads).

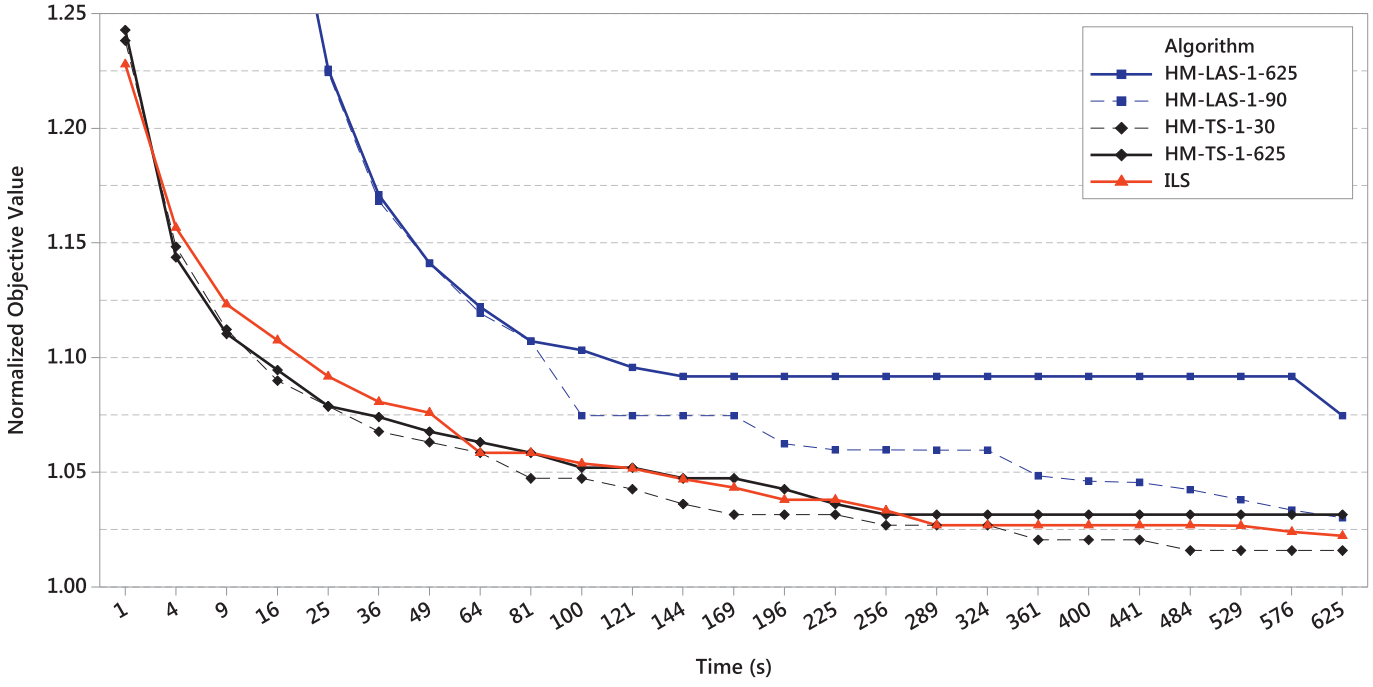


Fig. 3. Best results for CMB with homogeneous TS and LAS agents (4 threads).

tion and the natural diversification structure of the SA for high temperatures.

For CMB algorithms with heterogeneous agents, we tested the case in which there is one agent thread for each metaheuristic ILS, TS, SA and LAS.

The TS agent was set with parameters (*stSize*, *tbTime*) = (100, 100), SA with (T_0, α, *stSize*, ϵ) = (100, 0.9999, 25, 1) and LAS with (*lsSize*, *stSize*) = (100, 100). These parameters were chosen based on our sequential experiments and considering a trade-off between fast convergences and good quality of solutions. As before, we tested configurations with central memory sizes X = (1, 4, 8, 16, 32) and agent times Y = (1, 2, 5, 10, 30, 90, 315, 625). In Fig. 5, we plot the two best configurations with cooperative agents (*thin lines*) which were (X, Y) = (1, 10) and (1, 30) and the best configuration with independent agents (*blue-thick line*) which is (X, Y) = (1, 625). We observe that, configurations with cooperative agents (*thin lines*) perform better than configurations with independent agents (*blue-thick line*). Nevertheless, no heterogeneous configuration presented better results than the homogeneous ILS configuration *HM-ILS-1-30*, plotted in *black-thick line* in Fig. 5.

Finally, it is interesting to see how all best versions used the central memory with a single solution, indicating a strong elitism characteristic.

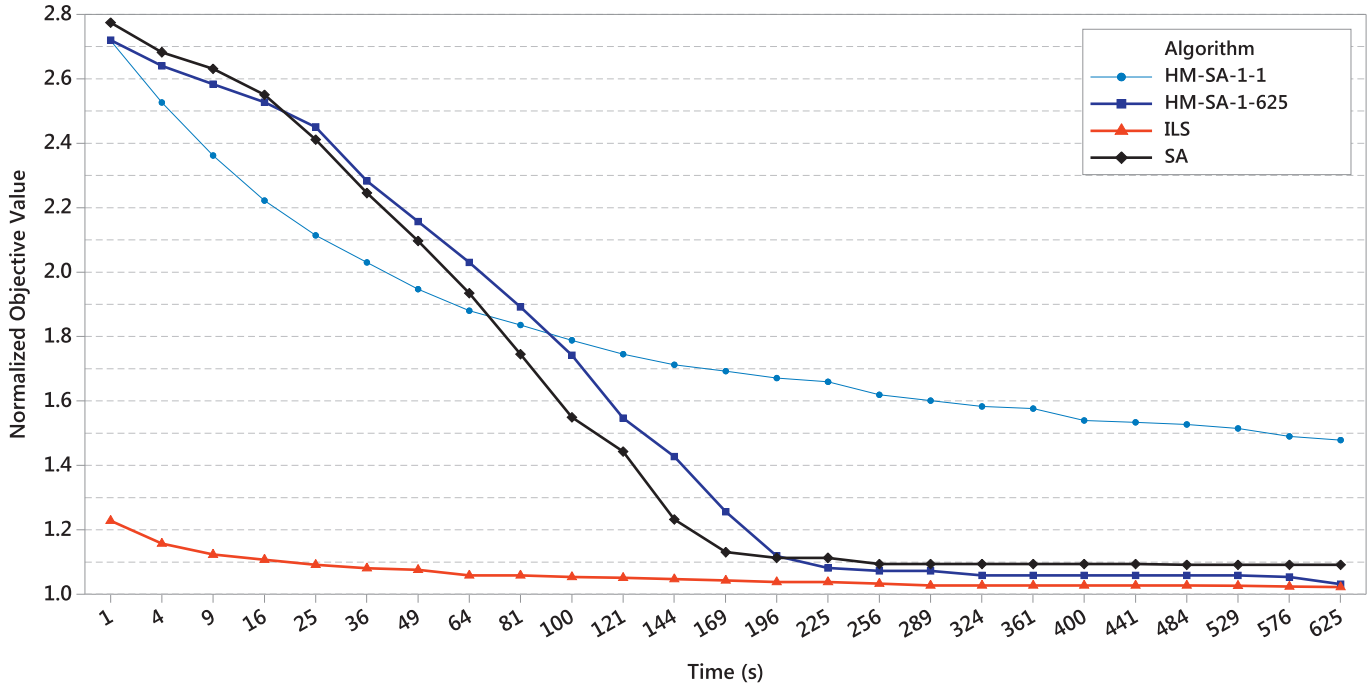


Fig. 4. Best results for CMB with homogeneous SA agents (4 threads).

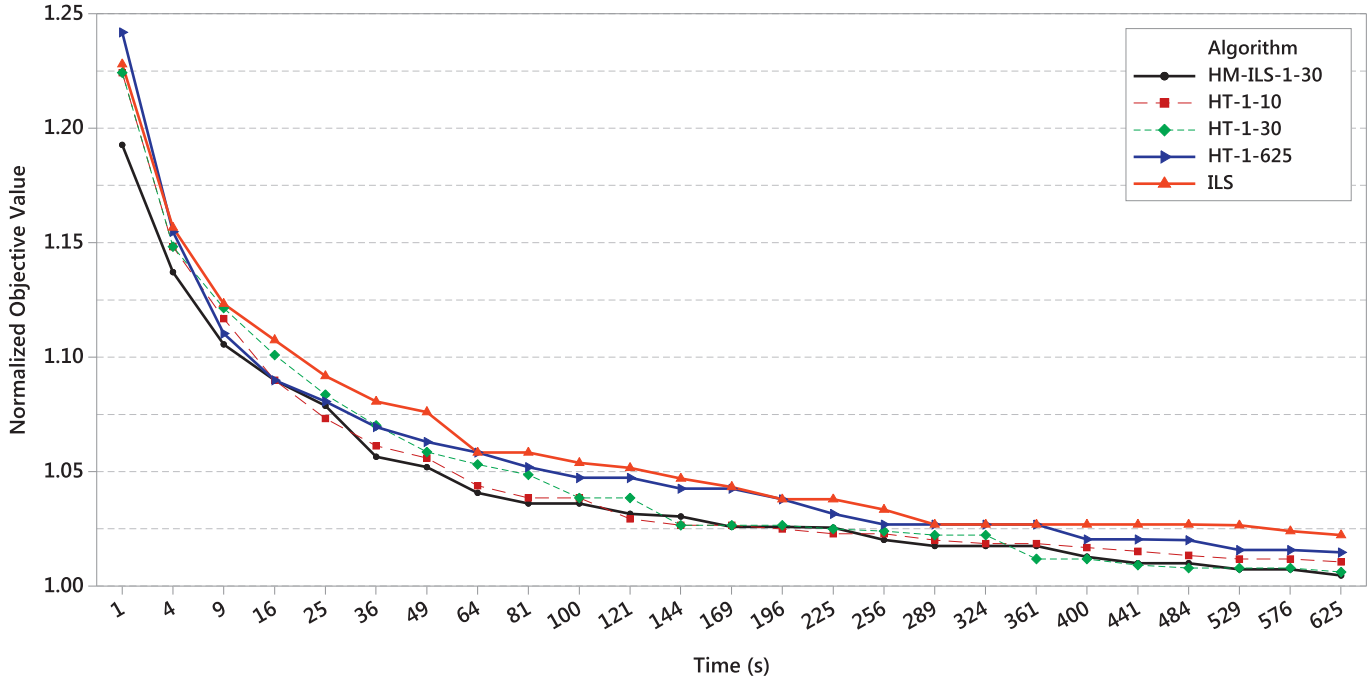


Fig. 5. Best results for CMB with heterogeneous agents (4 threads).

5.1.2. Analysis of results for DIMB parallel metaheuristics

In this section, we analyze a case of DIMB parallel metaheuristics in which the intensifier agents are several threads of ILS, which proved earlier to be the best sequential intensifier agent. As for diversification, our preliminary tests showed that the LAS is able to quickly converge to local optima with quality levels near to 10% worse than the benchmark values and keep moving to other solutions with the same quality. Based on these results, the LAS with parameters ($lsSize$, $stSize$) = (1000, 25) was chosen as our diversifier agent.

The DIMB algorithm was tested with four intensifiers (with a single solution in the intensification memory) and diversification memories with sizes X_D = (1, 4, 8, 16, 32) combined with probabilities ρ = (0.05, 0.10, 0.20, 0.25, 0.50). The best result were obtained with configuration X_D = 8 and ρ = 0.05. We also tested this configuration with nineteen intensifier threads. In Fig. 6, we show these two configurations (algorithms DIMB-4I1D-8-0.05 and DIMB-19I1D-8-0.05) and the best configuration of the CMB obtained earlier (algorithm HM-ILS-1-30). Comparing algorithms DIMB-4I1D-8-0.05 and HM-ILS-1-30, we observe that the scheme with diversification-intensification memories (DIMB) presents a better performance than the scheme with central memory (CMB). Fig. 6 also shows

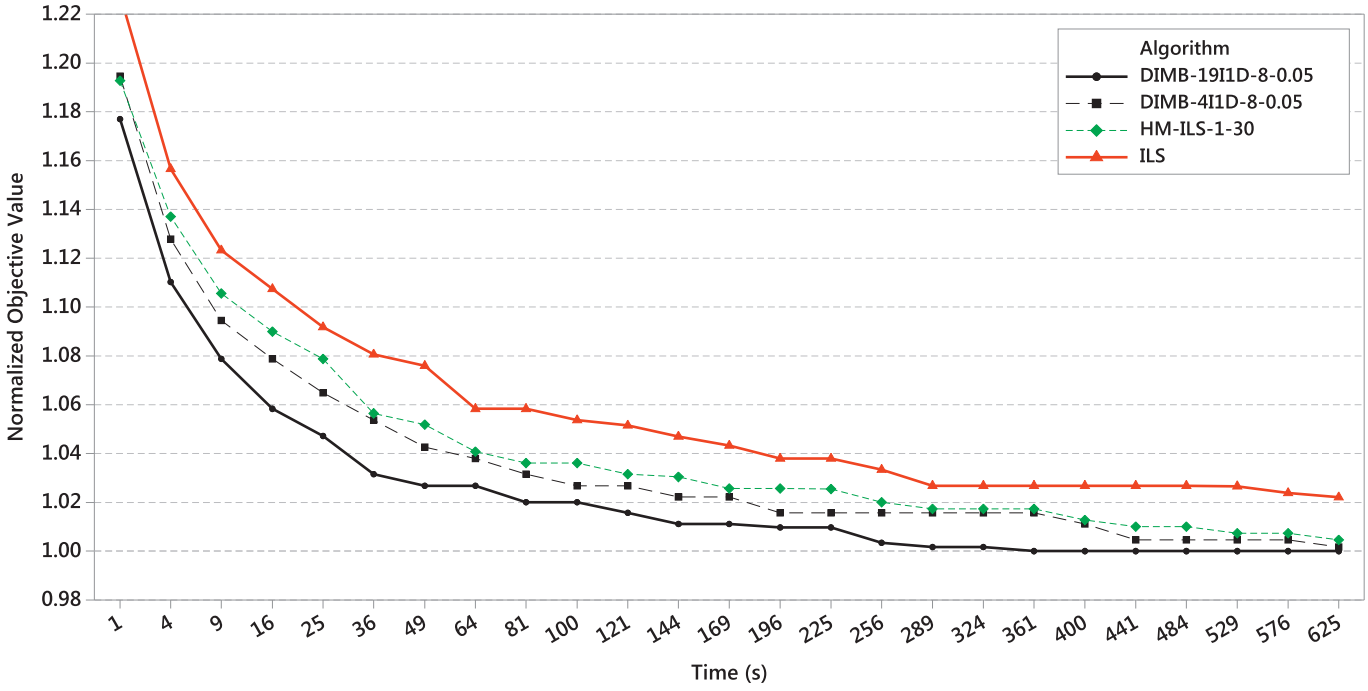


Fig. 6. Best results for DIMB parallel metaheuristics.

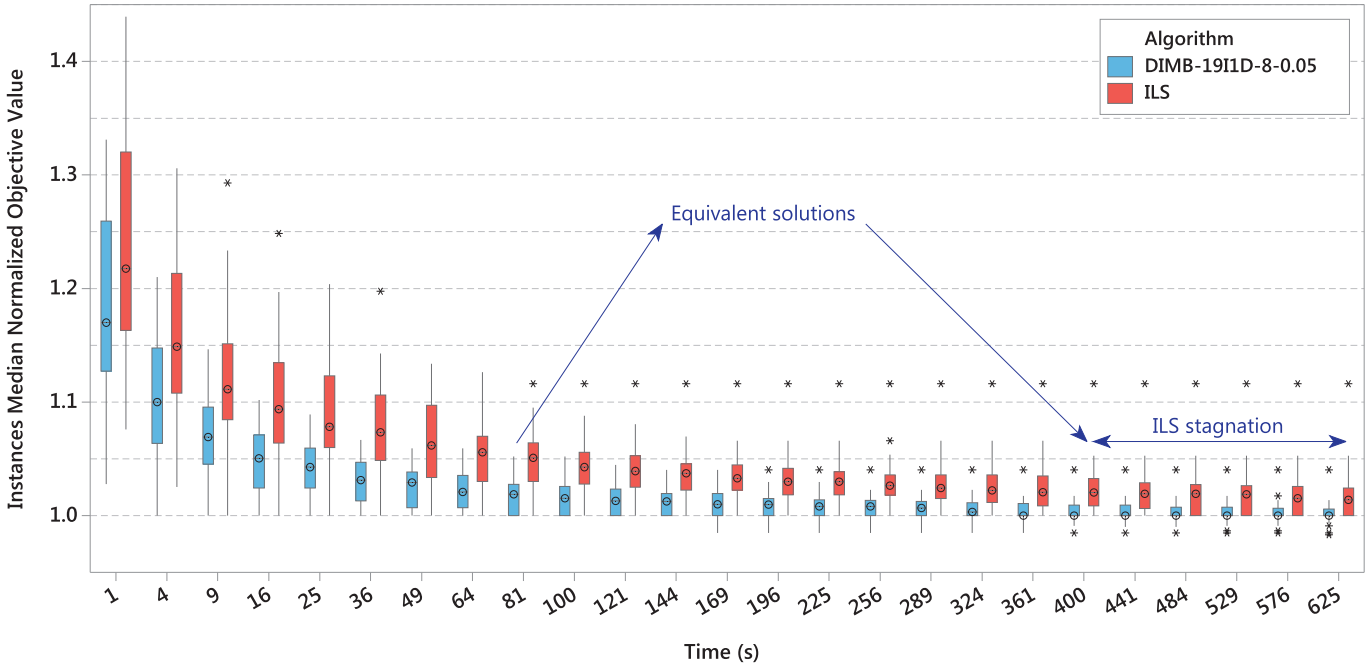


Fig. 7. Results of the best parallel algorithm DIMB-19I1D-8-0.05 compared to results of the best sequential ILS in all 34 instances from Table A.4. The asterisk symbols (*) represent outliers (best or worst solutions).

that increasing the number of threads has a beneficial effect on performance.

5.1.3. Sequential versus parallel metaheuristics

Fig. 7 plots the distribution of the instances' median samples for parallel and sequential algorithms for the 34 instances from Table A.4. This analysis provides some insight on the algorithms' robustness throughout the whole set of instances.

The best solutions of the sequential ILS were used as benchmark values. We observe that, the parallel algorithm solutions after 81 seconds are statistically equivalent (supported by a Kruskal–Wallis test for the equality of medians) to those of the sequential ILS algorithm after 400 seconds, which is the time when the sequential ILS stagnates. Apart from being faster, the parallel algorithm is also more robust than the sequential one in all instances, as it presents less scattered distributions.

5.2. Comparisons with state-of-the-art algorithms

In this section, we aim to provide evidence that the proposed parallel algorithm presents state-of-the-art results not only for the high school timetabling problem described in Section 2, but also for close variants of the problem which were already studied in the literature.

Table 1
Results of algorithm DIMB-19I1D-8-0.05 compared to results of CPLEX in all 34 instances from Table A.4.

	CPLEX (after 3 h)			DIMB-19I1D-8-0.05 (after 625 s)			
ID	LB	UB	Gap	Median	Gap	Best	Gap
1	878	1275	31.14	970	9.48	**950**	7.58
2	874	1190	26.55	970	9.90	**950**	8.00
3	913	1295	29.50	1020	10.49	**1010**	9.60
4	910	1270	28.35	1000	9.00	**990**	8.08
5	835	960	13.02	**870**	4.02	**870**	4.02
6	1070	1390	23.02	1100	2.73	**1080**	0.93
7	1642	2535	35.23	1880	12.66	**1810**	9.28
8	950	**950**	0.00	1030	7.77	980	3.06
9	1036	1320	21.52	1170	11.45	**1130**	8.32
10	420	**420**	0.00	**420**	0.00	**420**	0.00
11	1375	1895	27.44	1530	10.13	**1450**	5.17
12	1396	2140	34.77	1600	12.75	**1530**	8.76
13	420	**420**	0.00	**420**	0.00	**420**	0.00
14	1176	1720	31.63	1360	13.53	**1300**	9.54
15	395	**395**	0.00	**395**	0.00	**395**	0.00
16	584	750	22.13	650	10.15	**640**	8.75
17	529	670	21.04	590	10.34	**580**	8.79
18	443	480	7.71	**460**	3.70	**460**	3.70
19	350	**360**	2.78	**360**	2.78	**360**	2.78
20	431	500	13.80	**470**	8.30	**470**	8.30
21	1364	2235	38.97	1550	12.00	**1480**	7.84
22	1012	1255	19.36	1095	7.58	**1055**	4.08
23	2423	14085	82.80	2790	13.15	**2670**	9.25
24	2526	11045	77.13	2970	14.95	**2860**	11.68
25	1301	1610	19.19	1400	7.07	**1340**	2.91
26	732	980	25.31	780	6.15	**750**	2.40
27	716	890	19.55	750	4.53	**730**	1.92
28	1458	2090	30.24	1620	10.00	**1590**	8.30
29	1457	2090	30.29	1620	10.06	**1590**	8.36
30	1451	2205	34.20	1640	11.52	**1600**	9.31
31	1436	2200	34.73	1640	12.44	**1590**	9.69
32	1270	2080	38.94	1440	11.81	**1410**	9.93
33	1284	1930	33.47	1460	12.05	**1430**	10.21
34	1268	2050	38.15	1450	12.55	**1410**	10.07
Avg.:			26.23		8.68		6.49

5.2.1. Comparison with CPLEX

In this section, we compare the best parallel algorithm DIMB-19I1D-8-0.05 with CPLEX results in the 34 instances from Table A.4.

Table 1 shows the lower and upper bounds obtained by CPLEX after 3 hours of computational time, as well as the optimality gap. It also shows the median and best solutions (out of 25 runs) of the DIMB best configuration found earlier. CPLEX could prove optimality for instances 8, 10, 13 and 15 and presented small gaps for instances 18 and 19. With the exception of instance 8, our algorithm median solutions showed gaps (computed using the best lower bound provided by CPLEX after 3 hours) that are smaller than or equal to gaps obtained by CPLEX. In average, our gaps (median case) are three times smaller than the CPLEX's gaps.

5.2.2. The Brazilian HSTP solved in the ITC2011

We compare our parallel algorithm DIMB-19I1D-8-0.05 with the GOAL solver (Fonseca et al., 2016a). This method was the winner of the Third International Timetabling Competition (Post, Di Gaspero, Kingston, McCollum, & Schaerf, 2016) devoted to high school timetabling problems, ITC2011[1]. It provides a pool of methods to solve timetabling problems from different countries. The latest version of this solver (Fonseca, Santos, & Carrano, 2016b), which was used for these comparisons, employs a three-phase approach. At the first phase, the KHE software library (Kingston, 2015) is employed to obtain a starting solution. This solution is refined by a parallel Variable Neighborhood Search metaheuristic (VNS) at the second phase. Finally, the output of VNS is further refined with Fix-and-optimize MIP heuristics.

In this experiment, we consider the Brazilian HSTP described in instances of the ITC2011, named here as XHSTT-BV. These instances model the requirements 1 to 6 as hard constraints and requirements 7 to 9 as soft constraints. As requirement 10 is not defined in the XHSTT format, it is ignored here. We use weighting parameters $\alpha_3 = \alpha_4 = 100000, \alpha_5 = 10000, \alpha_6 = 5000, \alpha_7 = 1, \alpha_8 = 3$ and $\alpha_9 = 9$. Our methods were adapted to cope with these modifications.

Table 2 presents the results of 25 runs of both algorithms for the 34 instances from Table A.4. Both algorithms were set with 20 threads and a time limit of 625 seconds. As suggested by the authors of GOAL, additional parameters of their solver were set to default values (alg-timelimit = 62, form-timelimit = 62, initial-soln = KHE, algorithm = SVNS, formulation = FIXOPT, formfixopt-nresources = 5 and formfixopt-optinarow = 5). The table also includes the results of CPLEX (time limit of 3 hours) for the MIP program generated by the model from Section 2 adjusted to the new variant. Bold fonts highlight the best solutions found. These results show that CPLEX solved to optimality only instances 8 and 13 and proved infeasibility for instances 15 and 22. Instances 10 and 26 were proven optimal by using the heuristic solutions.

Table 2 also shows that our algorithm outperforms GOAL, both in terms of median and best solutions. Our algorithm only performed worse than GOAL in instance 8. The probable reason is that this instance has a large number of teachers' unavailable periods, in which it is expected that MIP neighborhoods (such those employed in GOAL) will perform better than conventional neighborhood approaches (such the one employed in our algorithm). We also note that the average gap of our median solutions (3.54 %) is smaller than the average gap of the best solutions of GOAL (4.47 %).

In Fig. 8, we focus on the computational time spent by the algorithms. The chart shows the distribution of the instances' median solutions generated by DIMB-19I1D-8-0.05 after 25 and 625 seconds and the results of GOAL solver after 625 seconds. A Kruskal–Wallis test for the equality of medians was run and indicated that the solutions generated by our parallel strategy after 25 seconds are of better quality than those of GOAL after 625 seconds.

5.2.3. The HSTP proposed by Souza (2000)

Finally, our best algorithm is adapted to deal with the HSTP proposed by Souza (2000). As mentioned in Section 2, in this problem soft requirements 6 (no holes in class/teacher pairs) and 10 (balancing on teachers' unnecessary working days) are ignored and requirement 5 is considered as a hard requirement.

For these experiments, we consider a set of seven instances[2] which have been used in the literature (Table 3). This problem has been addressed in previous articles (Dorneles et al., 2014; Santos, Ochi, & Souza, 2005; Santos et al., 2012; Saviniec et al., 2013; Souza, Ochi, & Maculan, 2003) which have contributed to finding the instances optimal values. The state-of-the-art algorithms for this problem are based on Fix-and-optimize heuristics (Dorneles et al., 2014) and sequential ILS based metaheuristics (Saviniec et al., 2013). The Fix-and-optimize heuristics were reported to find optimal solutions in each instance, on time limits within 10 to 60 minutes in a computer with processor Intel Core i5-2300 (2.8 gigahertz) and 4 gigabytes of RAM, running Linux OS. The ILS metaheuristics were reported to find optimal solutions in each instance, on a time limit of 15 minutes in a computer with

[1] https://www.utwente.nl/ctit/hstt/.

[2] http://labic.ic.uff.br/Instance/index.php?dir=SchoolTimetabling.

Table 2
Results of algorithm DIMB-19I1D-8-0.05 compared to results of the GOAL solver in the XHSTT-BV problem for the 34 instances described in Table A.4.

	CPLEX (after 3 hours)			DIMB-19I1D-8-0.05 (after 625 seconds)				GOAL (after 625 seconds)			
ID	LB	UB	Gap	Median	Gap	Best	Gap	Median	Gap	Best	Gap
1	682	810	15.80	710	3.94	**700**	2.57	730	6.58	716	4.75
2	682	802	14.96	711	4.08	**701**	2.71	730	6.58	716	4.75
3	717	926	22.57	749	4.27	**743**	3.50	770	6.88	762	5.91
4	716	928	22.84	743	3.63	**735**	2.59	768	6.77	749	4.41
5	629	648	2.93	**631**	0.32	**631**	0.32	648	2.93	642	2.02
6	755	945	20.11	777	2.83	**772**	2.20	814	7.25	800	5.63
7	1200	1376	12.79	1270	5.51	**1239**	3.15	1312	8.54	1297	7.48
8	675	**675**	0.00	698	3.30	686	1.60	682	1.03	676	0.15
9	799	865	7.63	827	3.39	**814**	1.84	838	4.65	829	3.62
10	298	300	0.67	**298**	0.00	**298**	0.00	300	0.67	**298**	0.00
11	943	1108	14.89	1005	6.17	**985**	4.26	1052	10.36	1024	7.91
12	1002	1078	7.05	1044	4.02	**1030**	2.72	1048	4.39	1034	3.09
13	273	**273**	0.00	**273**	0.00	**273**	0.00	**273**	0.00	**273**	0.00
14	923	1029	10.30	954	3.25	**939**	1.70	971	4.94	949	2.74
15						Infeasible					
16	475	523	9.18	484	1.86	**481**	1.25	498	4.62	490	3.06
17	454	499	9.02	461	1.52	**457**	0.66	468	2.99	462	1.73
18	310	325	4.62	**319**	2.82	**319**	2.82	324	4.32	**319**	2.82
19	251	**254**	1.18	**254**	1.18	**254**	1.18	**254**	1.18	**254**	1.18
20	310	349	11.17	**325**	4.62	**325**	4.62	328	5.49	**325**	4.62
21	1041	1149	9.40	1074	3.07	**1058**	1.61	1107	5.96	1091	4.58
22						Infeasible					
23	1791	2541	29.52	1910	6.23	**1867**	4.07	2013	11.03	1971	9.13
24	1941	2615	25.77	2079	6.64	**2038**	4.76	2171	10.59	2128	8.79
25	908	1086	16.39	944	3.81	**926**	1.94	987	8.00	966	6.00
26	570	583	2.23	576	1.04	**570**	0.00	603	5.47	584	2.40
27	551	569	3.16	556	0.90	**552**	0.18	566	2.65	560	1.61
28	1101	1363	19.22	1150	4.26	**1127**	2.31	1183	6.93	1167	5.66
29	1090	1371	20.50	1145	4.80	**1127**	3.28	1187	8.17	1168	6.68
30	1111	1418	21.65	1164	4.55	**1142**	2.71	1203	7.65	1175	5.45
31	1110	1371	19.04	1165	4.72	**1137**	2.37	1195	7.11	1181	6.01
32	996	1221	18.43	1043	4.51	**1024**	2.73	1069	6.83	1063	6.30
33	987	1308	24.54	1047	5.73	**1034**	4.55	1078	8.44	1060	6.89
34	975	1250	22.00	1041	6.34	**1015**	3.94	1070	8.88	1056	7.67
Avg.:			13.11		3.54		2.32		5.87		4.47

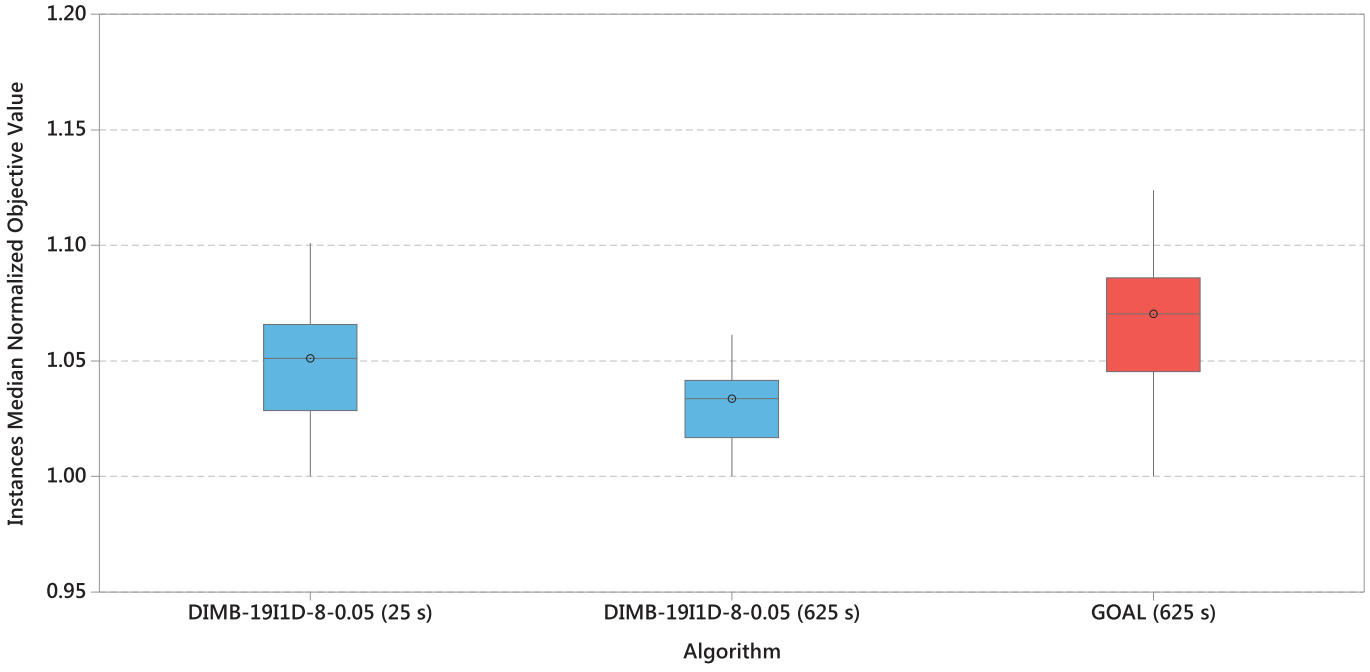


Fig. 8. Results of algorithm DIMB-19I1D-8-0.05 after 25 and 625 seconds compared to results of the GOAL solver after 625 seconds in the XHSTT-BV problem.

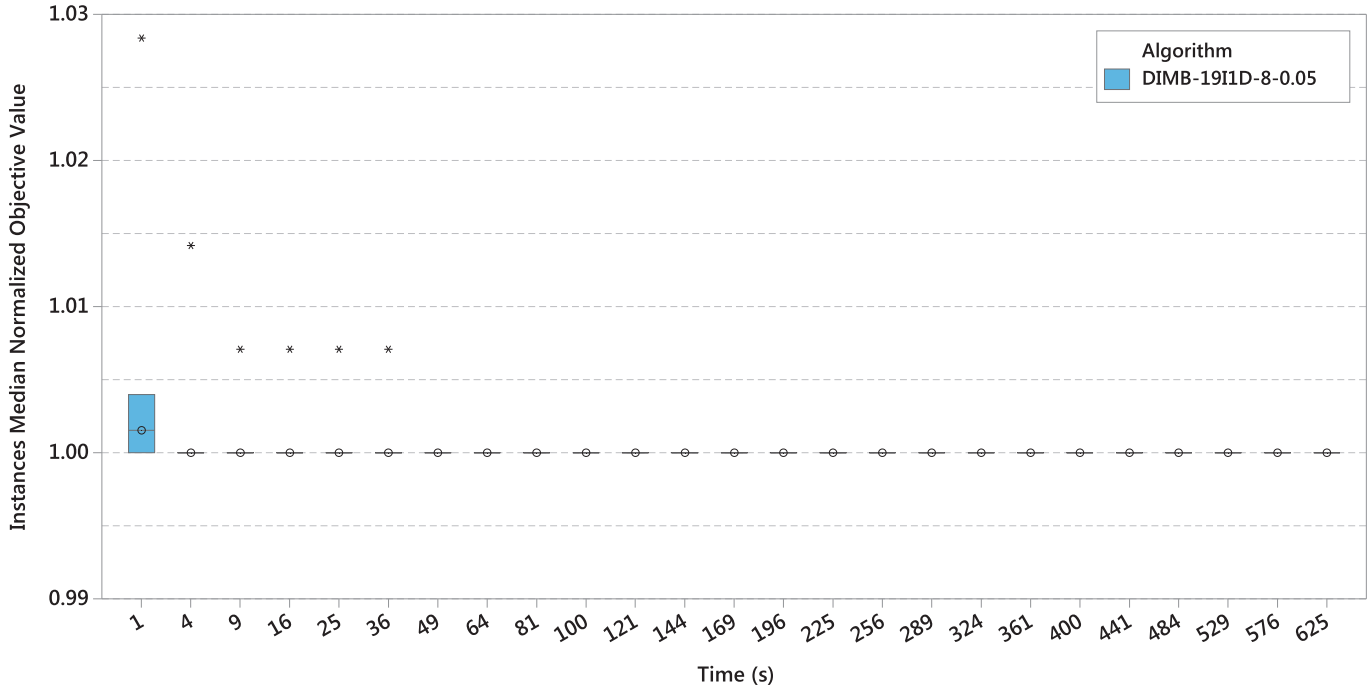


Fig. 9. Results of algorithm DIMB-19I1D-8-0.05 on instances proposed by Souza (2000).

Table 3
Features of instances proposed by Souza (2000). The last three columns show the total number of required lessons, the total number of required double lessons and the optimal objective values, respectively. The objective function weighting parameters are: $\alpha_3 = \alpha_4 = 100000$, $\alpha_5 = 10000$, $\alpha_7 = 1$, $\alpha_8 = 3$ and $\alpha_9 = 9$.

ID	$\|C\|$	$\|T\|$	$\|D\|$	$\|H\|$	$\Sigma_{c\in C, t\in T} RL_{ct}$	$\Sigma_{c\in C, t\in T} RDL_{ct}$	Optimal value
1	3	8	5	5	75	21	202
2	6	14	5	5	150	29	333
3	8	16	5	5	200	4	423
4	12	23	5	5	300	41	652
5	13	31	5	5	325	71	762
6	14	30	5	5	350	63	756
7	20	33	5	5	500	84	1017

processor Intel Xeon E7-4860 (2.26 gigahertz) and 30 gigabytes of RAM, running Windows OS.

In Fig. 9, we analyze the distribution of the instances' median samples generated by our parallel algorithm DIMB-19I1D-8-0.05. The instances optimal values described in Table 3 were used as benchmark values. We observe that, in the median case, the algorithm converges to optimal solutions in any instance within a time limit of 49 seconds. These results show that our algorithm is very robust to these instances and it is competitive with the state-of-the-art approaches.

6. Conclusions

In this paper, we conducted an extensive study with parallel trajectory-based metaheuristics for high school timetabling problems. Our study analyzed several aspects related to the design of these algorithms. In particular, we studied the homogeneity/heterogeneity of agents, the influence of intensification and diversification memories, and the frequency of information exchange among agents.

Our most efficient algorithm was obtained with the inclusion of a diversification memory, a very elitist central memory (only the best current solution was kept), short agent times which favored exchange of information between threads with frequent restarts and multiple copies of the same metaheuristic agent (an Iterated Local Search procedure). This configuration was able to consistently obtain good quality solutions for the problem at hand and also outperformed state-of-the-art algorithms for two variants of it. These variants ignored specific requirements or considered some of them as hard instead of soft constraints.

These results suggest that the proposed method is efficient and robust with respect to the enabling/disabling of constraints. This latter characteristic is particularly envisaged for HSTP algorithms since practical problems usually contain specific requirements. Our exploratory study limited itself to simulations with only four metaheuristic agents in which the threads had static execution times. Further research is aimed at exploring more flexible frameworks, in which other metaheuristics can be included and the agents' execution times can be self-calibrated according to the search history.

Acknowledgments

This research was supported by FAPESP-Brazil. Grants: 2013/13563-3 and 2015/10032-2. The authors would like to thank Professor George H. G. Fonseca for sharing the code of GOAL and also, the anonymous reviewers for their useful comments in our paper.

Appendix A. Description and tuning of the stand-alone metaheuristics

In this appendix, we describe our sequential metaheuristics for the HSTP. We discuss the data structure proposed to encode timetables, the neighborhood structure, the heuristic used to construct initial solutions and the objective function evaluation. The pseudo-code for each metaheuristic and tuning procedures are also presented.

A.1. Timetable encoding

Trajectory-based metaheuristics frequently modify copies of the current or global best solution. However, some algorithms do not

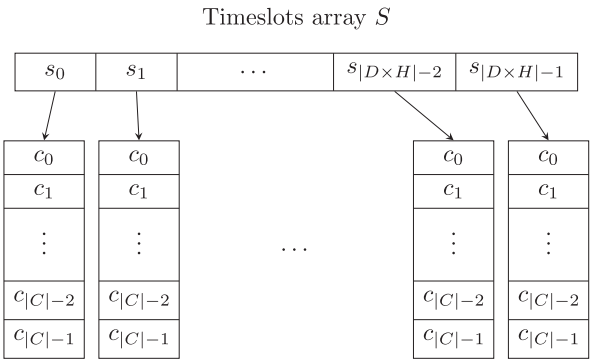


Fig. A.10. Proposed timetable encoding for the HSTP.

modify the whole copy, keeping parts that are identical to that of the previous solution. To avoid the duplication of unnecessary data (parts of the data structure that are copied but not modified), we designed a new timetable encoding to HSTP based on *persistent data structures* (Kaplan, 2005). In a persistent data structure, each modification generates a new version of the structure, keeping the previous version accessible. There are several consolidated methods to make a data structure persistent (Driscoll, Sarnak, Sleator, & Tarjan, 1989). One of these methods is based on *structural sharing*. The idea is to share with the new copy, parts of the original structure that remain unchanged.

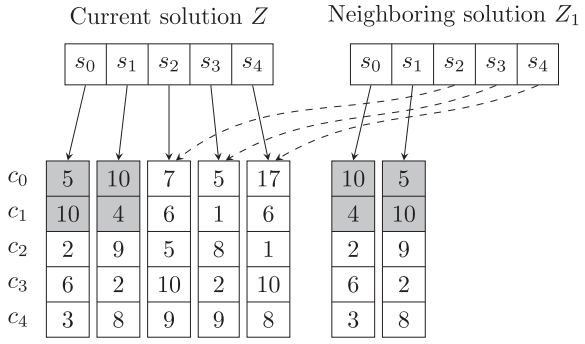
To apply this approach, we designed the encoding shown in Fig. A.10, in which S represents an array of all timeslots $(d, h) \in D \times H$. Each position of S points to an array of integer values. In each of these arrays, the first position stores the ID of the teacher assigned to class 0, the second position stores the teacher assigned to class 1, and so on. The next section explains how this timetable encoding is applied to generate the used neighborhoods.

A.2. Neighborhood structure

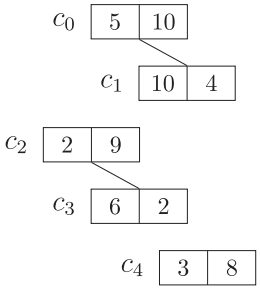
We use a two-swap operator which, for a given class $c \in C$, exchanges the teachers assigned to two different timeslots. Feasibility in hard requirements 1 (meeting of weekly required lessons), 2 (no clashes in classes' schedules) and 3 (no clashes in teachers' schedules) is maintained with the use of the Torque Neighborhood Operator (TQ) proposed in Saviniec et al. (2013). The TQ acts as a *Kempe Chain Interchange* (Lü, Hao, & Glover, 2011) that uses the idea of conflict graph and effects a series of consecutive swaps until there are no clashes in teachers nor classes' schedules.

Fig. A.11 shows examples of TQ moves with the encoding scheme proposed in Section A.1. The TQ operator is applied to a partial solution Z shown in Figure A.11(a). The conflict graph in Fig. A.11(b) identifies valid moves (connected components) in order to avoid clashes. The teachers assigned to classes c_0 and c_1 are swapped between timeslots s_0 and s_1. In the generated neighboring solution Z_1, timeslots s_0 and s_1 point to new arrays while remaining timeslots share the current arrays. Note that the arrays not changed in the move are shared between both solutions. Figs. A.11(c) and A.11(d) show that to operate or modify only timeslots s_0, s_1 and s_2, we do not need to copy the whole solution Z.

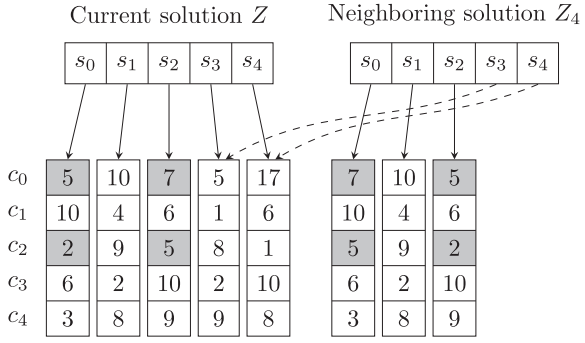
We employ this neighborhood generation scheme in all sequential and parallel metaheuristics proposed in this paper.



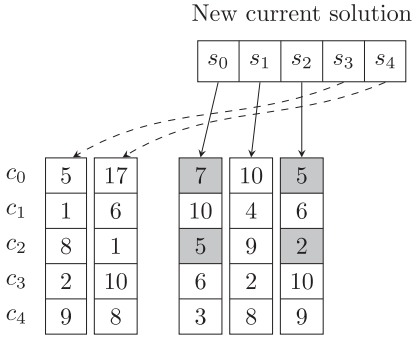
(a) Example of a TQ move applied to a partial solution Z showing the generated neighboring solution Z_1.



(b) Conflict graph for lessons assigned to timeslots s_0 and s_1 of solution Z, in Figure 11(a), showing three possible moves.



(c) Example of a fourth move, supposing that the three previous moves did not improve the current solution Z.



(d) The new current solution, supposing that the fourth move improved Z.

Fig. A.11. The TQ neighborhood generation with the proposed timetable encoding.

A.3. Initial solutions

Our initial solutions are constructed by the randomized heuristic described in Algorithm 1. This heuristic receives an input list L of pairs $(c \in C, t \in T)$, in which each entry represents one lesson to be scheduled. At each iteration, the heuristic selects a lesson from the list and assigns it to a timeslot that is randomly selected among the free timeslots. This heuristic always generates solutions that are feasible for hard requirements 1 and 2.

A.4. Objective function evaluation

Our metaheuristics are allowed to work in an infeasible space by relaxing constraints (3) and (4) and penalizing their violations in the objective function with high penalties to enforce feasibility. The objective function is augmented by

$$\textit{Minimize} \quad f(Z) = (21) + \sum_{i=3}^{4} \alpha_i \cdot V_i \tag{A.1}$$

Algorithm 1 Pseudo-code of the constructive heuristic.

SOLUTION-GENERATOR(L)

```
1  Initialize an empty solution Z
2  for each (e ∈ L) {
3      i = Take an integer in [0, |S| − 1]
4      while (class e.c is busy in timeslot s_i) {
5          i = (i + 1) mod |S|
6      }
7      Assign teacher e.t to class e.c in timeslot s_i
8  }
9  return Z
```

Algorithm 2 Pseudo-code of the sequential ILS.

ILS(Z_0, $timeOut$)

```
1  Z* = Z_0
2  while (elapsedTime < timeOut) {
3      Z = PERTURBATION(Z*)
4      do {
5          best = f(Z)
6          i = Take an integer in [0, |SP| − 1]
7          for (j = 1 to |SP|){
8              Compute the connected components for lessons
                 in timeslots pair sp_i of Z
9              for (each connected component) {
10                 Generate a new neighboring solution Z'
                     from Z
11                 if (f(Z') ≤ f(Z)) {
12                     Z = Z'
13                 }
14             }
15             i = (i + 1) mod |SP|
16         }
17     } while (f(Z) < best)
18     if (f(Z) ≤ f(Z*)) {
19         Z* = Z
20     }
21 }
22 return Z*
```

Algorithm 3 Pseudo-code of the sequential TS.

TS(Z_0, $timeOut$, $tbTime$, $stSize$)

```
1  Z = Z_0
2  Z* = Z_0
3  iter = 0
4  while (elapsedTime < timeOut) {
5      i = Take an integer in [0, |SP| − 1]
6      f_best = ∞
7      i_best = −1
8      Z_best = ∅
9      for (j = 1 to stSize){
10         Compute the connected components for lessons in
               timeslots pair sp_i of Z
11         for (each connected component) {
12             Generate a new neighboring solution Z' from Z
13             if (f(Z') ≤ f_best and
                   IsNotTabu(f(Z'), i, iter, tbTime)) {
14                 f_best = f(Z')
15                 i_best = i
16                 Z_best = Z'
17             }
18         }
19         i = (i + 1) mod |SP|
20     }
21     Insert the tuple (f(Z), i_best, iter) into the tabu list
22     if (f_best ≤ f(Z*)) {
23         Z* = Z_best
24     }
25     Z = Z_best
26     iter = iter + 1
27 }
28 return Z*
```

This expression is the objective function (21) plus the number of violations V_i, for hard requirements 3 and 4, weighted with penalty parameters α_i.

As the constructive heuristic (Algorithm 1) always generates feasible solutions with respect to hard requirements 1 and 2, and this is maintained by the neighborhood operator TQ, then only violations of hard requirements 3 and 4 are considered in the objective function.

In our implementations, we use incremental evaluation of solutions. The algorithms are designed to evaluate only the changed parts whenever a new move is performed.

A.5. Sequential metaheuristics for HSTP

Our sequential metaheuristics to the HSTP include versions of Iterated Local Search (ILS), Tabu Search (TS), Simulated Annealing (SA) and Late Acceptance Strategy (LAS).

We define SP as the array of all timeslot pairs (s_i, s_j), for $i, j = 0, \ldots, |S| - 1$ and $i \neq j$. Therefore, neighborhoods are defined by the way in which the timeslot pairs are explored in SP. Each of our algorithms may search this array in a different way.

A.5.1. Iterated local search

The pseudo-code of the implemented sequential iterated local search is shown in Algorithm 2. At each iteration of the outer loop (from line 2 to line 21), it makes a random perturbation (line 3) in the current solution, followed by a local search (loop from line 4 to line 17). The local search starts in a random index i of SP and evaluates the next $|SP|$ consecutive timeslot pairs. For each index,

Algorithm 4 Pseudo-code of the sequential SA.

SA($Z_0, timeOut, T_0, \alpha, stSize, \epsilon$)

```
1  Z = Z_0
2  Z* = Z_0
3  T = T_0
4  while (T ≥ ε and elapsedTime < timeOut) {
5      i = Take an integer in [0, |SP| − 1]
6      for (j = 1 to stSize){
7          Compute the connected components for lessons in
               timeslots pair sp_i
8          for (each connected component) {
9              Generate a new neighboring solution Z' from Z
10             Δ = f(Z) − f(Z')
11             if (Δ ≥ 0) {
12                 Z = Z'
13                 if (f(Z') < f(Z*)) {
14                     Z* = Z'
15                 }
16             } else {
17                 k = Take a continuous value in [0, 1]
18                 if (k ≤ e^{Δ/T}) {
19                     Z = Z'
20                 }
21             }
22         }
23         i = (i + 1) mod |SP|
24     }
25     T = α · T
26 }
27 return Z*
```

Algorithm 5 Pseudo-code of the sequential LAS.

LAS($Z_0, timeOut, lsSize, stSize$)

```
1  Z = Z_0
2  Z* = Z_0
3  for (j = 0 to lsSize − 1){
4      L_j = f(Z_0)
5  }
6  v = 0
7  while (elapsedTime < timeOut) {
8      i = Take an integer in [0, |SP| − 1]
9      for (j = 1 to stSize){
10         Compute the connected components for lessons in
               timeslots pair sp_i
11         for (each connected component) {
12             Generate a new neighboring solution Z' from Z
13             if (f(Z') ≤ f(Z) or f(Z') ≤ L_v) {
14                 Z = Z'
15                 if (f(Z') < L_v) {
16                     L_v = f(Z')
17                 }
18                 if (f(Z') < f(Z*)) {
19                     Z* = Z'
20                 }
21             }
22             v = (v + 1) mod lsSize
23         }
24         i = (i + 1) mod |SP|
25     }
26 }
27 return Z*
```

it generates the connected components according to what was discussed in Section A.2. If it finds a connected component that generates an improved solution, then the new solution is accepted. These steps are repeated while an improved solution is found after exploring $|SP|$ consecutive indexes[3]. The PERTURBATION procedure, in line 3, returns a random neighboring solution of the global best solution Z^*. This is equivalent to select a random index k of SP, construct the conflict graph for lessons assigned in timeslots sp_k of the solution Z^* and make a random TQ move with the associated connected components.

A.5.2. Tabu search

The pseudo-code of the implemented sequential tabu search is shown in Algorithm 3. At each iteration of the outer loop (from line 4 to 27), the algorithm searches the best neighbor of the current solution Z. The best neighbor is accepted if it is better or equal to the global best solution Z^*. The search starts in a random index i of SP, as in the ILS, and evaluates a strip of SP defined by the next $stSize$ consecutive timeslot pairs. Only moves that are not tabu are accepted. A tabu move is a tuple (f, $index$, $iter$), where f is the objective function value before the move, $index$ is the index of SP in which the move was made, and $iter$ is the iteration in which the move was made. The parameter $tbTime$ is the number of iterations in which a move remains tabu (the aspiration criterion). The procedure ISNOTTABU (line 13) checks if a move is tabu and also, removes those moves in which the tabu time has expired.

[3] The algorithm iterates through the indexes of SP in a circular fashion as in a circular linked list.

A.5.3. Simulated annealing

The pseudo-code of the implemented sequential simulated annealing is shown in Algorithm 4. The code follows a standard SA framework. However, instead of selecting each new neighboring solution randomly, as in standard SA, the algorithm chooses a random index of SP, as a starting point, and searches the next $stSize$ consecutive timeslot pairs. The outer loop (from line 4 to 26) controls the temperature level and stops after a time-out is reached or the temperature T reaches a small value given by parameter ϵ. The remaining steps are those from a standard SA framework. Parameters T_0 and α are the initial temperature and the cooling rate, respectively.

A.5.4. Late acceptance strategy

The last sequential metaheuristic proposed is a late acceptance strategy, for which a pseudo-code is presented in Algorithm 5. The LAS is a recent metaheuristic approach (Burke & Bykov, 2008) that maintains a list L of size $lsSize$ which records the objective values of the last $lsSize$ accepted solutions. The main idea of the LAS is to compare the current solution with a previous current solution recorded in L (Burke & Bykov, 2008). In our version of the LAS, the neighborhood is searched by strips, as in our versions of TS and SA described above.

A.6. Parameter tuning

In this section, we report the computational experiments carried out to calibrate the sequential metaheuristics described in Section A.5.

Table A.4 presents a list of 34 instances to the HSTP described in Section 2. These instances were collected in 13 Brazilian high

Table A.4
Features of the 34 real instances from Brazilian high schools. The columns present the number of classes, teachers, days, periods per day, total number of required lessons and total number of consecutive double lessons required in each instance.

ID	Instance	$\|C\|$	$\|T\|$	$\|D\|$	$\|H\|$	$\Sigma_{c\in C, t\in T} RL_{ct}$	$\Sigma_{c\in C, t\in T} RDL_{ct}$
1	CL-CEASD-2008-V-A	12	27	5	5	300	132
2	CL-CEASD-2008-V-B	12	27	5	5	300	132
3	CL-CECL-2011-M-A	13	31	5	5	325	144
4	CL-CECL-2011-M-B	13	31	5	5	325	143
5	CL-CECL-2011-N-A	9	28	5	5	225	107
6	CL-CECL-2011-V-A	14	29	5	5	350	164
7	CM-CECM-2011-M	20	51	5	5	500	234
8	CM-CECM-2011-N	8	30	5	5	200	96
9	CM-CECM-2011-V	13	34	5	5	325	142
10	CM-CEDB-2010-N	5	17	5	5	125	60
11	CM-CEUP-2008-V	16	35	5	5	400	192
12	CM-CEUP-2011-M	16	38	5	5	400	192
13	CM-CEUP-2011-N	3	15	5	5	75	36
14	CM-CEUP-2011-V	16	34	5	5	400	169
15	FA-EEF-2011-M	4	12	5	5	100	42
16	JNS-CEDPII-2011-M	8	19	5	5	200	85
17	JNS-CEDPII-2011-V	7	21	5	5	175	73
18	JNS-CEJXXIII-2011-M	5	18	5	5	125	60
19	JNS-CEJXXIII-2011-N	4	15	5	5	100	48
20	JNS-CEJXXIII-2011-V	5	18	5	5	125	60
21	MGA-CEDC-2011-M	19	37	5	5	475	210
22	MGA-CEDC-2011-V	12	31	5	5	300	131
23	MGA-CEGV-2011-M	31	62	5	5	775	352
24	MGA-CEGV-2011-V	32	75	5	5	800	357
25	MGA-CEJXXIII-2010-V	16	35	5	5	400	192
26	MGA-CEVB-2011-M	10	21	5	5	250	108
27	MGA-CEVB-2011-V	9	20	5	5	225	97
28	NE-CESVP-2011-M-A	18	45	5	5	450	212
29	NE-CESVP-2011-M-B	18	44	5	5	450	212
30	NE-CESVP-2011-M-C	18	45	5	5	450	211
31	NE-CESVP-2011-M-D	18	45	5	5	450	211
32	NE-CESVP-2011-V-A	16	44	5	5	400	183
33	NE-CESVP-2011-V-B	16	43	5	5	400	184
34	NE-CESVP-2011-V-C	16	43	5	5	400	182

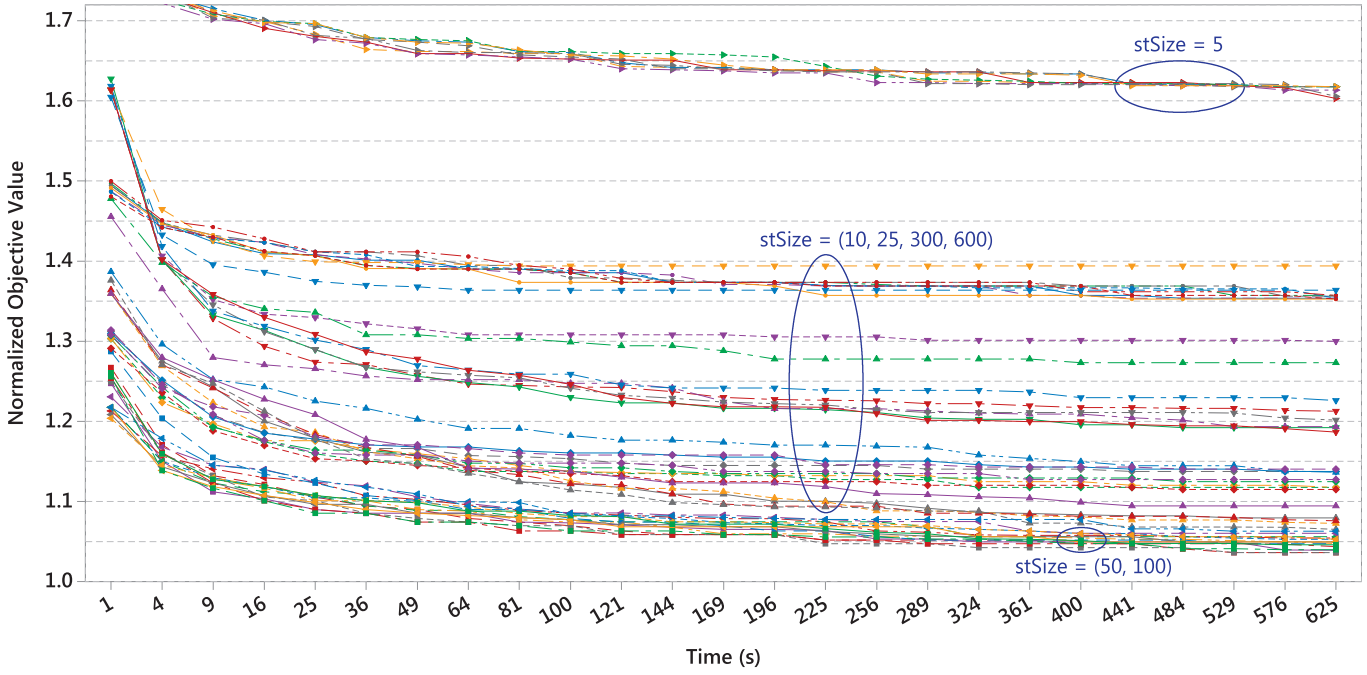


Fig. A.12. Results of sequential TS for different parameter configurations.

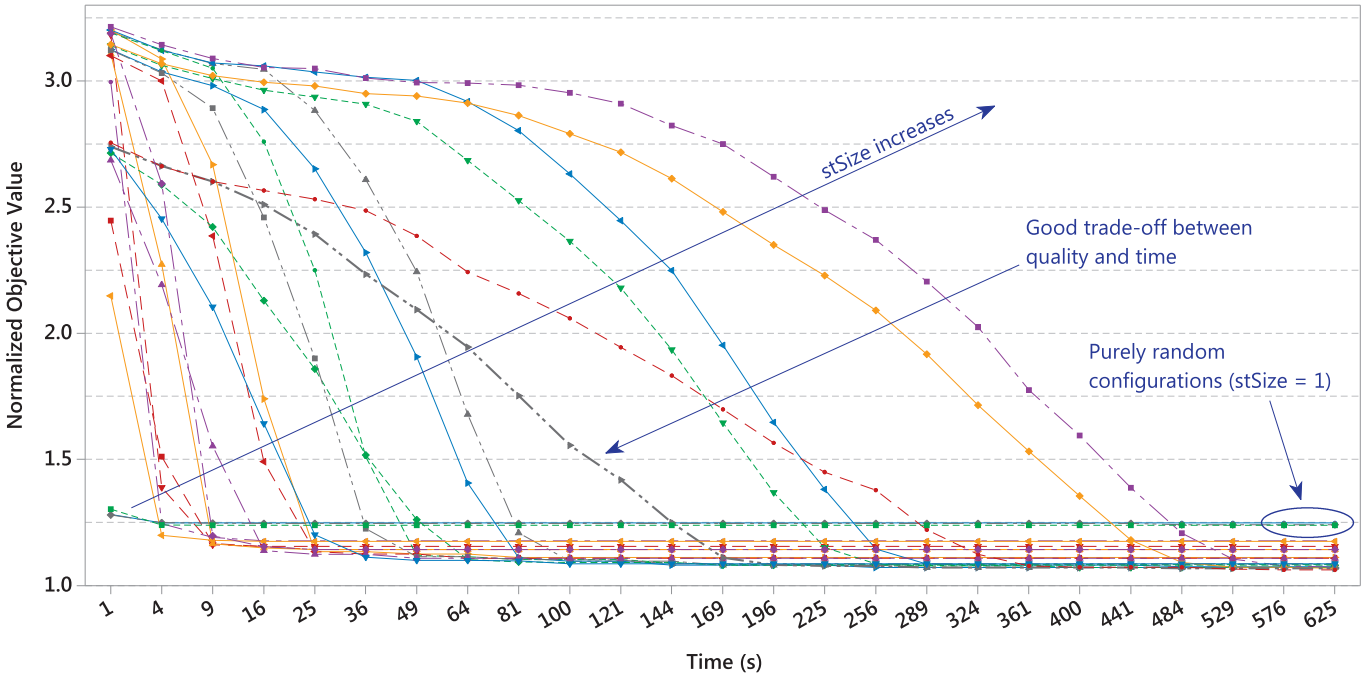


Fig. A.13. Results of sequential SA for different parameter configurations.

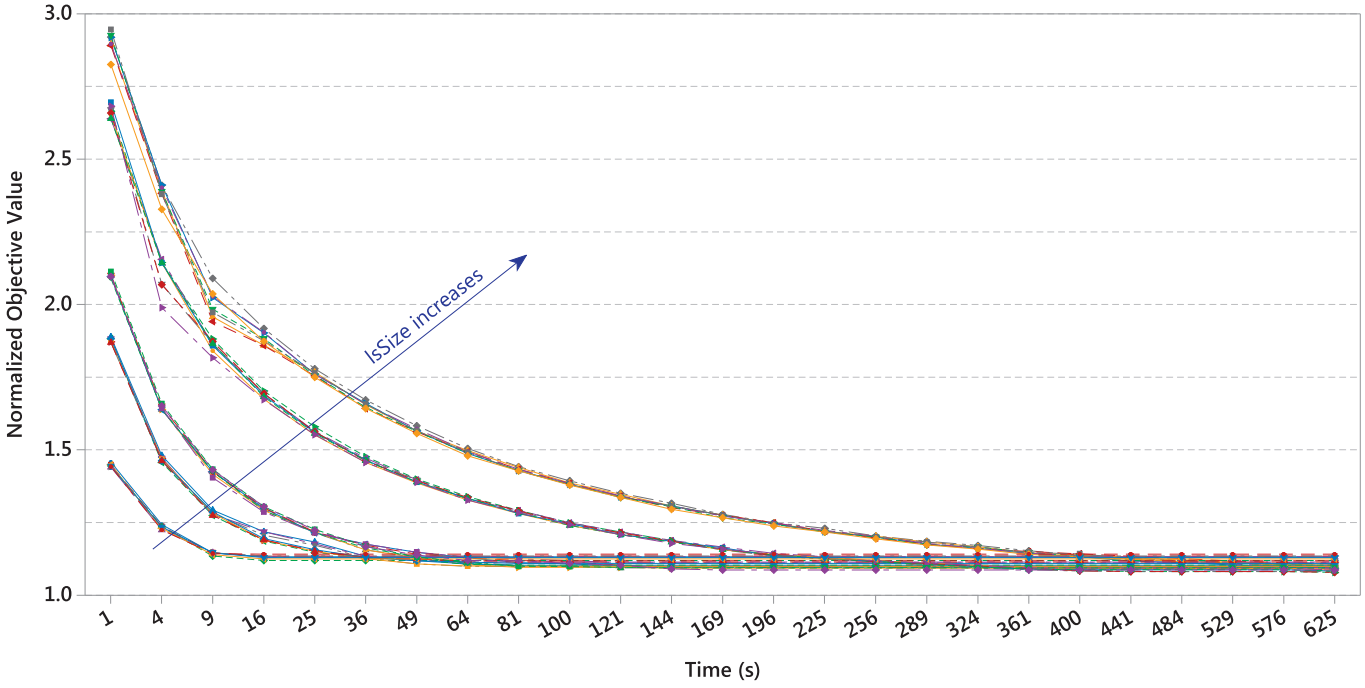


Fig. A.14. Results of sequential LAS for different parameter configurations.

schools in the State of Paraná (Saviniec & Constantino, 2017). The following weighting parameters are used in the objective function for these instances: $\alpha_3 = \alpha_4 = 100000, \alpha_5 = 100, \alpha_6 = 25$ and $\alpha_7 = \alpha_8 = \alpha_9 = \alpha_{10} = 10$. To these experiments, we select a subset of the 34 instances to analyze and tune the sequential metaheuristics. The reduced set of instances comprises: two small (10 and 13), two medium (6 and 26) and two large (7 and 24) instances.

Figs. A.12–A.15 present charts of normalized objective values over time for different algorithm configurations. The x-axis (time in seconds) is in quadratic scale. Objective values were collected for each time $x = i^2$ seconds, for $i = 1, ..., 25$. The y-axis are normalized objective values obtained as follows. Let Z_j be a timetable solution for the j-th instance, then its normalized objective value is given by $NOV(Z_j) = f(Z_j)/f(Z_j^*)$. Where $f(Z_j^*)$ is a benchmark objective value to instance j, which can be a lower bound or the best-known solution. As at each point x, each metaheuristic collects 25 samples for each instance, the final y-value in a point x is the median of the instances' median sample. In these charts, the reference line $(y = 1)$ represents the benchmark objective value $f(Z_j^*)$. In other words, the charts summarize the results for all instances and

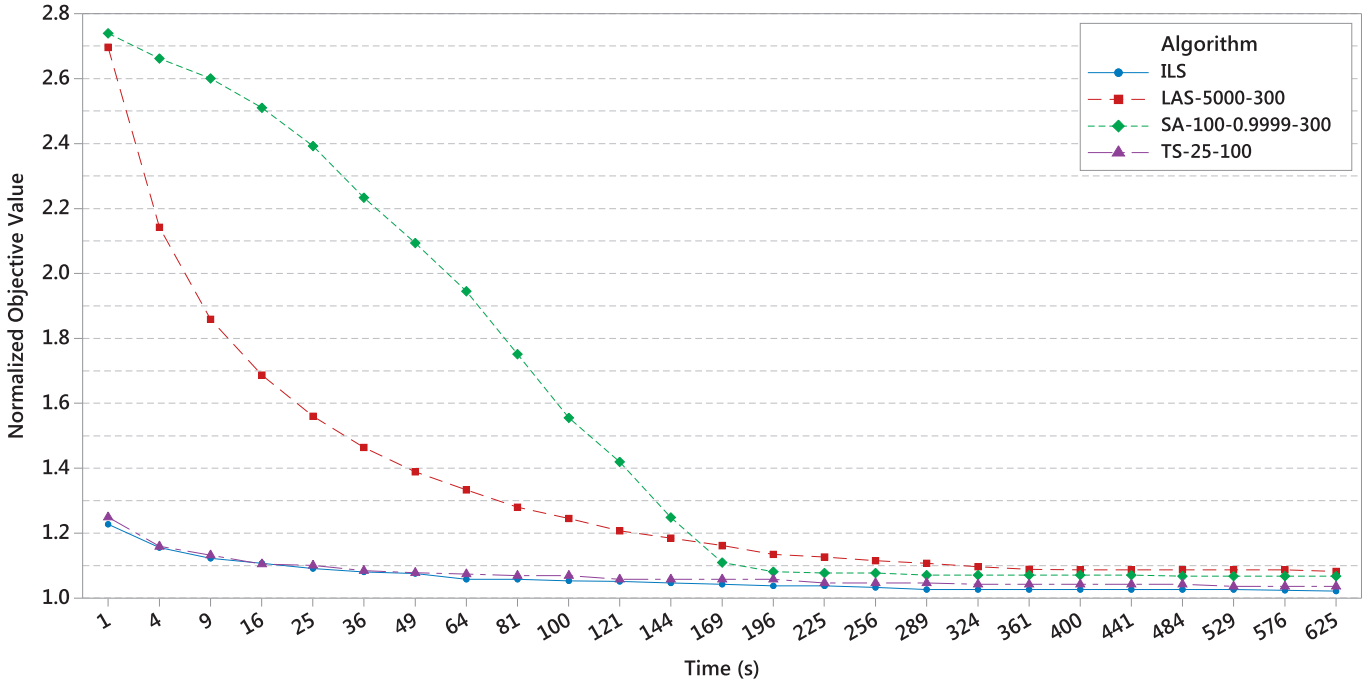


Fig. A.15. Best configurations for sequential metaheuristics.

indicate, for each time point, how close or far the algorithm median solutions are from the benchmark value, providing insights into the algorithm convergence rates.

A.6.1. Results

We tested a number of different values to calibrate parameters of TS, SA and LAS. The following values were tested:

- TS: *tbTime* = (5, 10, 25, 50, 100, 200, 400, 600) and *stSize* = (5, 10, 25, 50, 100, 300, 600).
- SA: T_0 = (100, 500, 1000), α = (0.99, 0.9999), *stSize* = (1, 5, 10, 25, 50, 100, 300, 600) and $\epsilon = 0$.
- LAS: *lsSize* = (100, 500, 1000, 5000, 10000) and *stSize* = (1, 5, 10, 25, 50, 100, 300, 600).

We analyze the behavior presented by different combinations of the above parameter values. Fig. A.12 shows the TS configurations obtained. The results indicate that the TS is very sensitive to the size of the neighborhood explored (parameter *stSize*). Small and large values for *stSize* lead the TS to converge to poor quality solutions. An appropriated value is around 100. Also, the best values for the tabu aspiration criterion (parameter *tbTime*) were observed to be between 10 and 100. These results suggest that the TS metaheuristic must be carefully calibrated with respect to neighborhood sizes.

Fig. A.13 shows the behavior of the SA for different configurations. The algorithm seems to be most sensitive to the cooling rate parameter. A good choice was $\alpha = 0.9999$. We observed that even small negative perturbations in this value could worsen convergence rates. Also, we observed that when the SA tended to be purely random, with small values for parameter *stSize*, the SA converged to poor quality solutions. On the other hand, when *stSize* tended to large values, the SA converged to better solutions, although with a slower convergence rate. In Fig. A.13 we plotted only configurations for $\alpha = 0.9999$. Configurations for $\alpha = 0.99$ performed as badly as purely random configurations.

Fig. A.14 shows the LAS configurations. The results show that LAS is not sensitive to the way in which the neighborhood is explored, parameter *stSize*. On the other hand, its convergence strongly depends on the length of the list of late accepted solutions, parameter *lsSize*. Large size lists lead the algorithm to converge to better quality solutions, however, the convergence is slow.

Fig. A.15 plots the best observed configurations for TS with (*tbTime*, *stSize*) = (25, 100), SA with (T_0, α, *stSize*) = (100, 0.9999, 300) and LAS with (*lsSize*, *stSize*) = (5000, 300) together with the ILS results. The benchmark value for an instance j is the objective value $f(Z_j^*)$ of the best solution Z_j^* found in the experiments with these sequential algorithms. The chart in Fig. A.15 shows that ILS and TS presented better results than SA and LAS, in general. The median objective values of the four algorithms after 625 seconds of execution are 2.2% (ILS), 3.6% (TS), 6.8% (SA) and 8.3% (LAS) worse than the benchmark values. While SA and LAS take 625 seconds to reach solutions that are around 7 to 8% worse than the benchmark values, ILS and TS compute solutions with the same quality in less than 80 seconds. Also, we note that ILS perform slightly better in quality than TS.

References

Abramson, D. (1991). Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management science, 37*(1), 98–113.

Abramson, D., & Abela, J. (1992). A parallel genetic algorithm for solving the school timetabling problem. In *Proceedings of the 15 Australian computer science conference* (pp. 1–11). Hobart, Australia.

Al-Yakoob, S. M., & Sherali, H. D. (2015). Mathematical models and algorithms for a high school timetabling problem. *Computers & Operations Research, 61*, 56–68. http://dx.doi.org/10.1016/j.cor.2015.02.011.

Alba, E., Luque, G., & Nesmachnow, S. (2013). Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research, 20*(1), 1–48.

Bożejko, W., Pempera, J., & Smutnicki, C. (2013). Parallel tabu search algorithm for the hybrid flow shop problem. *Computers & Industrial Engineering, 65*(3), 466–474.

Burget, R., & Rudová, H. (2016). Teacher-oriented fairness in course timetabling. In *Proceedings of the 11th international conference on the practice and theory of automated timetabling (patat 2016)* (pp. 33–44). Udine, Italy.

Burke, E. K., & Bykov, Y. (2008). A late acceptance strategy in hill-climbing for exam timetabling problems. In *Proceedings of the 7th conference on the practice and theory of automated timetabling, montreal, canada*.

Crainic, T. G., Gendreau, M., Hansen, P., & Mladenović, N. (2004). Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics, 10*(3), 293–314. doi:10.1023/B:HEUR.0000026897.40171.1a.

De Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research, 19*(2), 151–162.
Di Gaspero, L., McCollum, B., & Schaerf, A. (2007). The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). *Technical Report*. Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0, Queen's UniversityBelfast, United Kingdom.
Dorneles, Á. P., de Araújo, O. C., & Buriol, L. S. (2014). A fix-and-optimize heuristic for the high school timetabling problem. *Computers & Operations Research, 52, Part A*(0), 29–38.
Dorneles, Á. P., de Araújo, O. C., & Buriol, L. S. (2017). A column generation approach to high school timetabling modeled as a multicommodity flow problem. *European Journal of Operational Research, 256*(3), 685–695.
Driscoll, J. R., Sarnak, N., Sleator, D. D., & Tarjan, R. E. (1989). Making data structures persistent. *Journal of Computer and System Sciences, 38*(1), 86–124. doi:10.1016/0022-0000(89)90034-2.
Even, S., Itai, A., & Shamir, A. (1975). On the complexity of timetable and multi–commodity flow problems. In *16th annual symposium on foundations of computer science* (pp. 184–193). IEEE.
Fonseca, G. H., & Santos, H. G. (2014). Variable neighborhood search based algorithms for high school timetabling. *Computers & Operations Research, 52*, 203–208.
Fonseca, G. H., Santos, H. G., & Carrano, E. G. (2016a). Late acceptance hill-climbing for high school timetabling. *Journal of Scheduling, 19*, 453–465.
Fonseca, G. H., Santos, H. G., & Carrano, E. G. (2016b). Integrating matheuristics and metaheuristics for timetabling. *Computers & Operations Research, 74*, 108–117.
Fonseca, G. H., Santos, H. G., Toffolo, T. A., Brito, S. S., & Souza, M. J. (2016c). Goal solver: a hybrid local search based solver for high school timetabling. *Annals of Operations Research, 239*, 77–97.
Glover, F. (1990). Tabu search: A tutorial. *INTERFACES, 20*(4), 74–94.
Jin, J., Crainic, T. G., & Løkketangen, A. (2014). A cooperative parallel metaheuristic for the capacitated vehicle routing problem. *Computers & Operations Research, 44*, 33–41. http://dx.doi.org/10.1016/j.cor.2013.10.004.
Kaplan, H. (2005). Persistent data structures. In D. P. Mehta, & S. Sahni (Eds.), *Handbook of data structures and applications*. In *Chapman & Hall/CRC {series} computer and information science series*. Chapman & Hall/CRC.
Kingston, J. H. (2015). A software library for high school timetabling. http://sydney.edu.au/engineering/it/~jeff/khe/.
Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.
Kristiansen, S., Sørensen, M., & Stidsen, T. R. (2015). Integer programming for the generalized high school timetabling problem. *Journal of Scheduling, 18*(4), 377–392. http://dx.doi.org/10.1007/s10951-014-0405-x.
Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum, 30*(1), 167–190.
Lewis, R., Paechter, B., & McCollum, B. (2007). *Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition*. Cardiff Business School.
Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. *Handbook of metaheuristics* (pp. 320–353). Boston, MA, US: Springer.
Lü, Z., Hao, J., & Glover, F. (2011). Neighborhood analysis: A case study on curriculum-based course timetabling. *Journal of Heuristics, 17*(2), 97–118.
Luque, G., & Alba, E. (2015). Parallel hybrid trajectory based metaheuristics for real–world problems. In *Proceedings of international conference on intelligent networking and collaborative systems* (pp. 184–191). IEEE.
McCollum, B., McMullan, P., Burke, E. K., Parkes, A. J., & Qu, R. (2007). The second international timetabling competition: Examination timetabling track. *Technical Report*. Technical Report QUB/IEEE/Tech/ITC2007/-Exam/v4.0/17, Queen's UniversityBelfast, United Kingdom.
Pillay, N. (2014). A survey of school timetabling research. *Annals of Operations Research, 218*, 261–293.
Post, G., Ahmadi, S., Daskalaki, S., Kingston, J., Kyngas, J., Nurmi, C., & Ranson, D. (2012). An xml format for benchmarks in high school timetabling. *Annals of Operations Research, 194*(1), 385–397.
Post, G., Di Gaspero, L., Kingston, J. H., McCollum, B., & Schaerf, A. (2016). The third international timetabling competition. *Annals of Operations Research, 239*(1), 69–75.
Qu, R., Burke, E. K., McCollum, B., Merlot, L. T., & Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of scheduling, 12*(1), 55–89.
Sánchez-Oro, J., Sevaux, M., Rossi, A., Martí, R., & Duarte, A. (2015). Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies. *Electronic Notes in Discrete Mathematics, 47*, 85–92.
Santos, H., Ochi, L., & Souza, M. (2005). A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem. *Journal of Experimental Algorithmics, 10*, 2–9.
Santos, H., Uchoa, E., Ochi, L., & Maculan, N. (2012). Strong bounds with cut and column generation for class-teacher timetabling. *Annals of Operations Research, 194*(1), 399–412.
Saviniec, L., & Constantino, A. A. (2017). Effective local search algorithms for high school timetabling problems. *Applied Soft Computing, 60*, 363–373. doi:10.1016/j.asoc.2017.06.047.
Saviniec, L., Constantino, A. A., Romão, W., & Santos, H. G. (2013). Solving the high school timetabling problem to optimality by using ILS algorithms. In *Proceedings of Brazilian symposium on operations research, Sobrapo, Rio de Janeiro, Brazil* (pp. 3330–3341).
Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review, 13*(2), 87–127.
Sørensen, M., & Dahms, F. H. (2014). A two-stage decomposition of high school timetabling applied to cases in denmark. *Computers & Operations Research, 43*, 36–49. http://dx.doi.org/10.1016/j.cor.2013.08.025.
Souza, M., Ochi, L., & Maculan, N. (2003). A Grasp–Tabu search algorithm for solving school timetabling problems. In *Metaheuristics: Computer decision-making* (pp. 659–672). Boston: Kluwer Academic Publishers.
Souza, M. J. F. (2000). *Programação de Horários em Escolas: Uma Aproximação por Metaheurísticas*. Rio de Janeiro: Universidade Federal do Rio de Janeiro Programa de Pós-Graduação em Engenharia de Sistemas e Computação (Ph.D. thesis).
Srndic, N., Pandzo, E., Dervisevic, M., & Konjicija, S. (2009). The application of a parallel genetic algorithm to timetabling of elementary school classes: A coarse grained approach. In *Proceedings of 12th international symposium on Information, communication and automation technologies, ICAT 2009.* (pp. 1–5). IEEE.
Subramanian, A., Drummond, L. M. A., Bentes, C., Ochi, L. S., & Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research, 37*, 1899–1911.