Integrating Matheuristics and Metaheuristics for Timetabling

George H. G. Fonseca[a,b], Haroldo G. Santos[c], Eduardo G. Carrano[a,d]

[a] *Graduate Program in Electrical Engineering*
Universidade Federal de Minas Gerais
Av. Antônio Carlos 6627, 31270-901
Belo Horizonte, MG, Brazil
[b] *Department of Information Systems and Computing*
Universidade Federal de Ouro Preto
St. Diogo de Vasconcelos 328, 35400-000
Ouro Preto, MG, Brazil
[c] *Department of Computing*
Universidade Federal de Ouro Preto
St. Diogo de Vasconcelos 328, 35400-000
Ouro Preto, MG, Brazil
[d] *Department of Electrical Engineering*
Universidade Federal de Minas Gerais
Av. Antônio Carlos 6627, 31270-901
Belo Horizonte, MG, Brazil

Abstract

The High School Timetabling Problem requires the assignment of times and resources to events, while sets of required and desirable constraints must be considered. The most common approach for this problem is to employ metaheuristic methods. This work presents a matheuristic approach that combines a Variable Neighbourhood Search algorithm with mathematical programming-based neighbourhoods for high school timetabling. Computational experiments on well-known benchmark instances demonstrate the success of the proposed hybrid approach, which outperforms the standalone Variable Neighbourhood Search algorithm by far. Additionally, the proposed algorithm was able to improve 15 out of 17 current best known solutions in

Email addresses: george@decsi.ufop.br (George H. G. Fonseca), haroldo@iceb.ufop.br (Haroldo G. Santos), egcarrano@ufmg.br (Eduardo G. Carrano)

a very famous benchmark set.

Keywords: Matheuristics, Metaheuristics, Timetabling

1. Introduction

The High School Timetabling Problem (HST) consists in assigning times and resources to events respecting several hard and soft constraints. Some usual constraints are to respect the availability of teachers, to respect the limit on lessons of the same class in a day, and to avoid idle times between activities. The schedule, which is generally made for a week, is repeated until the end of the class period. Although it is usually referred as HST, the problem can be defined in such a way that it can be extended to other levels, such as university courses. Beyond its practical importance, this problem is $\mathcal{NP}$-Hard [1], which makes it of interest for Operations Research and Artificial Intelligence communities.

The Third International Timetabling Competition (ITC2011) [2] motivated the development of several approaches to solve this problem. The competition considered the eXtended Markup Language for High School TimeTabling (XHSTT) format [3], in which several features related to scheduling problems can be properly defined. Nowadays, more than 40 real world instances, from 12 different countries, are available in this format. They can be used to evaluate the performance of algorithms for high school/university timetabling.

In the ITC2011 competition, metaheuristic approaches achieved remarkable results. The four finalists employed metaheuristics as the main solver or part of it: GOAL solver [4], the winner of the competition, was a combination of Simulated Annealing (SA) and Iterated Local Search (ILS); Lectio, ranked second, employed an Adaptive Large Neighbourhood Search (ALNS) [5]; HySTT, ranked third, proposed a Hyper-heuristic approach [6]; finally HFT, ranked fourth, was based on an Evolutionary Algorithm [7].

More recently, the GOAL team released a new solver based on the Variable Neighbourhood Search algorithm [8], Kingston developed a solver based on his library for handling XHSTT instances [9], and the HySTT team worked towards an improved version of their Hyper-heuristic approach [10]. In a different direction, Kristiansen *et al.* [11] proposed the first integer programming formulation for XHSTT timetabling problems.

The integration of metaheuristics and mathematical programming approaches, namely matheuristics, is a growing field in operations research. For example, some recent works in [12, 13, 14] proposed matheuristics for vehicle routing, flow shop scheduling, and nurse rostering respectively. Sorensen and Stidsen [15] presented some preliminary results in matheuristics for XHSTT timetabling problems. However, no problem-specific neighbourhood was presented in their work. In this paper, a hybrid approach is proposed, which applies Variable Neighborhood Search (VNS) at the beginning of the search and invokes a matheuristic algorithm after VNS stagnation. The intention behind this approach is to find an interesting balance between exploration and exploitation, in order to reach better solutions.

The remainder of the paper is organized as follows. The Generalized High School Timetabling problem and the XHSTT format are described in Section 2. The proposed algorithm is presented in Section 3. Computational experiments and results achieved are reported in Section 4. Finally, some concluding remarks are drawn in Section 5.

2. Generalized Timetabling Problem

In general, the educational timetabling problem consists in assigning times and resources (teachers, classes and rooms) to events (lectures), while respecting several constraints given a priori. Examples of usual constraints are: (*i*) do not allow a resource to attend more than one event at the same time; (*ii*) respect the resources unavailable times, or; (*iii*) split the events into sub-events of valid size. Figure 1 presents an example of timetabling for the resource "5th grade class", whose events (lectures) are distributed along 25 different times (time periods). Assuming that an active constraint demands that the last time of each day should be empty, it is possible to identify a violation at Wednesday (event "Phis"). The complete timetable solution is the set of all resource assignments.

There are several problem categories in the context of educational timetabling. The most recurrent classifications are High School Timetabling [16], University Course Timetabling [17] and Student Sectioning [18]. Moreover, significant differences can be noted in the timetabling requirements of different educational institutions, specially from different countries. In practice, this variety makes it hard to apply an existing solver to a new timetabling instance – usually it is necessary to hand-code a new solver to the instance, taking into account its specific requirements. As a consequence, it is often

5th grade class

Mon	Tue	Wed	Thu	Fri
Eng_1	$Span_3$	$Math_1$	$Chem_1$	$Math_3$
Eng_2	$Span_4$	$Math_2$	$Chem_2$	
$Span_1$	Eng_3	Bio_1	$Info_1$	$Math_4$
$Span_2$	Eng_4	Bio_2	$Info_2$	$Phis_2$
		$Phis_1$		

Avoid unavailable times violation:
Event Phis is assigned to the last time of Wednesday.

Figure 1: Example of timetable for "5th grade class".

complicated to compare different solution approaches. The XHSTT format was proposed for handling such difficulties. This format defines a generic way to create resources and events, and it allows the specification of 16 different constraint types, which may be hard, soft or not applicable, depending on the instance requirements. Any timetabling problem that can be specified using these 16 constraints is suitable to be solved with any XHSTT solver. This flexibility justifies the choice for this format in this work. A brief description of the format is given in the following subsection.

2.1. XHSTT Format

A XHSTT instance is composed of four entities:

Times: contains information about the *times* available for allocation. These *times* may also be grouped into *TimeGroups*.

Resources: contains the resources available for assignment. Each resource has a specific *ResourceType*. Resources can be also grouped into *ResourceGroups*.

Events: represents the events to be scheduled. Each event has a duration (number of *times* to be occupied) and it demands a set of resources. Eventually, times and resources may be pre-assigned to events. If these entities are not pre-assigned, then the solver should be responsible to make such an assignment. Events also have a workload demand, which must be considered by its assigned resources. They are also commonly grouped into *EventGroups*.

Constraints: represents the set of constraints that should be satisfied in a solution for an instance of this problem. Table 1 presents the 16 constraint types available in XHSTT format. Each constraint may be set as hard or soft. The infringement of a hard constraint implies in infeasibility, while the soft constraints measure the quality of feasible solutions (smaller values indicate better solutions). Each constraint also has a cost, which expresses the penalty for a single violation, and a cost function, which defines how violations are penalized in the objective function. Detailed description of this format can be found in Post *et al.* [3] and Kingston [19].

Table 1: Different constraint types in the XHSTT format [20].

Constraint	Description
Assign Resource	Event should be assigned a resource
Assign Time	Event should be assigned a time
Split Events	Event should split into a constrained number of sub-events
Distribute Split Events	Event should split into sub-events of constrained durations
Prefer Resources	Event resource assignment should come from resource group
Prefer Times	Event time assignment should come from time group
Avoid Split Assignments	Set of events should be assigned the same resource
Spread Events	Set of events should be spread evenly through the cycle
Link Events	Set of events should be assigned the same time
Order Events	Set of events should be ordered
Avoid Clashes	Resource's timetable should not have clashes
Avoid Unavailable Times	Resource should not be busy at unavailable times
Limit Idle Times	Resource's timetable should not have idle times
Cluster Busy Times	Resource should be busy on a limited number of days
Limit Busy Times	Resource should be busy a limited number of times each day
Limit Workload	Resource's total workload should be limited

3. Hybrid Algorithm

The proposed hybrid algorithm is composed of three sequential steps: (*i*) the KHE solver [9, 19] is employed to generate an initial solution; (*ii*) a VNS algorithm, with six neighbourhood structures, is employed to improve this solution as much as possible, until stagnation, and; (*iii*) a matheuristic

method is employed to provide fine improvement of the current solution until timeout condition is reached. It is considered that VNS stagnates when it reaches one tenth of the available time without obtaining any improvement. A basic scheme of the proposed approach can be seen in Figure 2. The main parts of the method are discussed in the next sections.

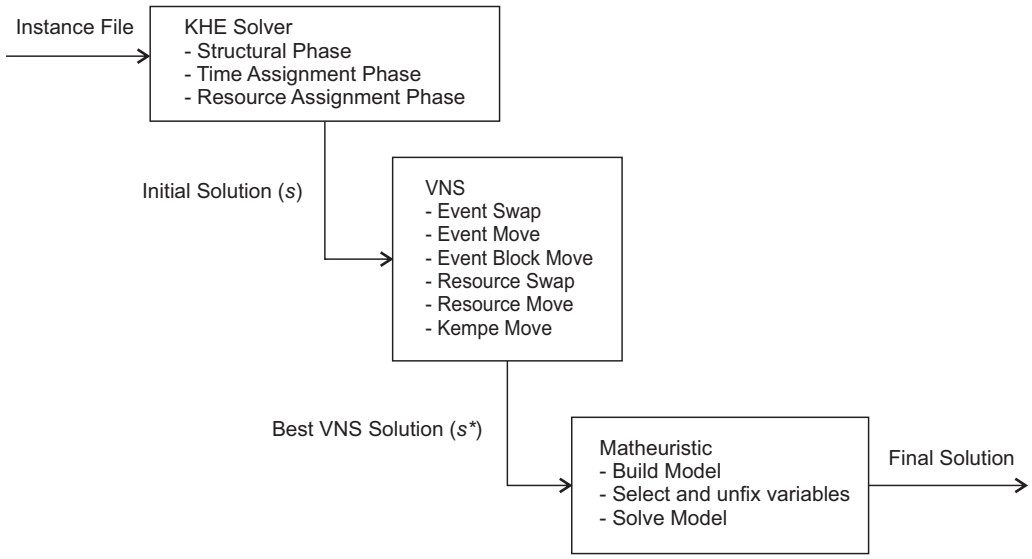

Figure 2: Scheme of the proposed approach.

3.1. KHE Solver

The Kingston High School Timetabling Engine is a platform for handling instances of HST. It also provides a solver, which is used in this work to generate initial solutions since it can find acceptable assignments in short time [19]. KHE generates a solution through three steps: structural phase, time assignment phase and resource assignment phase. For sake of brevity, only a brief description of the method is provided below. For additional information, please refer to [9, 19].

At the beginning of the *structural phase*, an initial solution is built with no times or resources assigned. In this step, events are split into sub-events, whose durations depend on split event constraints, and the sub-events (or *meets*) are grouped into sets called *nodes*. Sub-events derived from the same event are nested in the same node. Sub-events whose original events are

connected by spread events or avoid split assignment constraints also lie in the same node. Events connected by link event constraints have their meets connected in such a way that the time assigned to one of these meets is also extended to the other connected meet. Each meet also contains a set of times called *time domain*, which defines the times that could be assigned to the meet. *Time domains* are chosen based on preferred time constraints. A meet contains one *task* for each demanded resource in the event that it was derived from. Each task also contains a set of resources of the proper type called *resource domain*. The resources available in *resource domain* are based on preferred resource constraints [9]. Pre-assigned times and resources are also assigned in this step.

In *time assignment phase*, a time is assigned to each meet. At first, a layer, which is a set of nodes containing meets preassigned to a given resource, is built for each resource in which a hard avoid clash constraint applies. Then, the layers are sorted in such a way that the hardest layers (layers with less available choices for assignment) come first, and the times are assigned to the meets of each layer, one by one. This assignment is made through a minimum-cost matching between meets of a given layer and times. Each edge of the graph has a cost proportional to the impact of this assignment on the objective function.

Finally, the resources are assigned in the *resource assignment phase*. For each resource type, one of two procedures is executed: (i) if the resource assignment for this resource type is constrained by avoid split assignment constraints, a resource packing algorithm is invoked; (ii) otherwise, a simple heuristic is used. These two procedures can be described as follows:

- The packing of a resource consists in assigning tasks to the resource in such a way that the solution cost is kept as low as possible. It is accomplished through maximum use of the resource, under its workload limits. The resources are placed in a priority queue, in which more demanded resources are prioritized. At each iteration, a resource is dequeued and processed.

- The simple heuristic consists in assigning the resource that minimizes the objective function for each task, from the most constrained to the least constrained.

It is possible to estimate the number of tasks whose resource assignment is infeasible through a maximum matching in an unweighed bipartite graph

(tasks are demand nodes and resources are supply nodes).

3.2. Variable Neighborhood Search

The original Variable Neighborhood Search algorithm was proposed by Mladenovic and Hansen [21]. It consists in a local search method that explores the search space by making systematic changes in the neighborhood structure. Algorithm 1 presents the implementation of VNS used in this work. Initially, neighbourhood 1 is selected (line 2). Afterwards, at each iteration, a new neighborhood function k is selected according to a pre-established order. A random neighbor s' is generated using this neighborhood (line 4) and a descent method is applied to s' (line 5). If the best solution found by the descent method s'' is better than the best known solution s (line 6), then s'' replaces s and the first neighborhood structure is chosen to continue the search (lines 7 and 8). Otherwise, the algorithm switches to the next neighborhood structure (line 10), and the search continues. The process ends when the stop condition is met.

Algorithm 1: Basic Structure of the VNS algorithm

Input: Initial solution s.
Output: Best solution s found.

```
1  while stopping criterion is not met do
2      k ← 1;
3      while k ≤ k_max do
4          s' ← RandomNeighbor(N_k(s));
5          s'' ← descentMethod(s');
6          if f(s'') ≤ f(s) then
7              s ← s'';
8              k ← 1;
9          else
10             k ← k + 1;
11 return s;
```

The VNS algorithm employed in this work for improving KHE solutions is the one proposed in [8]. It is a variant of the original algorithm called Skewed VNS, originally proposed in [22]. This variant employs a relaxed rule to accept the candidate solution s'', as shown in equation (1).

$$f(s'') - \alpha \times \rho(s, s'') \leq f(s) \tag{1}$$

where α is a predefined parameter and $\rho(s, s'')$ is the distance between s and s''.

Therefore, it becomes possible to accept a new solution that is worse than s if the distance between this solution and s is large enough to trigger the update condition. Finally, the distance between the solutions is evaluated using the distance metric considered in [8].

Random Non-Ascendent (RNA) movements, with a stopping criterion of 1,000,000 consecutive iterations without improvement, was used in the decent phase. This is justified by the fact that the union of several neighborhood functions usually generates a very large search space, composed of many flat landscapes. At each iteration, the decent method randomly selects one of six neighborhood functions and generates a neighbor. If the neighbor complies with the update rule, then it is accepted.

The following neighborhood functions are considered:

- Event Swap (ES).
- Event Move (EM).
- Event Block Move (EBM).
- Resource Swap (SW).
- Resource Move (RM).
- Kempe Move (KM).

The selection of the neighborhood function is done according to a given set of probabilities. If the instance requires the assignment of resources (i.e. there is at least one Assign Resource constraint), then the probabilities are: $p(\text{ES}) = 0.20$, $p(\text{EM}) = 0.38$, $p(\text{EBM}) = 0.10$, $p(\text{RS}) = 0.20$, $p(\text{RM}) = 0.10$ and $p(\text{KM}) = 0.02$. Otherwise, the neighborhood functions RS and RM are not used, and the probabilities become: $p(\text{ES}) = 0.40$, $p(\text{EM}) = 0.38$, $p(\text{EBS}) = 0.20$ and $p(\text{KM}) = 0.02$. These values were adjusted based on experimentation. A description of the six neighborhood functions employed is given in the next subsection.

3.2.1. Neighborhood Structure

The neighborhood structure $N(s)$ considered is composed of six moves (or neighborhood functions)[1]. This neighborhood structure is very similar to the one proposed by the winner of ITC2011 [4, 23], except that the move Permute Resources was removed. This move is computationally expensive and it does not provide significant improvement on the candidate solutions. The considered functions are presented in sequence.

1. Event Swap (ES) Two events e_1 and e_2 are selected and their *times* t_1 and t_2 are swapped. Figure 3 presents an example of this move.

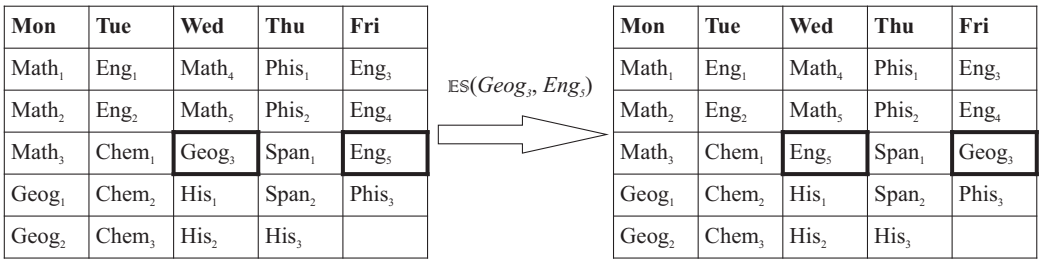

Figure 3: Example of Event Swap [4].

2. Event Move (EM)
   An event e_1 is moved from its original *time* t_1 to an empty *time* t_2. Figure 4 presents an example of this move.
3. Event Block Swap (EBS)
   Similarly to ES, the Event Block Swap swaps the *times* of two events e_1 and e_2. However, if the events have different durations, e_1 is moved to the last time occupied by e_2. This move allows time swaps without losing the allocation contiguity. Figure 5 presents an example of this move. When the events are not contiguous, this move works exactly as Event Swap.
4. Resource Swap (RS)
   The resources r_1 and r_2, assigned to events e_1 and e_2, are swapped. Such an operation is only allowed if the resources r_1 and r_2 are of the

[1]In terms of notation, a neighborhood function k is denoted by $N_k(s)$.

Mon	**Tue**	**Wed**	**Thu**	**Fri**
$Math_1$	Eng_1	$Math_4$	$Phis_1$	Eng_3
$Math_2$	Eng_2	$Math_5$	$Phis_2$	Eng_4
$Math_3$	$Chem_1$	$Geog_3$	$Span_1$	Eng_5
$Geog_1$	$Chem_2$	His_1	$Span_2$	$Phis_3$
$Geog_2$	$Chem_3$	His_2	His_3	

$\mathbb{EM}(Chem_3, Sex\_5)$

Mon	**Tue**	**Wed**	**Thu**	**Fri**
$Math_1$	Eng_1	$Math_4$	$Phis_1$	Eng_3
$Math_2$	Eng_2	$Math_5$	$Phis_2$	Eng_4
$Math_3$	$Chem_1$	$Geog_3$	$Span_1$	Eng_5
$Geog_1$	$Chem_2$	His_1	$Span_2$	$Phis_3$
$Geog_2$		His_2	His_3	$Chem_3$

Figure 4: Example of Event Move [4].

Mon	**Tue**	**Wed**	**Thu**	**Fri**
$Span_1$	Eng_1	$Math_3$	$Phis_1$	Eng_3
$Math_1$	Eng_2	$Math_4$	$Phis_2$	Eng_4
$Math_2$	$Chem_1$	$Geog_3$	$Span_2$	Eng_5
$Geog_1$	$Chem_2$	His_1	$Span_3$	$Phis_3$
$Geog_2$	$Chem_3$	His_2	His_3	

$\mathbb{EBS}(Span_1, Math_1)$

Mon	**Tue**	**Wed**	**Thu**	**Fri**
$Math_1$	Eng_1	$Math_3$	$Phis_1$	Eng_3
$Math_2$	Eng_2	$Math_4$	$Phis_2$	Eng_4
$Span_1$	$Chem_1$	$Geog_3$	$Span_2$	Eng_5
$Geog_1$	$Chem_2$	His_1	$Span_3$	$Phis_3$
$Geog_2$	$Chem_3$	His_2	His_3	

Figure 5: Example of Event Block Swap [4].

same type (i.e. both have to be teachers). Figure 6 presents an example of this move.

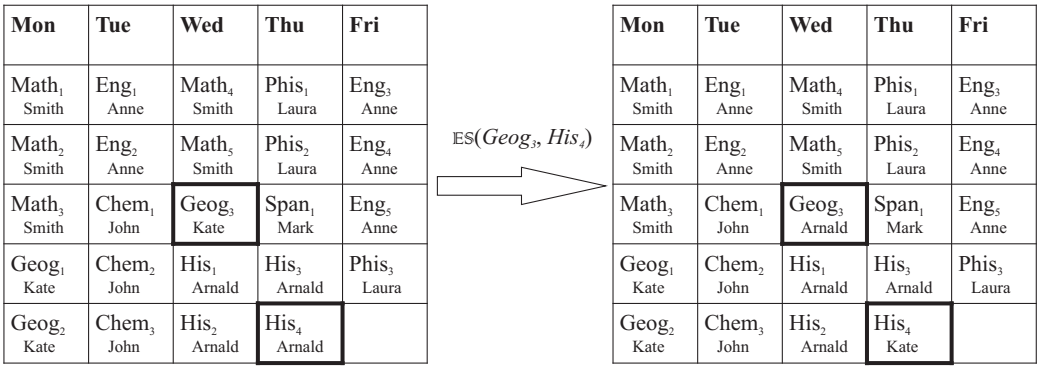

Figure 6: Example of Resource Swap [4].

5. Resource Move (RM)
   The resource r_1, assigned to an event e_1, is replaced by a new resource r_2, randomly selected from the available resources that can be used to attend e_1. Figure 7 presents an example of this move.
6. Kempe Move (KM)
   Two distinct times t_1 and t_2 are selected. The events assigned to times t_1 and t_2 are listed and represented as nodes in a graph. If two nodes (events) n_1 and n_2 share resources, they are connected by an edge. Edges are created only between nodes assigned in distinct times. Therefore, the generated graph is bipartite, and it is known as conflict graph. Every edge in the conflict graph has a weight, which is the cost difference in the objective function assuming the exchange of times between the events in the pair (n_1, n_2). Afterwards, the method looks for the lowest cost path in the conflict graph and it makes the exchange of times in the chain. This procedure is similar to the one proposed by Tuga *et al.* [24]. Figure 8 presents an example of this move.

3.3. Matheuristic

Matheuristics are heuristic algorithms made by the cooperation between metaheuristics and mathematical programming methods (MP) [25, 26, 27].

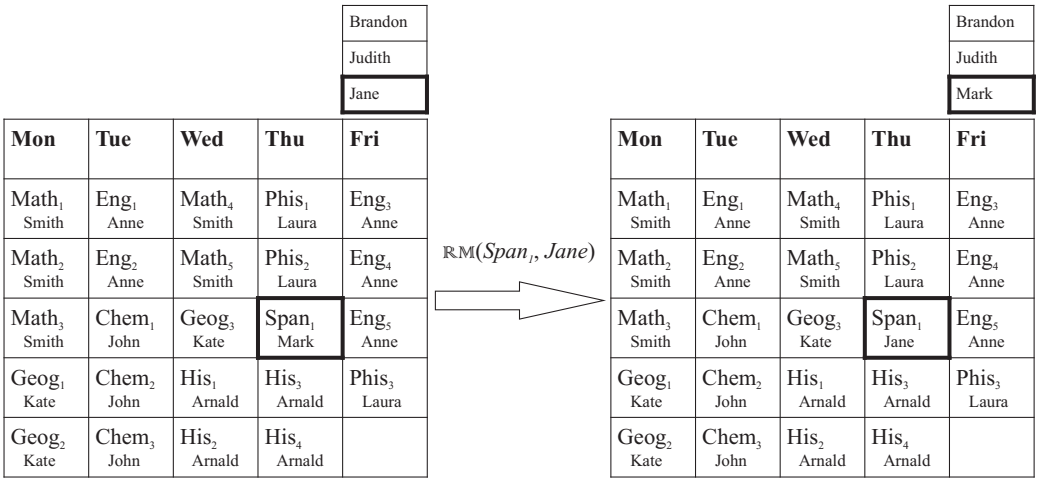

Figure 7: Example of Resource Move [4].

In integrative matheuristic model, metaheuristics are incorporated inside exact algorithms or vice-versa [28].

In the proposed approach, a metaheuristic works at the master level, controlling low level local search procedures. These local searches are reduced Integer Programming (IP) models, in which a subset of variables is fixed to the current values in the incumbent solution, and the remaining variables of the model can be freely modified by the IP solver. The IP model will be presented in the following subsection. Afterwards, the algorithm that was built based on this model is described.

3.3.1. Integer Programming Formulation

The IP formulation considered in this work was recently proposed by Kristiansen *et al.* [11]. This model is able to handle any XHSTT instance. For sake of brevity, only the input data, the basic variables, and some constraints of the formulation are described here. The complete formulation can be found in the original reference.

The input data for this formulation is a set of times T, a set of time groups TG, a set of resources R, a set of resource groups RG, a set of events E, a set of event groups EG, and a set of constraints C. An event $e \in E$ has a duration $d_e \in \mathbb{N}$, and a number of event resources, each one denoted as $er \in EventRes(e)$. An event resource defines the demand of a given

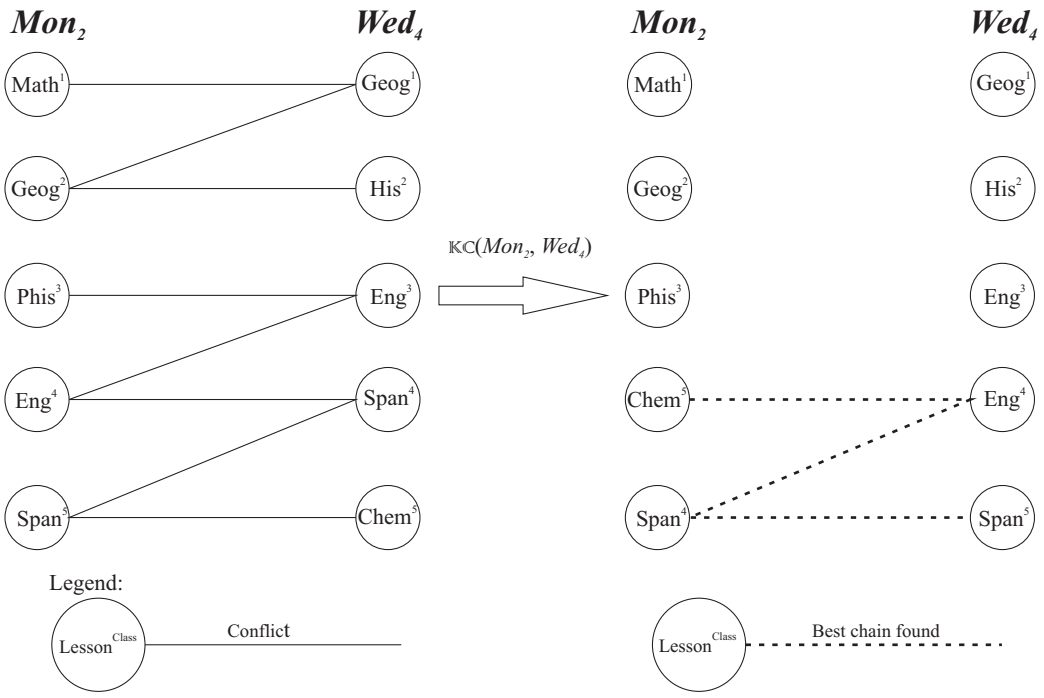

Figure 8: Example of Kempe Move [4].

resource for the respective event. This resource can be pre-assigned or not, and, if it is not pre-assigned, to assign a resource of the proper type becomes solver responsibility. Furthermore, an event resource er can undertake a specific role_{er} in event e, which is used to link the event resource to certain constraints. Generally, an event has to be split into sub-events, whose sum of durations matches the duration of the original event. This formulation creates the "full set" of sub-events $se \in SE$ with different lengths, such that all possible combinations of sub-events for a given event can be handled. For example, an event of duration 4 generates a set of possible sub-events of the following durations: 1, 1, 1, 1, 2, 2, 3 and 4.

Variable $x_{se,t,er,r} \in \{0,1\}$ assumes value 1 if sub-event $se \in SE$ is assigned to start time $t \in T$ and resource $r \in Resources(er)$ is assigned to event resource $er \in EventRes(se)$, or 0 otherwise. To reduce the amount of non-zeros in the IP model, three auxiliary variables are introduced: (i) binary variable $y_{se,t}$ assumes value 1 if sub-event $se \in SE$ is assigned to start time $t \in T$; (ii) variable $v_{t,r} \in \mathbb{N}$ denotes the number of times that resource $r \in R$

is used in time $t \in T$; (iii) binary variable $w_{se,er,r} \in \{0,1\}$ is 1 if sub-event $se \in SE$ is assigned to resource $r \in R$ for event resource $er \in EventRes(se)$, or 0 otherwise.

Each constraint $c \in C$ applies to a set of events, resources or event groups, called its points of application $p \in AppliesTo(c)$. The points of application should be interpreted in an abstract way: p may represent an event e, a resource r or an event group eg. The number of violations is given by slack variables $s_{c,p}^{constype} \in \mathbb{N}$. This slack variable is calculated differently for each constraint present in XHSTT. In the following, three XHSTT constraints are presented. The formulation of the remaining constraints can be found in [11]. Let set $C^{constype} \subset C$ denotes all constraints of a certain type:

Assign Time : The assign time constraint penalizes sub-events in which times are not assigned. Slack variable $s_{c,e}^{assigntime}$ represents the total duration of those sub-events derived from the specific event in which a time is not assigned.

$$d_e - \sum_{t \in T, se \in SubEvents(e)} d_{se} \times y_{se,t} = s_{c,e}^{assigntime} \quad \forall c \in C^{assigntime}, e \in AppliesTo(c) \tag{2}$$

Avoid Unavailable Times : An avoid unavailable times constraint specifies that certain resources are unavailable for any event at certain times. Slack variable $s_{c,r}^{unavailabletimes}$ denotes the number of unavailable times that a resource is attending events. Let $t \in UnavailableTimes(c)$ denote that t is an unavailable time for constraint $c \in C^{unavailabletimes}$.

$$\sum_{t \in UnavailableTimes(c)} q_{r,t} = s_{c,r}^{unavailabletimes} \quad \forall c \in C^{unavailabletimes}, r \in AppliesTo(c) \tag{3}$$

Avoid Clashes : These constraints specify that certain resources should not have clashes, which means they should not be assigned to two or more resources simultaneously. The constraint produces a set of deviations for each resource. For each time, the number of occurrences of given resource minus one is calculated to estimate the deviation ($s_{c,r,t}^{avoidclashes}$) of that resource for that time.

$$v_{t,r} - 1 \leq s_{c,r,t}^{avoidclashes} \quad \forall c \in C^{avoidclashes}, r \in AppliesTo(c), t \in T \tag{4}$$

The objective function penalty for each constraint is calculated according to its weight $w_c \in \mathbb{N}$, number of violations $s_{c,p}^{constype} \in \mathbb{N}$, and cost function type $CostFunction$. There are three types of cost function in XHSTT: linear, quadratic and step. The most common is the linear one, which is calculated as follows:

$$f(s_{c,p}^{constype}) = w_c \times \sum_{p \in AppliesTo(c)} s_{c,p}^{constype}. \tag{5}$$

Given the definition of all constraint types of XHSTT, and their respective slack variables, the objective of the model is stated as to minimize z, as shown in (6).

$$z = f(s_{c,er}^{assignres}) + f(s_{c,e}^{assigntime}) + \cdots + f(s_{c,r}^{limitworkload}) \tag{6}$$

The differentiation between hard and soft constraints is handled as follows:

1. If the input solution to the matheuristic has any hard constraint violation:
   (a) Load the model with hard constraints only;
   (b) Solve the model with regard to hard constraints;
   (c) If the matheuristic achieves a solution that does not violate any hard constraint:
      i. Include the soft constraints in the model;
      ii. Include a constraint which states that the cost of hard constraint violation cannot increase ($z^{hard} = 0$);
      iii. Solve the model with regard to soft constraints.
2. Otherwise:
   (a) Load the full model;
   (b) Include a constraint which states that the cost of hard constraint violation cannot increase ($z^{hard} = 0$);
   (c) Solve the model with regard to soft constraints.

3.3.2. Algorithm

The main idea of the algorithm is similar to the one proposed in Sorensen and Stidsen [15]. Considering that X represents the set of $x_{se,t,er,r}$ variables, s represents the current solution and $n(.)$ a neighbourhood function, the proposed matheuristic is presented in Algorithm 2.

Algorithm 2: Proposed matheuristic

Input: XHSTT instance P, initial solution s, number of resources n, maximum of optimal iterations per neighborhood size $optMax$.

Output: Best solution s found.

```
1  M ← load IP model;
2  X ← load values for variables from s;
3  optInARow ← 0;
4  while elapsedTime < timeout do
5      V ← ∅;
6      count ← 0;
7      while count < n do
8          randomly select a resource ρ;
9          foreach variable x_{se,t,er,r} ∈ X do
10             if r = ρ then
11                 V ← V ∪ {x_{se,t,er,r}};
12         count ← count + 1;
13     fix variables X \ V to their current value;
14     s ← invoke IP solver with short time limit;
15     if IPSolverStatus = OptimalSolution then
16         optInARow ← optInARow + 1;
17         if optInARow = maxOpt then
18             n ← n + 1;
19     else
20         optInARow ← 0;
21         if n > 1 then
22             n ← n − 1;
23     release fixed variables;
24 return s;
```

The algorithm takes as input an instance of XHSTT problem, an initial solution s and two parameters: the number n of resources to have their variables freed per iteration and the maximum of consecutive iterations achieving the optimal solution before increasing the neighborhood size $optMax$ (line 2). This initial solution was obtained by the VNS algorithm. In line 5, an empty set (list) of variables is created. While the number n of resources was not selected yet, a resource ρ is randomly selected (line 8); then, for each variable $x_{se,t,er,r} \in X$, if the resource r in $x_{se,t,er,r}$ is the same as the selected resource ρ, the variable $x_{se,t,er,r}$ is added to set V. In line 13, all variables except the selected ones have their values fixed. In line 14 the IP solver is invoked with a small time limit. After that, the neighborhood size is adjusted if necessary (lines 15 to 22) and all variables are freed for the next iteration of the algorithm (line 23).

The number of resources n to be selected per iteration was set as 5 as well as the maximum of consecutive iterations achieving the optimal solution before increasing the neighborhood size $optMax$. In line 14, a short limit of a tenth of the total available time was considered for each iteration. Usually the IP solver takes only a few seconds per iteration.

Figure 9 presents an example of this IP neighborhood. Suppose a timetabling problem with one class (5th grade) and three teachers (John, Kate and Luke). Consider also that it is not desirable for teachers to be involved in lectures for more than two days. In this example, resources John and Kate are randomly selected to have their schedule optimized at this iteration. Figure 10 presents the results after solving the IP model with these variables set free. Note that it would be hard to remove this constraint violation through the VNS neighborhoods, a large and unlikely chain of Event Swap moves would be required.

It is important to mention that alternative methods of selecting which variables should be freed at each matheuristic iteration were also explored, more precisely: (i) all variables from a given time group (e.g. all variables from Monday); (ii) all variables from a set of randomly selected times (e.g. all variables from Monday_3, Tuesday_1, Friday_2 and Friday_3); (iii) all variables from a set of randomly selected events; and (iv) a percentage of the variables of the IP model, at random. These alternative neighborhoods are not presented in this paper since they led to worse results when compared with the resource neighborhood. In addition, it is very hard to adjust the neighborhood size properly in those cases.

5th grade class

Mon	Tue	Wed	Thu	Fri
Eng_1	$Span_3$	$Math_1$	$Chem_1$	$Math_3$
Eng_2	$Span_4$	$Math_2$	$Chem_2$	$Phis_1$
$Span_1$	Eng_3	Bio_1	$Info_1$	$Math_4$
$Span_2$	Eng_4	Bio_2	$Info_2$	$Phis_2$

Jonh

Mon	Tue	Wed	Thu	Fri
		$Math_1$		$Math_3$
		$Math_2$		
			$Info_1$	$Math_4$
			$Info_2$	

Kate

Mon	Tue	Wed	Thu	Fri
			$Chem_1$	
			$Chem_2$	$Phis_1$
		Bio_1		
		Bio_2		$Phis_2$

Luke

Mon	Tue	Wed	Thu	Fri
Eng_1	$Span_3$			
Eng_2	$Span_4$			
$Span_1$	Eng_3			
$Span_2$	Eng_4			

Cluster busy times violation: More that two working days for John and for Kate

Figure 9: Example of resource optimization IP neighorhood.

5th grade class

Mon	Tue	Wed	Thu	Fri
Eng_1	$Span_3$	$Math_1$	$Math_3$	$Chem_1$
Eng_2	$Span_4$	$Math_2$	$Math_4$	Phi_1
$Span_1$	Eng_3	Bio_1	$Info_1$	$Chem_2$
$Span_2$	Eng_4	Bio_2	$Info_2$	$Phis_2$

Jonh

Mon	Tue	Wed	Thu	Fri
		$Math_1$	$Math_3$	
		$Math_2$	$Math_4$	
			$Info_1$	
			$Info_2$	

Kate

Mon	Tue	Wed	Thu	Fri
				$Chem_1$
				Phi_1
		Bio_1		$Chem_2$
		Bio_2		$Phis_2$

Luke

Mon	Tue	Wed	Thu	Fri
Eng_1	$Span_3$			
Eng_2	$Span_4$			
$Span_1$	Eng_3			
$Span_2$	Eng_4			

Figure 10: Solution timetables after the optimization of the resources selected in Figure 9.

4. Computational Experiments

Experiments were performed on an Intel ® i7 4510-U 2.6 Ghz PC with 8GB of RAM under Ubuntu 12.04 operating system. The software was coded in C++ and compiled with GCC 4.6.1. The obtained results were validated with the HSEval validator[2]. An academic version of Gurobi 6.5.1 was used to solve IP models.

The results are represented by the pair (H, S), where H and S denote the cost of hard and soft constraint violations, respectively. When hard-constraints are not violated, only the cost of soft-constraint violations is reported. Our solver, along with our solutions and reports, can be found at the GOAL-UFOP website[3]. We invite the interested reader to validate our results.

4.1. Results

The goal of the first experiment was to estimate the performance of the proposed Hybrid Solver on a scenario similar to the one of the ITC2011 competition. Therefore, the instances of the competition were considered[4]. In addition, the same rules were applied: the time limit was adjusted to be equivalent to 1,000 seconds in the benchmark provided by the organizers and the number of available threads was set to 1.

Regarding the ranking procedure, each algorithm was executed 5 times for each instance and the average result was recorded. Each solver received a ranking on each instance, from 1 (best) to N (worst), according to the average costs obtained (N is the number of algorithms compared). The solver with the smaller average rank is considered as the winner.

VNS algorithm [8] was used as benchmark because, to the best of authors' knowledge, it has the best average performance on these instances. Since only two algorithms were considered for comparison, they were ranked as 1 or 2 on each instance and the average rank could vary between 1 and 2. The obtained results are shown in Table 2.

The rankings obtained by VNS and the hybrid solver are a strong evidence that the hybrid algorithm is superior. To confirm that, an one-sided paired t-test with 95 % confidence level was conducted. The confidence interval

[2] `http://sydney.edu.au/engineering/it/` and `~jeff/hseval.cgi`

[3] `http://www.goal.ufop.br/softwares/hstt`

[4] `https://www.utwente.nl/ctit/hstt/archives/XHSTT-ITC2011-hidden/`

Table 2: Average solution costs for Standalone VNS and Hybrid solvers.

Instance	Standalone VNS [8]	Hybrid Solver
BrazilInstance2	29.0	**5.8**
BrazilInstance3	104.8	**31.2**
BrazilInstance4	(6.4, 110.6)	**63.6**
BrazilInstance6	121.0	**51.6**
FinlandElementarySchool	3.0	3.0
FinlandSecondarySchool2	0.0	0.0
Aigio1stHighSchool2010-2011	0.8	**0.0**
Italy_Instance4	50.2	**32.2**
KosovaInstance1	13.8	**9.0**
NetherlandsKottenpark2003	1384.8	**1257.8**
NetherlandsKottenpark2005A	**(13.8, 6950.8)**	(14.4, 5677.8)
NetherlandsKottenpark2008	(15.4, 31039.4)	**(15.0, 68015.2)**
NetherlandsKottenpark2009	**(11.0, 12197.0)**	(11.8, 11545.0)
Woodlands2009	(6.6, 0.0)	**57.8**
Spanish school	963.6	**474.8**
WesternGreeceUniversity3	**5.0**	5.6
WesternGreeceUniversity4	5.2	**5.0**
WesternGreeceUniversity5	0.0	0.0
Ranking	1.75	**1.25**

$[-\infty, -0.17]$ and the p-value of 4.3×10^{-3} corroborate the statement that the hybrid algorithm is better than the standalone metaheuristic.

For some instances on which VNS only obtained infeasible solutions, the proposed algorithm was able to find feasible solutions. This is the case for BrazilInstance4, KosovaInstance1 and Woodlands2009. However, both algorithms could not reach feasible solutions on Dutch instances. Future work will focus on understanding and overcoming this limitation.

For the Ducth instances, the matheuristic was not even invoked for most executions, since the metaheuristic phase did not reach the stagnation condition within the time limit. It explains the similar performance achieved by both approaches for this set of instances (see Table 2).

4.2. Improving Best Known Solutions

The 25 instances of the XHSTT-2014 archive[5] were used to estimate the efficiency of the algorithm when larger processing times are available. This set was considered because its best known solutions are continuously updated by the community. In these tests, the Hybrid Solver was allowed to run for 36,000 seconds and two combinations were considered:

HS–BNS: Hybrid Solver is applied considering the current best known solution as the initial solution.

HS–KHE: Hybrid Solver is applied considering the KHE solution as the initial solution, as the previous experiment.

On the one hand, the intention behind testing the HS–BNS is to evaluate the refinement capacity of the proposed algorithm. This setup is particularly useful if good solutions are available, such as the scheduling of the last semester, for example. On the other hand, HS–KHE is more general since it applies to cases in which initial solutions are not known a priori.

The results obtained by the two combinations are shown in Table 3. Results of HS-KHE are split into three columns, KHE, VNS and Math. These columns contain, respectively, the initial solution generated by KHE, the best solution found by VNS, and the final solution achieved after applying the matheuristic. Additionally, the lower bounds ($\mathcal{LB}$) and the current best known solutions ($\mathcal{UB}$) are also reported in the table. In column HS-BNS and

[5] `https://www.utwente.nl/ctit/hstt/archives/XHSTT-2014/`

sub-column Math (of HS-KHE), results which are better than the current best known solutions are highlighted in bold. Results marked with a dash '-' are already optimal and, therefore, they cannot be improved.

Table 3: HS–BNS and HS–KHE results

Instance	$\mathcal{LB}$	$\mathcal{UB}$	HS–BNS	KHE	HS–KHE VNS	HS–KHE Math
AU-BG-98	0	(1, 386)	**415**	(3, 608)	(2, 398)	(2, 398)
AU-SA-96	0	24	**17**	(4, 22)	(4, 21)	(3, 21)
AU-TE-99	0	125	**33**	(2, 152)	(1, 36)	(1, 36)
BR-SA-00	5	5	-	31	22	**5**
BR-SM-00	51	51	-	(8, 117)	(3, 99)	52
BR-SN-00	35	35	-	113	104	**35**
DK-FG-12	285	3310	**1514**	3411	1669	**1668**
DK-HG-12	(7, 0)	(12, 3124)	**(12, 2611)**	(12, 4689)	(12, 3371)	(12, 3371)
DK-VG-09	(0, 0)	(2, 4097)	**(2, 2718)**	(2, 4691)	(2, 2765)	**(2, 2765)**
ES-SS-08	334	336	336	657	415	351
FI-PB-98	0	0	-	0	0	**0**
FI-WP-06	0	1	1	19	11	2
FI-MP-06	77	83	**77**	102	84	**77***
GR-H1-97	0	0	-	0	0	**0**
GR-P3-10	0	0	-	2	0	**0**
GR-PA-08	0	4	**3**	11	3	**3**
IT-I4-96	27	34	**27**	46	42	**27***
KS-PR-11	0	3	3	17	4	**0***
NL-KP-03	0	617	**420**	1371	1151	1103
NL-KP-05	89	1078	**784**	(15, 10117)	(8, 4460)	(8, 4460)
NL-KP-09	170	9180	**6265**	(10, 5125)	(8, 8370)	(7, 64470)
UK-SP-06	0	(16, 2258)	**(15, 1892)**	(66, 3788)	(53, 1524)	(53, 1524)
US-WS-09	0	697	**111**	532	124	**124**
ZL-LW-09	0	0	-	(13, 16)	16	**0**
ZL-WL-09	0	0	-	(16, 0)	(3, 0)	**0**

The proposed algorithm showed remarkable results on improving best known solutions: 15 out of 17 non-optimal solutions of the instance set considered were improved. New optimal solutions were reached for FI-MP-06, IT-I4-96 and KS-PR-11. Taking into account the experiments in which the KHE solutions were used as starting point, one can conclude that the

achieved solutions are good in several instances, even beating the previous best known solutions or achieving the optimal solution in some cases. However, for some datasets, such as the Australian and Dutch instances, the solver is still not very efficient.

Finally, it should be noticed that the time spent on optimization (10 hours) is not critical in many applications since the schedule can be performed weeks, or even months, before the beginning of the semester.

5. Concluding Remarks

This paper presented a hybridization of metaheuristics and matheuristics for timetabling problems. This hybridization was little explored in the literature and achieved remarkable results, surpassing by far the standalone metaheuristic algorithm. Aiming to improve the current best known solutions, the proposed algorithm achieved remarkable results: 15 out of 17 solutions were improved through this new approach.

Future work will focus on investigating smarter ways of selecting the free variables in the matheuristic. Other possible future works are (i) to improve the existing mathematical programming formulation for XHSTT, and; (ii) to make a deeper study about neighbourhood sizes and ILP based procedures, such as Local Branching [29] and Relaxation Induced Neighborhoods (RINS) [30].

Acknowledgment

The authors would like to thank the Brazilian agencies CAPES, CNPq, and FAPEMIG for the financial support.

References

[1] M. R. Garey, D. S. Jonhson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, USA, 1979.

[2] G. Post, L. Di Gaspero, J. Kingston, B. McCollum, A. Schaerf, The third international timetabling competition, Annals of Operations Research (2013) 1–7.
URL http://dx.doi.org/10.1007/s10479-013-1340-5

[3] G. Post, J. Kingston, S. Ahmadi, S. Daskalaki, C. Gogos, J. Kyngas, C. Nurmi, N. Musliu, N. Pillay, H. Santos, A. Schaerf, XHSTT: an XML archive for high school timetabling problems in different countries, Annals of Operations Research 218 (1) (2014) 295–301. doi:10.1007/s10479-011-1012-2.
URL http://dx.doi.org/10.1007/s10479-011-1012-2

[4] G. Fonseca, H. Santos, T. Toffolo, S. Brito, M. Souza, GOAL solver: a hybrid local search based solver for high school timetabling, Annals of Operations Research (2014) 1–21.
URL http://dx.doi.org/10.1007/s10479-014-1685-4

[5] M. Sørensen, S. Kristiansen, T. Stidsen, International Timetabling Competition 2011: An Adaptive Large Neighborhood Search algorithm, 2012, pp. 489–492.

[6] A. Kheiri, E. Ozcan, A. J. Parkes, HySTT: Hyper-heuristic search strategies and timetabling, in: Proceedings of the ninth international conference on the practice and theory of automated timetabling (PATAT 2012), 2012, pp. 497–499.

[7] J. Romrös, J. Homberger, An evolutionary algorithm for high school timetabling, PATAT '12 Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling, 2012.

[8] G. H. Fonseca, H. G. Santos, Variable neighborhood search based algorithms for high school timetabling, Computers & Operations Research 52, Part B (0) (2014) 203 – 208, recent advances in Variable neighborhood search.
URL http://www.sciencedirect.com/science/article/pii/S0305054813003328

[9] J. H. Kingston, KHE14: An algorithm for high school timetabling, in: 10th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2014), York, United Kingdom, 2014, pp. 26–29.

[10] L. N. Ahmed, E. Ozcan, A. Kheiri, Solving high school timetabling problems worldwide using selection hyper-heuristics, Expert Systems With Applications, in review.

[11] S. Kristiansen, M. Sørensen, T. Stidsen, Integer programming for the generalized high school timetabling problem, Journal of Scheduling (2014) 1–16.
URL http://dx.doi.org/10.1007/s10951-014-0405-x

[12] S. Pirkwieser, G. R. Raidl, Matheuristics for the periodic vehicle routing problem with time windows, Proceedings of matheuristics (2010) 28–30.

[13] F. Della Croce, A. Grosso, F. Salassa, A matheuristic approach for the total completion time two-machines permutation flow shop problem, in: P. Merz, J.-K. Hao (Eds.), Evolutionary Computation in Combinatorial Optimization, Vol. 6622 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 38–47.
URL http://dx.doi.org/10.1007/978-3-642-20364-0_4

[14] H. G. Santos, T. A. Toffolo, R. A. Gomes, S. Ribas, Integer programming techniques for the nurse rostering problem, Annals of Operations Research (2014) 1–27.
URL http://dx.doi.org/10.1007/s10479-014-1594-6

[15] M. Sørensen, T. R. Stidsen, Hybridizing integer programming and metaheuristics for solving high school timetabling, 10th International Conference on the Practice and Theory of Automated Timetabling (2014) 557–560.

[16] A. Colorni, M. Dorigo, V. Maniezzo, Metaheuristics for high school timetabling, Computational optimization and applications 9 (3) (1998) 275–298.

[17] T. Frühwirth, S. Abdennadher, University course timetabling, in: Essentials of Constraint Programming, Cognitive Technologies, Springer Berlin Heidelberg, 2003, pp. 117–122.

[18] T. Müller, K. Murray, Comprehensive approach to student sectioning, Annals of Operations Research 181 (1) (2010) 249–269.

[19] J. H. Kingston, A software library for school timetabling, available at http://sydney.edu.au/engineering/it/~jeff/khe/, Accessed in October / 2015 (2012).

[20] G. Post, S. Ahmadi, S. Daskalaki, J. H. Kingston, J. Kyngas, C. Nurmi, D. Ranson, An XML format for benchmarks in high school timetabling, in: Annals of Operations Research DOI 10.1007/s10479-010-0699-9., 2010, pp. 3867 : 267–279.

[21] N. Mladenovic, P. Hansen, Variable neighborhood search, in: Computers and Operations Research, 1997, pp. 24, 1097–1100.

[22] P. Hansen, N. Mladenovic, Variable Neighborhood Search: A Chapter of Handbook of Applied Optimization., Les Cahiers du GERAD G-2000-3. Montreal, Canada, 2000, Ch. 8.

[23] G. Fonseca, H. Santos, T. Toffolo, S. Brito, M. Souza, A SA-ILS approach for the High School Timetabling Problem, in: PATAT '12 Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling, 2012.

[24] M. Tuga, R. Berretta, A. Mendes, A hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem, in: Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on, IEEE, 2007, pp. 400–405.

[25] J. Puchinger, G. Raidl, Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification, in: J. Mira, J. Álvarez (Eds.), Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Vol. 3562 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 41–53.

[26] M. Boschetti, V. Maniezzo, M. Roffilli, A. Bolufé Röhler, Matheuristics: Optimization, simulation and control, in: M. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, A. Schaerf (Eds.), Hybrid Metaheuristics, Vol. 5818 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 171–177.
URL http://dx.doi.org/10.1007/978-3-642-04918-7_13

[27] V. Maniezzo, T. Stützle, S. Voß, Matheuristics: Hybridizing Metaheuristics and Mathematical Programming, 1st Edition, Springer, 2010.

[28] G. Raidl, J. Puchinger, Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization, in: C. Blum,

M. J. B. Aguilera, A. Roli, M. Sampels (Eds.), Hybrid Metaheuristics, Vol. 114 of Studies in Computational Intelligence, Springer Berlin Heidelberg, 2008, pp. 31–62.

[29] M. Fischetti, A. Lodi, Local branching, Mathematical Programming 98 (1-3) (2003) 23–47.
URL http://dx.doi.org/10.1007/s10107-003-0395-5

[30] E. Danna, E. Rothberg, C. Le Pape, Exploring relaxation induced neighborhoods to improve mip solutions, Mathematical Programming 102 (1) (2005) 71–90.