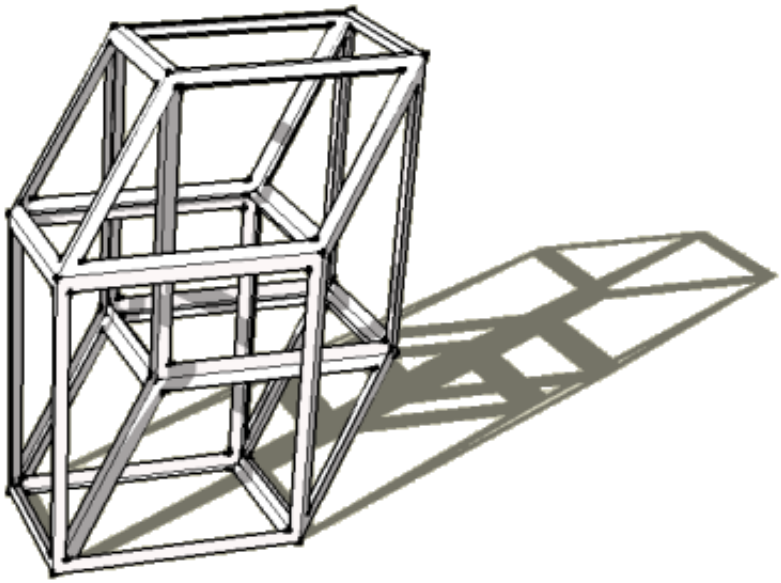
UNIVERSITY OF TWENTE.

Master thesis

A Hyperheuristic for generating Timetables in the XHSTT Format



Graduation Committee:	Dr.ir. G.F. Post Prof.dr. M. Uetz Dr. R.M.J. van Damme
Author:	BSc. Mathijs ter Braak
Graduation period:	September, 2011 - June, 2012

June 26, 2012

Abstract

The High School Timetabling (HSTT) Problem is amongst the most widely used timetabling problems. This problem has varying structures in different high schools even within the same country or educational system. The HSTT Problem in several countries has been studied in order to find a common set of constraints and objectives. The HSTT problem represented in XML format (XHSTT) been designed in order to better model the complete problem and facilitate data exchange between high school timetabling researchers. In this master project, a high school scheduling program is extended, in order to facilitate the construction and application of algorithms to several high school instances. Constraints, a cost evalution of solutions, and several algorithms are implemented. We make use of combinations of algorithms, which can improve the quality of schedules within a certain computation time limit. A hyperheuristic is applied which finds the best combination of algorithms at different stages of the solution process. The hyperheuristic is tested on several real high school instances from different parts of the world.

Preface

This report is the result of the research I have done for my master thesis at the University of Twente in the Netherlands, which was part of the Applied Mathematics master programme at the Discrete Mathematics and Mathematical Programming department. The subject was the high school timetabling problem. I have worked on a planning program and extended it, such that it can create timetables for high school instances from all over the world and compare my results to other researchers working on the same problem.

I found it a great privilege to get the opportunity of working on the High School Timetabling project. Researchers have been working on the problem for years and I enjoyed to be able to give a small contribution. This project is continued in order to get even better solutions within a certain amount of time.

I would like to thank Gerhard Post for his enthousiasm, for always taking the time for giving valuable advice and comments, and for thinking along on improvements of my work and the scheduling program. This always gave me an extra motivation. I would also like to thank other collegues of the DMMP department, especially Johann Hurink and Marc Uetz for assisting me during my years at Applied Mathematics.

Contents

1 Background

In this section, we start with an introduction and the description of the HSTT problem. Next, in Section 2, we describe the goals of the work on this master thesis and give an overview of the report.

1.1 Introduction

The focus of this master thesis is the high school timetabling problem. The High School TimeTabling (HSTT) problem lacks the availability of exchangeable benchmarks in a uniform format. Therefore, in the past six years, researchers have been working on a standard format for the modelling of (high) school timetabling problems and for exchanging HSTT instances. The website dedicated to this project is [1]. The work on this thesis is based on this benchmark project. [2] For this benchmark project, the data format XML was chosen. Therefore, High School Timetabling (HSTT) problems formulated using this format, are called XML High School Timetabling (XHSTT) problems. XHSTT is introduced in Section 3.

The format is supplied with tens of instances, which are availalbe on the website. Researchers have used severeal techniques in order to come up with the best thinkable solutions. Some solutions are optimal, others are near-optimal or could use some larger improvements. The instances are usually too large for computation of every possible solution. Therefore, it is still a challenge to come up with one or more techniques that find the best solutions within a reasonable amount of time. Besides that, methods that are used for one instance, might not be useful at all for another. It would be nice if there was some technique that works for any instance, but usually instance-specific methods are needed, which use specific information about the instance in order to find a good solution.

In this thesis, we describe the HSTT problem and the corresponding XHSTT format. A Delphi program was available which could read instances of this format, but could not yet run algorithms, evaluate constraints or create solutions to the instances. We give a description of this programming work, the heuristics and a hyperheurstic that we created in an attempt to get good solutions within a certain amount of time.

1.2 The HSTT Problem

A HSTT problem describes the problem of the assignment of resources to events and events to times in a high school. Events are activities like lessons or meetings. Resource assignments are demands made by an event which are essential for the event to take place, like teachers, students or rooms. An example of an event is given as XML code in Figure 1. The interpretation of this code is illustrated in a diagram in Figure 2.

```
<Event Id="EN (L1HV2)_8501">
    <Name>EN (L1HV2)</Name>
    <Duration>2</Duration>
    <Course Reference="gr_Course_EN-L1HV2_85"/>
    <Resources>
        <Resource Reference="Class_L1HV2">
            <Role>Class</Role>
            <ResourceType Reference="Class"/>
        </Resource>
        <Resource Reference="Teacher_SCM">
            <Role>Teacher</Role>
            <ResourceType Reference="Teacher"/>
        </Resource>
        <Resource>
            <Role>RoomAlg</Role>
            <ResourceType Reference="Room"/>
        </Resource>
    </Resources>
    <EventGroups>
        <EventGroup Reference="gr_EventsRoomKind_Alg"/>
        <EventGroup Reference="gr_AllEvents"/>
    </EventGroups>
</Event>
```

Figure 1: Example of an event (XML tree view)

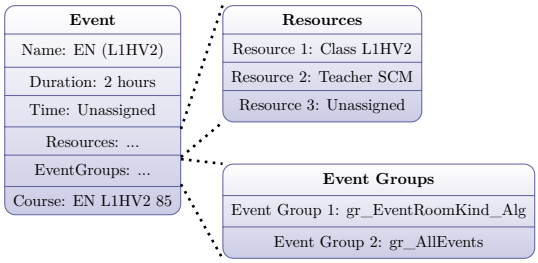


Figure 2: Example of an event (block diagram)

As is shown in Figure 1, an event can be for example 'EN L1HV2', an english lesson intended for class L1HV2. The event is demanded to consume a number of hours. This is called the *Duration* of an event. In this case the duration is two hours. Further more, it needs to be assigned to a starting hour. In this case, no starting hour is assigned yet. A planner decides which one this will be. Because the duration is two, this event will occupy the starting hour plus the succeeding hour.

The event demands that several resources are available to this event at the hours at which this event will take place. The class L1HV2 and the teacher SCM are already assigned to the event. The event still demands a third resource which is not assigned yet. This is a decision of a planner, as well as the event starting time.

The event is the member of some *Event Groups*. An event group is a set of events which have something in common. For example, this event is member of the event group gr_EventRoomKind_Alg. This is a group of all rooms that are of a general type. It is also a member of the group gr_AllEvents, an event group of all events. The introduction of groups is useful for implementation of some constraints, which will be explained in Section 4.

Constraints define the feasibility and quality of a resulting timetable. The combination of a set of events, a set of resources, a time horizon and constraints defining which assignments are allowed or desired, describes the HSTT problem.

1.3 Constraints

In this subsection, an overview of the common constraints of the HSTT problem is given. This overview gives just a motivation for the constraints. For a more technical description, see Section 4.

Split Events and Distribute Split Events

A high school has allocated a number of hours to each event. For example, it may be required that a certain class should have ten hours of English a week. Obviously, a class should not get a lesson that has a duration of ten hours. Either the high school management or the planner should think of a way to split this event into several subevents. So this decision can be predefined or left to the planner. In both cases, the Split Events and Distribute Events constraints define the boundaries on what is allowed regarding the splitting of events. For more details, see Section 3.6.

Assign Resource and Prefer Resource

Usually, each event needs resources which are essential for an event to take place. Sometimes these resources are preassigned. For example, it is usually known which students are assigned to which event. However, in some instances of HSTT problems there are resource assignment decisions to be made by the planner. The Assign Resource constraint checks whether these decisions have been made. Obviously, it is not only important that every event is assigned the required resources . It also matters which resources. For example, English lessons need teachers that can teach English and classes may not be allowed to take lessons after the 6th hour. Therefore, the Prefer Resource constraint tells from which set of resources a certain resource may be chosen.

Assign Time and Prefer Time

Usually events need to be assigned to a time. The Assign Time constraint checks whether this is done. The Prefer Time constraint gives the set of times that are preferred. For example, we might prefer that a biology lesson takes place during the first six hours of a day.

Spread Events and Link Events

Like in the example where it is required that a class takes English lessons with a total duration of ten hours a week, it is desirable that these events should not be given on the same day. On the contrary, it is usually desired that they are uniformly spread over the week. The Spread Events constraint checks whether certain groups of events are spread in a certain way. Besides that, some events are required to be linked, that is, they should be assigned the same times. This is evaluated by the Link Events constraint.

Avoid Clashes and Limit Idle Times

Resources can not attend more than one meeting at the same time. Attending more than one event at once is called a clash. The Avoid Clashes constraint penalyzes whenever a resource is assigned to more than one event at a certain time. Usually it is desired that the number of idle times of resources is limited. This means that for each working day, the number of idle hours is limited. Idle hours are the hours between the busy hours of a day. A busy hour is an hour at which a resource is assigned to an event. The Limit Idle Times constraint raises a penalty if the number of idle hours is outside the given range.

Avoid Unavailable Times, Cluster Busy Times and Limit Busy Times

Resources might have times at which they are unavailable. This is checked by the Avoid Unavailable Times constraint. Besides that, it is normally better when the number of days on which a resource is busy is limited. That is called clustering of busy hours. The amount by wich these hours are clustered is checked by the Cluster Busy Times constraint. The Limit Busy Times constraint puts limits on the number of busy hours of a day, on which a resource is at least one time busy.

1.4 Related problems

Two other classifications of timetabling are the Course timetabling problem and the Examination timetabling problem. See for example [3]. They are discussed here to see how they are related to the High School timetabling problem.

The University course timetabling problem

University course timetabling is the weekly scheduling for all the lectures of a set of university courses, minimizing the overlaps of lectures of courses having common students. The main difference with school timetabling problems is that schools usually have classes with disjoint sets of students, while university courses usually are attended by groups of students which are not disjoint. Moreover, it is common that a professor teaches only one course, while school teachers teach more than one class. Besides that, the assignment of rooms plays a more important role in university timetables, while room or teachers assignment is often a secondary problem in school timetabling. This is because, in school timetabling, classes have a specialized room, like their own room or can be assigned to any room available, with exception of biology or sport rooms.

The Examination timetabling problem

Examination timetabling is the scheduling for the exams of a set of university courses, avoiding overlap of exams of courses having common students, and spreading the exams for the students as much as possible. The students can not skip exams, therefore the conflict condition is very strict. There can be contraints of a type different from the university course timetabling problem. For example, it could

be preferred that students have only one exam a day or that exams should be spread among a number of periods.

1.5 Metaheuristics

The HSTT problem is known to be NP-hard. Therefore, it is hard to decide how the events and resources should be assigned, in order to meet demands made by the school management. The focus of this research project is to investigate metaheuristics that try to solve HSTT problems or give a good approximation to an optimal solution.

2 Goals and approach

This section discusses the goals of the project and the approach.

2.1 Goals

At the beginning of this master project, there was a Delphi computer program available that could read XHSTT instances and parse the elements of an instance together with the other instance data into Delphi objects.

- The first goal is to extend the program such that it is also able to facilitate the application of algorithms on the instance data. Therefore, a planner, solution structure and the evaluation of constraints must be created. An overview of the XHSTT model is given in Section 3, while the associated constraints are described in Section 4. The scope of this master thesis does not cover decisions on how to assign resources to events. This is not a big deal for finding solutions to instances, because in most instances, the resources are already preassigned.
- The second goal is to think of and write heuristics. At the start of this master project, there was no algorithm available yet, although there are algorithms described in literature. A description of literature is given in Section 5.
- The third goal is to create combinations of algorithms which can generate good quality schedules within a reasonable amount of computation time.

The approach and method is discussed in Section 6 and the computational results are given in Section 8.

2.2 Approach

First, the XHSTT format is studied and a computer program that can load instances of this format is extended with the evaluation of constraints and a solution structure. Secondly, literature is studied to get ideas on basic heuristics for the construction of high school schedules. After that, several algorithms are implemented and tested. A hyperheurstic is constructed which can decide which algorithms to apply at a certain stage of the solution process, using different combinations and configurations of algorithms.

3 The XHSTT model

In this section is described how HSTT is formulated as XHSTT format in XML. XHSTT stands for XML (Extensible Markup Language) High School Timetabling. This format has been designed in such a way that, hopefully, all HSTT instances and solutions of instances can be described using this format.

3.1 Overview of the XHSTT archive

First, an overview of the elements of the XHSTT archive is given. The archive consists of a set of instances and a set of corresponding solutions, if known. An XHSTT instance is a description of a high school scheduling problem. They are formed by the basic objects of the XHSTT model, which are sets of times, resources, events and constraints. They are visualized as a tree in Figure 3 and as block diagram in Figure 4.

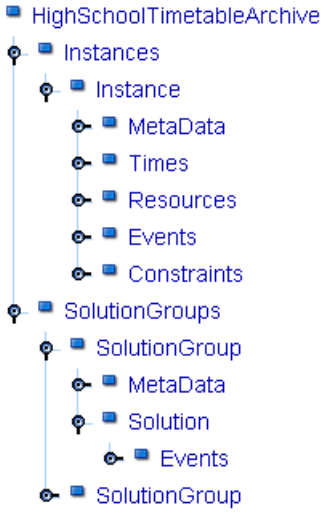
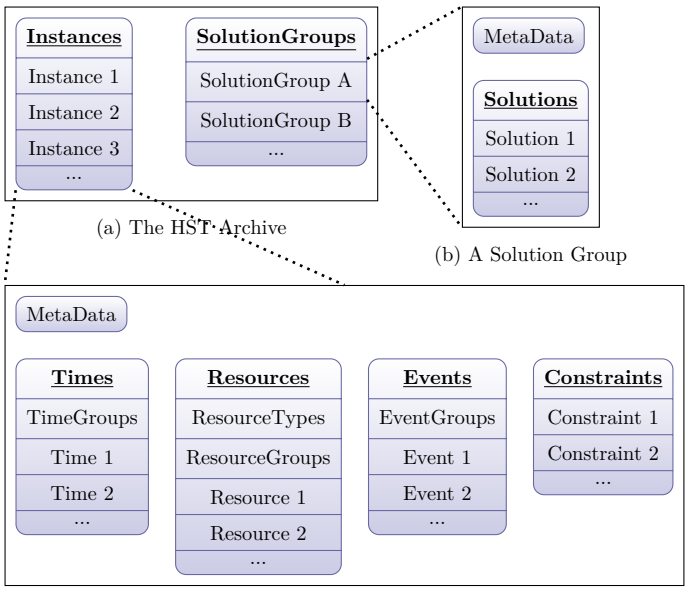


Figure 3: XHSTT tree view



(a) The HST Archive

(b) A Solution Group

(c) An Instance

Figure 4: XHSTT Block diagram

3.2 The Events

Secondly, we explain the elements, starting with the Events.

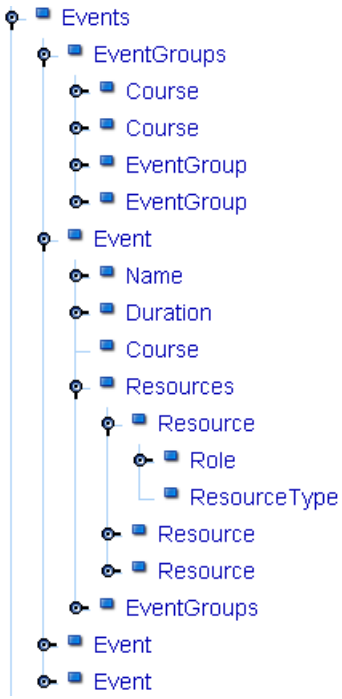


Figure 5: Events element tree view

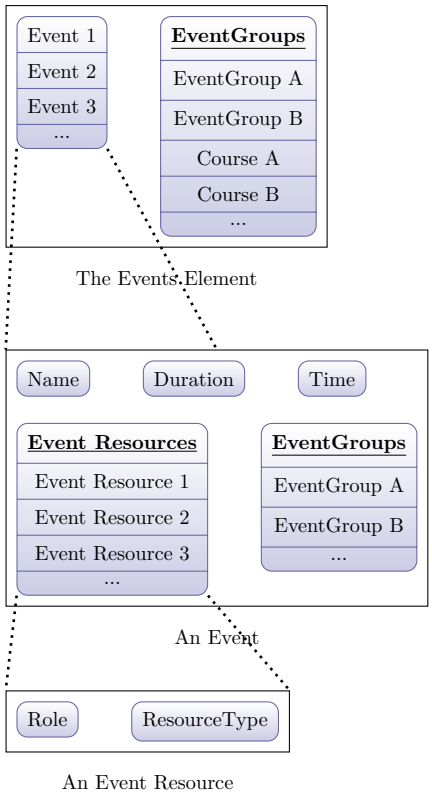


Figure 6: Events element block diagram

The Events element consist of a number of events and defines a number of EventGroups. See the example below. An event refers to the EventGroups if it is a member of an EventGroup. An event can have a number of EventResources. These can be seen as 'slots' to which a resource can be assigned. If a resource is pre-assigned to this event, the corresponding reference is already mentioned in the event. Otherwise, an event resource has a role and a resource type, specifying the kind of resource that is required. The constraints define which resources of the corresponding resource type can play a certain role. Besides that, an event has a duration. That is the number of time periods that must be assigned to this event. Furthermore, it refers to a starting time, whenever a starting time is pre-assigned. (See Figure 5 and 6).

```
<Event Id="en (KO1HV2A)_221000">
 <Name>en (KO1HV2A)</Name>
 <Duration>1</Duration>
 <Course Reference="gr_Course_en-KO1HV2A_2210"/>
 <Resources>
  <Resource Reference="Class_KO1HV2A">
   <Role>Class</Role>
   <ResourceType Reference="Class"/>
  </Resource>
  <Resource Reference="Teacher_HAM">
   <Role>Teacher</Role>
   <ResourceType Reference="Teacher"/>
  </Resource>
  <Resource>
   <Role>RoomAlg</Role>
   <ResourceType Reference="Room"/>
  </Resource>
 </Resources>
 <EventGroups>
  <EventGroup Reference="gr_EventsRoomKind_Alg"/>
  <EventGroup Reference="gr_AllEvents"/>
  <EventGroup Reference="gr_1HV2.en36"/>
 </EventGroups>
</Event>
```

Figure 7: Example of an Event in XML

We give an example. See Figure 7. This is the XML code of an English event. The unique identifier of the event is 'en (KO1HV2A)_221000'. This is of course useful to identify the event. If we refer to this event, this identifier is used. The event has the child elements 'Name', 'Duration', 'Course', 'Resources' and 'EventGroups'. The name of the event is 'en (KO1HV2A)'. We see that between the Duration tags, there is a 1. This means that the event takes 1 hour. Which hour is to be decided. The event has no time pre-assigned, because the child element 'time', which is not obligatory, is not present here. The EventGroups are 'gr_EventsRoomKind_Alg', 'gr_AllEvents', 'gr_1HV2.en36' and the Course 'gr_Course_en-KO1HV2A_2210', which is just a special case of an EventGroup. These are groups of events. It is useful to group the events to which the same constraint applies. This can be for example a group of all events that require a certain room. Because we refer to these event groups here, it means that the event is in each of these groups.

The event has three resource demands. It demands a resource of ResourceType 'Class' and Role 'Class', a resource of ResourceType 'Teacher' with Role 'Teacher' and one of ResourceType 'Room' with Role 'RoomAlg'. The Class and Teacher demands are already satisfied by the preassigned resources 'Class_KO1HV2A' and 'Teacher_HAM' respectively. This is, because right after the 'Resource' element, there is a reference to these resources, which means that those are the resources that are assigned. After the 'Resource' element of the Room resource, there is no reference mentioned. This mean that the demand for a room is still not satisfied and that this is a decision of the planner. Thus there is specified what kind of resource is demanded. We demand a resource of ResourceType 'Room' that can play the Role 'RoomAlg'. It is not possible to assign a resource of the wrong ResourceType. We use a constraint to define which resources can play the Role 'RoomAlg'. For

example, we may not want to give this English lesson in Gym room, thus a Gym room is of the correct ResourceType, but can not play the Role 'RoomAlg'.

3.3 Resources

Normally, events need a number of resources. Here an explanation of resources is given.

Figure 8: Resources element tree view

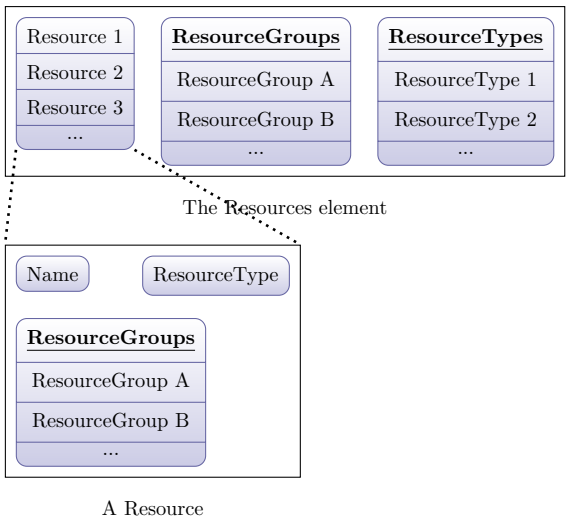


Figure 9: Resources element block diagram

The Resources element consist of a number of resources and defines a number of ResourceGroups and ResourceTypes. Each resource has a name and refers to one of these ResourceTypes and can refer to any number of ResourceGroups, whenever it is a member of such group. (See Figure 8 and 9).

3.4 Times

Besides resources, usually, events have to be assigned to a certain starting time. Here an explanation of the Times element is given.

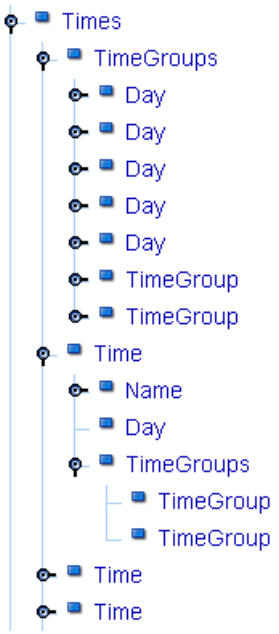


Figure 10: Times element tree view

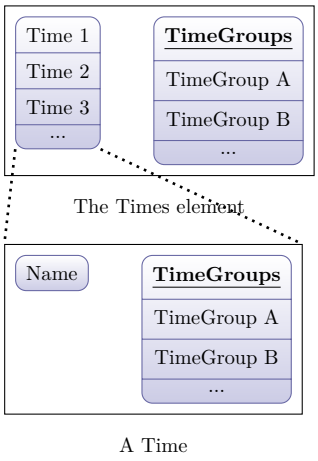


Figure 11: Times element block diagram

The Times element has a number of times and defines a number of TimeGroups. Each time has a name and will refer to the TimeGroups it is a member of. (See Figure 10 and 11).

3.5 The Constraints

Figure 12: Constraints element tree view

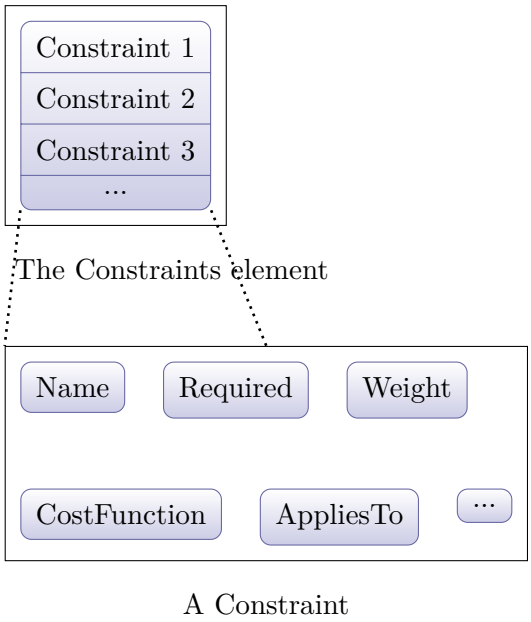


Figure 13: Constraints element block diagram

The Constraints element contains a number of constraints. Each constraint has a name. It has a field Required which tells whether it is a hard or a soft constraint. The field Weight gives the weight factor of the constraint. The CostFunction tells which function must be applied whenever the constraint detects a deviation from the required timetable. The AppliesTo field tells to which elements the constraint applies. (See Figure 12 and 13). Besides that, a constraint can have other properties, which are constraint specific. For more details, see Section 4.

3.6 The Solution

In this section we describe how a Solutions element is built up. It represents a collection of solutions, where each solution represents a timetable for a certain instance. A solution is the result of a certain number of scheduling decisions.

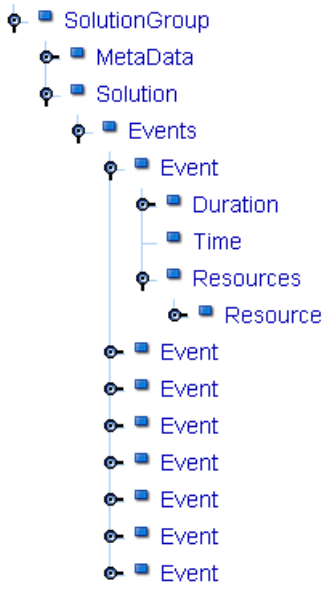


Figure 14: SolutionGroup element tree view

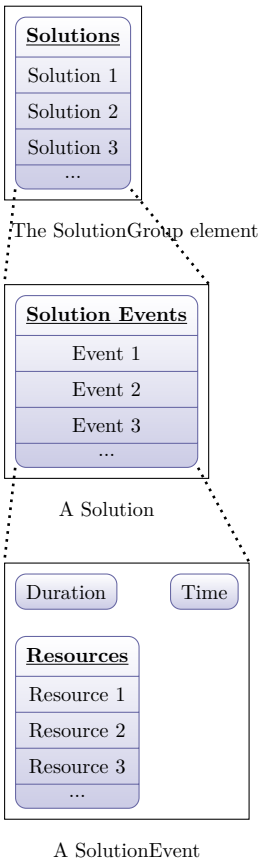


Figure 15: SolutionGroup element block diagram

The Solution contains a list of solution events, which correspond to the defined Events in the instance (See figure 15). Each event has a reference to the corresponding event. However, it is possible that more than one solution event refer to the same instance event. This is, because a solver is allowed to split an event into several solution events, each scheduled a different time. The cost of this splitting is calculated by the constraints. The total duration of these solution events must equal the duration of the corresponding instance event. Therefore, the duration of a solution event is either defined by the instance event duration or by an optional element Duration in the solution event.

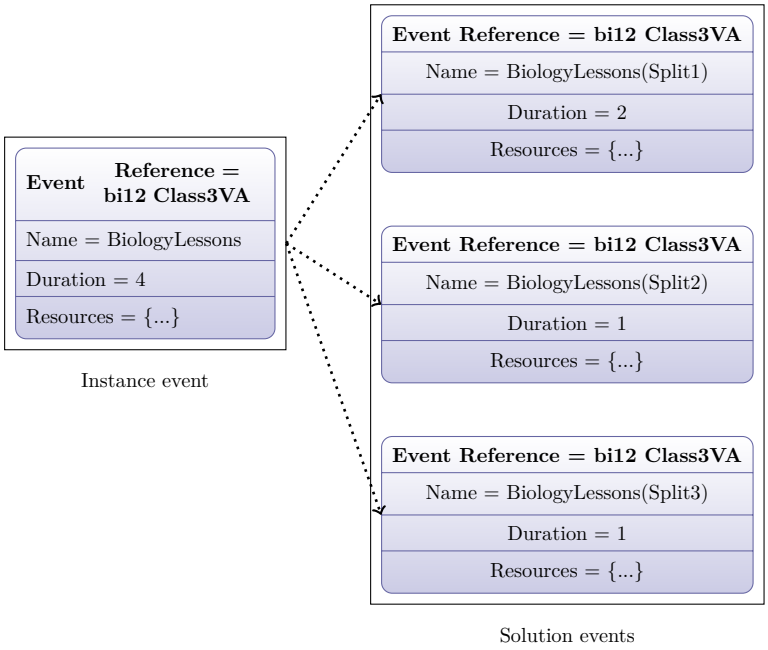


Figure 16: The splitting of an event

We give an example in Figure 16. We have an event called 'bi12 Class3VA'. It's duration is four hours. Obviously, no student should have a biology lesson taking four hours, so we want to split this event. One option is to split this event into one event of duration two, and two events of duration one. Every split event will refer to the original instance event, so that constraints know which split events are part of the same instance event.

Further more, The child element Time may have a referrence to the assigned time and the child element Resources may have a number of Resource elements which are assigned to certain roles of the event.

4 Constraints

The constraints that are used in XHSTT have been introduced in Section 1.3. This section illustrates the use of these constraints.

4.1 The cost specification

Each solution has a cost. The cost is a tuple of two integers:

$$Cost \equiv (\text{Feasibility cost}, \text{Objective cost})$$

Each constraint contributes to the cost. A constraint can either contribute to the feasibility cost or to the objective cost, depending on the boolean *Required*, supplied with the constraint. If $Required = True$, then the constraint contributes to the feasibility cost. If $Required = False$, the constraint yields objective cost. This relates to the fact that every possible solution of the right syntax is a solution, whether it is feasible or not.

The interpretation of the cost is as follows. If the feasibility cost is positive, it means that the corresponding solution does not satisfy each required constraint. However, if it is close to zero, this integer tells that only a small part of the events and resources yield a feasibility cost, and we might be a few steps away from a feasible solution. If the objective cost is positive, some constraints that are not required are violated.

How each constraint contributes to the cost is described here. Each constraint defines a *deviation*, a *weight* and a *cost function*. The deviation is the amount by which a constraint is violated. For each point of application, a deviation is calculated. The point of application is the element that causes a possible violation. A point of application can be for example an event, resource, time group or event group. A cost function, supplied with the constraint, defines how to map these deviations onto the set of nonnegative integers. This can be either a linear or non-linear function. The weight defines the significance of the constraint. The heigher the weight, the more it is important to satisfy a constraint. For each point of application, the cost is the weight multiplied by the cost function over the deviation(s):

$$\text{Cost}_i = \text{Weight} * \text{CostFunction}(\text{Deviations}_i) \quad (1)$$
$$\text{where Deviations}_i = \text{deviations of point of application i}$$

```
<AssignTimeConstraint Id="AssignTimes_1">
 <Name>AssignTimes</Name>
 <Required>true</Required>
 <Weight>1</Weight>
 <CostFunction>Sum</CostFunction>
 <AppliesTo>
  <EventGroups>
   <EventGroup Reference="gr_AllEvents"/>
  </EventGroups>
 </AppliesTo>
```

Figure 17: Example of an AssignTimeConstraint

For example, suppose there is a constraint 'AssignTimeConstraint' which checks whether every event is assigned to a time. See Figure 17. Then we supply this constraint with a set of events to which this constraint applies, because we may not want to apply this constraint to every event that occurs in the instance, but to a subset of these events. But in this case, it is the event group 'AllEvents', of which obviously every event is a member. We define the point of application to be each event mentioned under the 'AppliesTo' section of the constraint. This means, that a deviation is calculated for each of these events. We define the deviation to be equal to one if the event is not assigned a time, and zero otherwise. As can be seen in the figure, the weight of the constraint is one and the cost function is 'Sum'. Suppose there is an event which is not assigned to a time. Then the cost for this event would be: 1 * Sum(1) = 1. If there are 20 events assigned to a time and 10 unassigned, then this constraint would yield 10 times a cost of one and 20 times a cost of zero, resulting in a total cost of 10. The total cost of this constraint, is just the sum of the costs of each event in the AppliesTo section of the constraint.

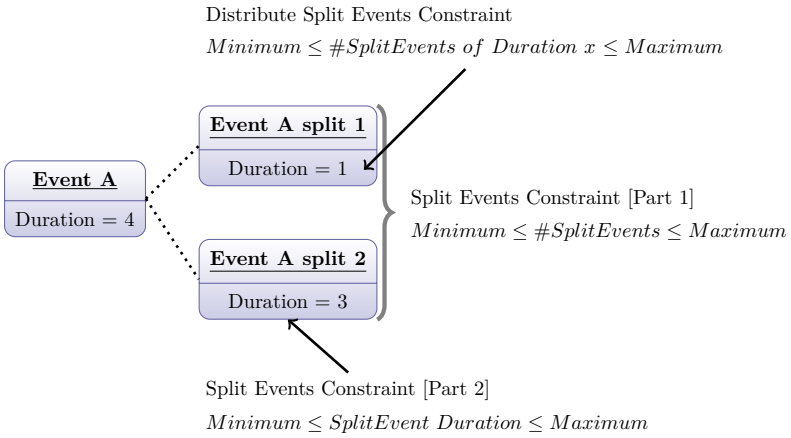


Figure 18: Splitting Constraints

4.2 The Split Events constraint

This constraint puts limits on the number of events into which an event may be split. For explanation of splitting events, see Section 3.6. A deviation is calculated by the sum of the deviation of part 1 and part 2.

Part 1 of this constraint says:

$$Minimum \leq \#SplitEvents \leq Maximum$$

Part 2 of this constraint says:

$$Minimum \leq SplitEventDuration \leq Maximum$$

In Figure 18, Event A of duration four is split into two events of duration 1 and 3 respectively. Suppose the range of *#SplitEvents* is between 4-5. So for Event A, the number of split events differs an amount of 2 from this range. Thus the Part 1 of the deviation yields a deviation of 2. Suppose the range of *SplitEventDuration* is between 3-3. Then there is one split event with a duration that is not in this range. This yields a deviation of 1 for Part 2. So the deviation for Event A is $2 + 1 = 3$.

4.3 The Distribute Split Events constraint

Sometimes it is demanded that the planner is given a specification of particular split event durations that are desired or not desired. The Distribute Split Events constraint puts limits on the number of split events of a particular duration. See Figure 18. The constraint is as follows:

$$Minimum \leq \#\text{SplitEvents of Duration x} \leq Maximum$$

Suppose that the number of split events with duration $x = 1$ is demanded to be in the range 3-4. Since event A in Figure 18 is split into an event of duration 1 and an event of duration 3, there is only 1 event of duration 1. This amount falls 2 short of the demanded range. Thus the deviation for this constraint is 2.

```
<Event Id="Event362">
 <Name>Event362</Name>
 <Duration>2</Duration>
 <Course Reference="gr_Event362"/>
 <Resources>
  <Resource Reference="1-BAT-A"/>
  <Resource Reference="JD"/>
  <Resource>
   <Role>GIM</Role>
   <ResourceType Reference="Room"/>
  </Resource>
  <Resource>
   <Role>PISTA</Role>
   <ResourceType Reference="Room"/>
  </Resource>
 </Resources>
 <ResourceGroups/>
 <EventGroups>
  <EventGroup Reference="gr_AllEvents"/>
 </EventGroups>
</Event>
```

Figure 19: Example of an event that demands resources

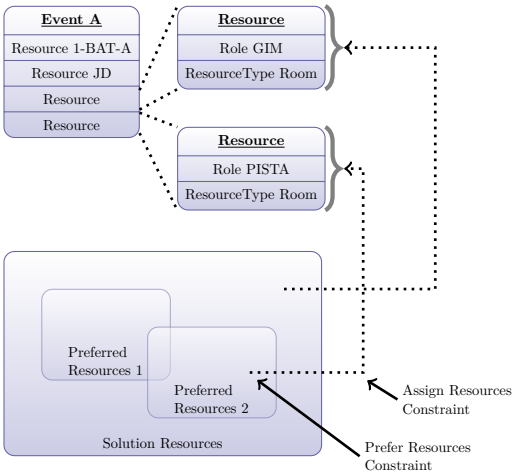


Figure 20: Resource assignment Constraints

```
<AssignResourceConstraint Id="AssignResources_PISTA">
 <Name>Assign PISTA</Name>
 <Required>true</Required>
 <Weight>1</Weight>
 <CostFunction>Sum</CostFunction>
 <AppliesTo>
  <EventGroups>
   <EventGroup Reference="gr_AllEvents"/>
  </EventGroups>
 </AppliesTo>
 <Role>PISTA</Role>
</AssignResourceConstraint>
```

Figure 21: AssignResource Constraint example

```
<PreferResourcesConstraint Id="PreferredResourcesPISTA">
 <Name>Prefer PISTA room</Name>
 <Required>true</Required>
 <Weight>1</Weight>
 <CostFunction>Sum</CostFunction>
 <AppliesTo>
  <EventGroups>
   <EventGroup Reference="gr_AllEvents"/>
  </EventGroups>
 </AppliesTo>
 <Resources>
  <Resource Reference="PISTA1"/>
  <Resource Reference="PISTA2"/>
 </Resources>
 <Role>PISTA</Role>
</PreferResourcesConstraint>
```

Figure 22: PreferResources Constraint example

4.4 The Assign Resource constraint

This constraint checks whether each resource demand of an event is assigned a resource. A resource demand must be satisfied by one or more resources for the whole duration of the corresponding event. For example, if an event is split into several events, each split event covers a different part of the event duration. Thus each part should be assigned the demanded resources.

An example of an event that demands resources is illustrated in Figure 19 and 20. Event A is already assigned the resources '1-BAT-A' and 'JD'. It still demands two other resources which are not assigned yet. One demand is the demand for a resource of *ResourceType* 'Room', that can play the role 'GIM', the other one is a

demand for a resource of ResourceType 'Room' as well, but a resource that can play the role 'PISTA'. It is illegal to assign a resource of the wrong ResourceType. The attribute 'Role' is an identifier which is used to distinguish between resources of the same ResourceType in one event. It is used by the Prefer Resources Constraint.

An example of a corresponding Assign Resource Constraint is given in Figure 21. This constraint is applied to all events in the 'AppliesTo' section, which have demands for resources with the Role 'PISTA'. It checks whether these events are assigned a resource or not. For this constraint, it does not matter which resource is assigned, it simply checks *if* there is one assigned. For each event, the deviation is the number of hours that are not covered by a resource.

4.5 The Prefer Resources constraint

This constraint checks whether each assigned resource is one of the preferred ones. The specification of the preferred resources is supplied with the constraint.

An example of a Prefer Resources constraint is given in Figure 22. This constraint is applied to every event in the AppliesTo section of this constraint which demand resources that have the given Role 'PISTA'. Events under AppliesTo that do not demand resources with the Role 'PISTA' are ignored by this constraint. So the first unpreassigned resource demand of EventA in Figure 20 is not a point of application of this constraint, because it has the role GIM. But the second resource demand in this figure *is*, because it has the role 'PISTA'. So this constraint specifies that this resource demand should be assigned one of the resources mentioned under Resources in Figure 22. In this case that should be either 'PISTA1' or 'PISTA2'. The deviation of this constraint is the number of hours during which this event example does have a resource assigned, but a resource which is not one of the preferred ones.

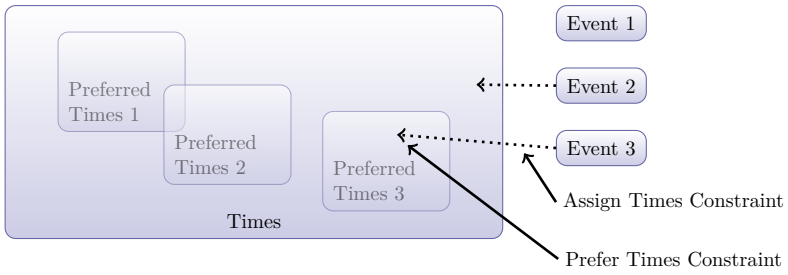


Figure 23: Event assignment Constraints

```
<AssignTimeConstraint Id="AssignTimes_1">
 <Name>AssignTimes</Name>
 <Required>true</Required>
 <Weight>1</Weight>
 <CostFunction>Sum</CostFunction>
 <AppliesTo>
  <EventGroups>
   <EventGroup Reference="gr_AllEvents"/>
  </EventGroups>
 </AppliesTo>
```

Figure 24: AssignTime Constraint example

```
<PreferTimesConstraint Id="PreferredTimes_8">
 <Name>PreferredTimesDurationTwo</Name>
 <Required>true</Required>
 <Weight>2</Weight>
 <CostFunction>Sum</CostFunction>
 <AppliesTo>
  <EventGroups>
   <EventGroup Reference="gr_AllEvents"/>
  </EventGroups>
 </AppliesTo>
 <TimeGroups>
  <TimeGroup Reference="gr_TimesDurationTwo"/>
 </TimeGroups>
 <Duration>2</Duration>
</PreferTimesConstraint>
```

Figure 25: PreferTimes Constraint example

4.6 The Assign Time constraint

This constraint checks whether events are assigned to a time. Like the AssignResource Constraint, this constraint does not check what time is assigned, only *if* a time is assigned.

An example of this constraint is given in Figure 24. This constraint specifies that the events mentioned under AppliesTo, in this case all events, require to be assigned a time. How this constraint is evaluated is illustrated in Figure 23. Event2 and Event3 are both assigned to a time. However, for Event1, no time assignment has been made. Because this event is mentioned under AppliesTo of this constraint, the constraint is applied to this event. The deviation is the number of hours of the event duration that should be assigned a time but are not. In this case, it is the total duration of Event1. Suppose the duration of Event1 is 3 hours, then the deviation is 3. Event2 and Event3 both have no deviation, because they have been assigned to a time.

4.7 The Prefer Times constraint

This constraint checks whether each time assignment is one of the preferred ones. The preferred times are specified by this constraint.

An example is given in Figure 25. The constraint is applied to all events that are member of the group 'gr_AllEvents'. The constraint specifies that for each of these events, if a time is assigned, it should be one of the hours that are in the group of hours 'gr_TimesDurationTwo', the group of hours that are not the last hour of a day.

In Figure 23, Event1 and Event2 are both not assigned to a preferred resource, where Event3 is. All three events are member of the group 'gr_AllEvents'. However, only Event2 and Event3 are points of application of this constraint, because they are are assigned to a time. Since Event2 is assigned to a non-preferred time, this assignment results in a deviation. The deviation is the total duration of Event2.

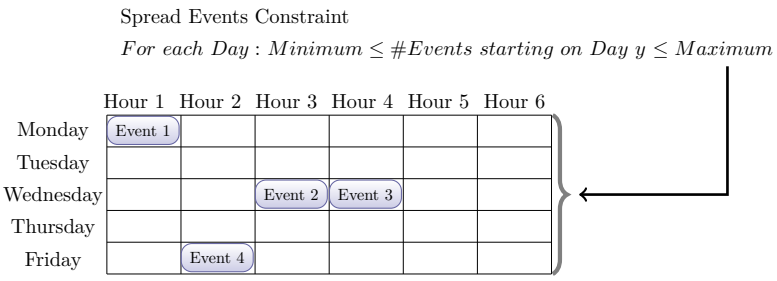


Figure 26: Event spreading Constraints

4.8 The Spread Events constraint

This constraint puts limits on the time assignments of a group of events. This constraint specifies a group of hours. It counts the number of events from the event group which have a starting time which is one of these hours. This should not exceed a maximum or minimum.

For example, suppose this constraint is applied to a group of events, namely EventGroupA consisting of Event1, Event2, Event3 and Event4. Suppose the number of events is limited by the range 0-1 for each day of the week. Then in Figure 26, we see that there is one event scheduled on monday, zero on tuesday, two on wednesday, zero on thursday and one on friday. Then for each day, there are 0-1 events of EventGroupA scheduled, except for wednesday. On wednesday there are two events scheduled, namely Event3 and Event4. For EventGroupA, This constraint yields several deviations, one deviation for each day. For each day, the deviation is the difference between the number of scheduled events and the specified range 0-1. So EventGroupA has a deviation of zero for monday, tuesday, thursday and friday. It yields a deviation of one for wednesday, since 2 falls 1 short off the range 0-1. These five deviations are summed by the cost function into a cost, as specified by Equation 1.

```
<LinkEventsConstraint Id="Link_7-8">
        <Name>Link Event7 and Event8</Name>
        <Required>true</Required>
        <Weight>1</Weight>
        <CostFunction>Sum</CostFunction>
        <AppliesTo>
                <EventGroups>
                        <EventGroup Reference="gr_Event7and8"/>
                </EventGroups>
        </AppliesTo>
</LinkEventsConstraint>
```

Figure 27: LinkEvents Constraint example

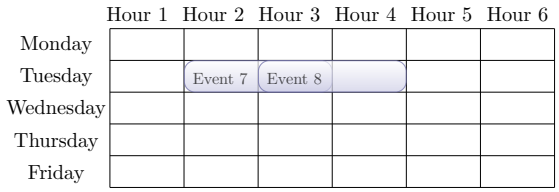


Figure 28: Linked Event Constraints

4.9 The Link Events constraint

This constraint specifies that certain events should have the same start and finish time.

We give an example in Figure 27 and Figure 28. This constraint is applied to the event group 'gr_Event7and8', which consists of the events Event7 and Event8. The constraint specifies that these events should be linked. That is, they should be scheduled on the same time. This is illustrated by Figure 28, in which Event7 is scheduled on Tuesday Hour 2+3, while Event8 is scheduled on Tuesday Hour 3+4. The deviation of the event group is the number of hours on which not all events of the event group are scheduled, but at least one of them. In this case, on Tuesday Hour 3, both Event7 and Event8 are scheduled. This yields no deviation. However, on Tuesday Hour 2, only Event7 is scheduled. On Tuesday, Hour 4, only Event8 is scheduled. Therefore, the deviation for the event group is 2. The cost associated with the event group is the weight multiplied by the CostFunction over the deviation 2, as specified by Equation 1.

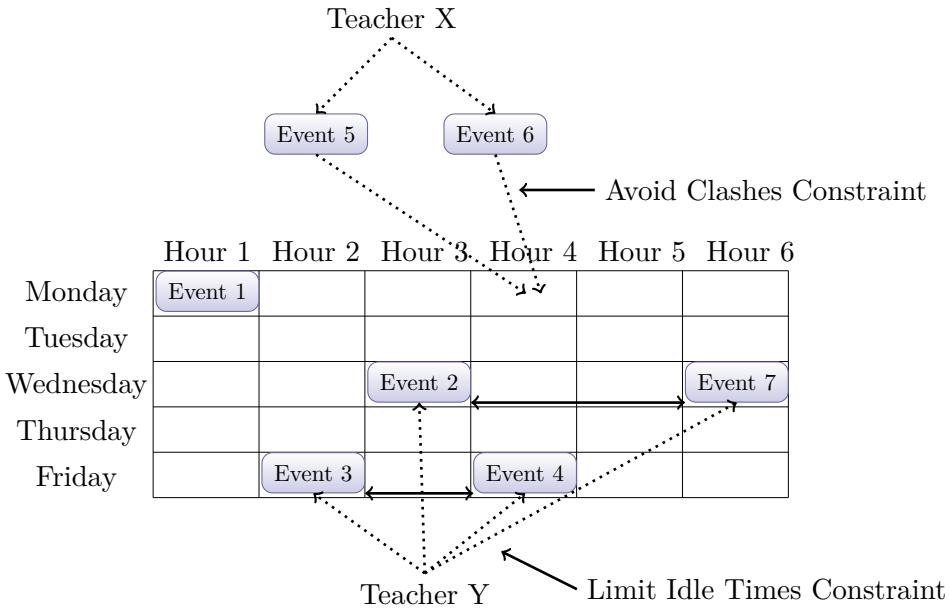


Figure 29: Clashes and idle times Constraints

```
<AvoidClashesConstraint Id="NoResourceClashes_11">
 <Name>NoResourceClashes</Name>
 <Required>true</Required>
 <Weight>1</Weight>
 <CostFunction>Sum</CostFunction>
 <AppliesTo>
  <ResourceGroups>
   <ResourceGroup Reference="gr_Teachers"/>
   <ResourceGroup Reference="gr_Students"/>
   <ResourceGroup Reference="gr_Rooms"/>
   <ResourceGroup Reference="gr_Classes"/>
  </ResourceGroups>
 </AppliesTo>
</AvoidClashesConstraint>
```

Figure 30: Avoid Clashes Constraint example

```
<LimitIdleTimesConstraint Id="FreePeriodsConstraint_15">
 <Name>No more than 0 FreePeriods</Name>
 <Required>false</Required>
 <Weight>2</Weight>
 <CostFunction>SquareSum</CostFunction>
 <AppliesTo>
  <ResourceGroups>
   <ResourceGroup Reference="gr_Teachers"/>
  </ResourceGroups>
 </AppliesTo>
 <TimeGroups>
  <TimeGroup Reference="gr_Day_ma"/>
  <TimeGroup Reference="gr_Day_di"/>
  <TimeGroup Reference="gr_Day_wo"/>
  <TimeGroup Reference="gr_Day_do"/>
  <TimeGroup Reference="gr_Day_vr"/>
 </TimeGroups>
 <Minimum>0</Minimum>
 <Maximum>0</Maximum>
</LimitIdleTimesConstraint>
```

Figure 31: Limit Idle Times Constraints

4.10 Avoid Clashes constraint

This constraint penalyzes whenever a resource attends two or more events at the same time. The constraint is applied to each resource in the AppliesTo section. If also resource groups are mentioned, it is applied to every resource in each resource group.

We give an example of this constraint in Figure 30. The constraint is applied to the teachers in the group 'gr_Teachers', students in the group 'gr_Students', to

rooms in the group 'gr_Rooms' and to classes in the group 'gr_Classes'. Suppose Teacher X from the group 'gr_Teachers is assigned to a resource demand of Event5 and to one of Event6. This is illustrated in Figure 29. Suppose Event5 and Event6 are both assigned to Hour 4 of Monday. Then this Avoid Clashes constraint yields a deviation, because Teacher X should not attend Event5 and Event6 at the same hour. The assignment of Event5 and Event6 using the same resource at the same hour is called a clash. The deviation is equal to the number of events to which the resource is scheduled too many at this hour. So in this case the deviation is 1. If a resource has more than 1 hours to which it attends more than 1 event, the deviation of the resource is the sum of the deviations at each time.

4.11 Limit Idle Times constraint

This constraint specifies that a resource should not be busy, idle and then busy again, with respect to each of one or more days.

An example is given in Figure 31. This constraint is applied to all teachers and specifies that for each day, both the minimum and maximum of the number of idle hours should be zero. In this case the days are Monday, Tuesday, Wednesday, Thursday and Friday. There is only a single deviation, namely the sum over the number of out-of-range idle hours of each day that are out of the specified range 0-0. This is illustrated by an example in Figure 29. Teacher Y is only assigned to the resource demands of the events Event2, Event3, Event4 and Event7. The events are assigned to Wednesday Hour 3, Friday Hour 2, Friday Hour 4 and Wednesday Hour 6 respectively. So the teacher is busy on these hours and is not busy on the others. We see that Teacher Y has many hours at which the teacher is not assigned to a resource demand. However, only on Wednesday and Friday we see that there are hours in between busy hours for Teacher Y. These are the idle hours of the teacher.

For Monday, Tuesday and Thursday, the number of idle hours is 0 and in the range 0-0. For Wednesday, the number of idle hours is 2. This is 2 too many. For Friday, the number of idle hours is 1, that is 1 too many. Thus the deviation for Teacher Y is $2+1=3$. The Cost associated with Teacher Y is calculated as follows. The Cost-Function is 'SquareSum', that is the square of the sum of deviations. The Weight is 2. Since there is only one deviation, namely 3, the cost is: $2 * 3^2 = 18$.

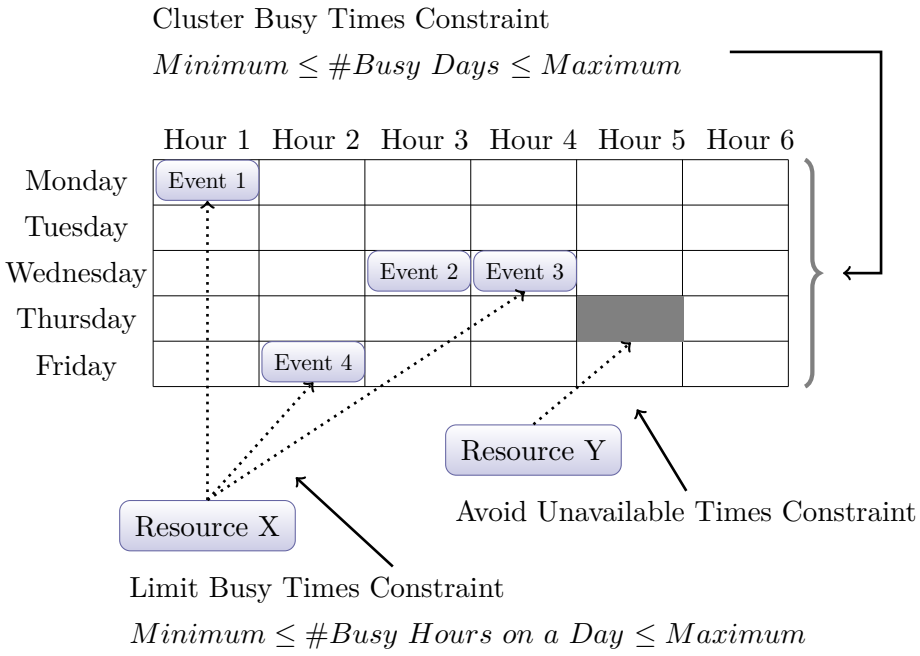


Figure 32: Resource spreading Constraints

4.12 Avoid Unavailable Times constraint

Like the Prefer Times constraint which is applied to events, this constraint specifies that a resource is not available at a certain time. Therefore it should not be assigned to events that are assigned to one of the unavailable times.

In Figure 32, Resource Y is assigned to the resource demand of an event at Hour 5 of Thursday. However, this is an hour at which this resource is unavailable. The deviation is the number of hours to which a resource is assigned but at which it is unavailable, in this case 1.

4.13 Limit Busy Times constraint

This constraint puts limits on the number of times at which a resource is busy with respect to a busy day. This constraint says:

$$Minimum <= \#Busy\ Hours\ on\ a\ Day <= Maximum$$

We give an example in Figure 32. Suppose a resource is assigned to the resource demand of Event1, Event2, Event3 and Event4. Suppose $Minimum = 2$ and $Maximum = 3$. We see that Resource X is only busy on Monday, Wednesday and Friday, thus the constraint produces only deviations for these days. We see that on Monday, the number of busy hours falls 1 short off the range 2-3. On Wednesday, the number of busy hours is 2 and is within the range 2-3. On Friday, we have again one busy hour too few. Thus, we have a deviation of 1 for Monday

and a deviation of 1 for Friday. The Cost associated with Resource X is Weight * CostFunction(Deviation of 1,Deviation of 1), by Equation 1.

4.14 Cluster Busy Times constraint

This constraint puts limits on the number of days at which a resource is busy at least one of the hours. This constraint says:

$$Minimum \leq \#Busy\ Days \leq Maximum$$

In Figure 32, Resource X is busy on Monday, Wednesday and on Friday. Suppose $Minimum = 2$ and $Maximum = 3$. Then the number of busy days is within the demanded range. Therefore the deviation of Resource X is 0.

5 Literature

There is quite some literature about school timetabling. The following papers were particularly interesting for my master thesis.

5.1 Creating High School Timetables using Tabu Search

This bachelorassignment [4] is not published, but available in printed form at the university of Twente. It is interesting, because it describes a method for finding a good initial schedule, which supported the idea for a Grading algorithm in Section 6.3. It is also based on [5], in which the combination of ejection chains with tabu search is described, which is used in Section 6.7.

A timetable for the Dutch high school Kottenpark is created. This problems considers creating a weekly timetable using the following constraints:

1. Pre-defined: duration of events, teacher assignments to events
2. Hard constraints: assignment of times and room types to linked events
3. Soft constraints: assignment of times and room types to events that are not linked, spreading events over the week, avoiding idle times for teachers and advanced class students, teacher unavailable times, avoiding idle times for base class students
4. There are some constraints which are not used, in order to simplify the problem: Preferring times of events, limit teacher busy times, splitting of events, constraints regarding the scheduling of some special events not at the same time or directly next to each other, assignment of rooms

A feasible schedule is created using tabu search, scheduling only the linked events and choosing a room type for each of the event. The difficulty of scheduling an event is defined by weight factors. This difficulty depends on the number of event assignments of the assigned teacher, the number of days that an event needs to be scheduled implied by the spreading constraint, the number of occupation times of the room, and the duration of events.

The initial state of this tabu search algorithm is a schedule with a certain number of events that are scheduled. These events can be a block of events that are linked. A new state is found by taking a next unscheduled event and search for the best time and room type assignment. The cost of each state transition is the sum of the weight of clashing events events minus the weight of the event to be scheduled. If the best assignment results in a clash, the conflicting events are unscheduled. The tabu list is defined as the list of state transitions where there is a conflict between recently unscheduled events. This algorithm is extended by taking into account the soft-constraint costs into the state transition cost.

Using this model in combination with this tabu search algorithm, feasible schedules are found, while a certain part of the soft-constraints are still violated.

5.2 A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems

This paper [6] describes a simulated annealing algorithm for timetabling. It is used in Section 6.5. It is interesting, because they claim that this algorithm increases the efficiency and performance of simulated annealing.

This paper approximately solves the high school timetabling problem using a simulated annealing based algorithm. The set of constraints considered are:

1. Pre-defined: room assignment to each event
2. Hard constraints: avoid clashes, avoid unavailable times, no idle times for class
3. Soft constraints: spread events for class, limit busy times for teacher, limit idle times for teacher

The solution approach consists of two phases. In phase 1, first a randomly generated timetable is created, in which exactly one teacher is assigned to each subject of each class and for every subject, the demanded number of events is scheduled. Next, simulated annealing is used to improve the hard constraint violations. In each iteration, for each class, a period is chosen in which the assignments violate any hard constraints. Neighbours are evaluated by switching the events of a class scheduled in this period with the events of this class in a random other period.

In phase 2, a new simulated annealing algorithm is started. Again, in each iteration, two random periods are chosen in which neighbours are evaluated by switching the events of a class scheduled in these periods. If a neighbour violates any hard constraints, it is rejected. Otherwise it is accepted if it improves the soft constraint violations or by some small probability if it makes it worse.

This algorithm can compete with other effective approaches, but it has still to be tested on large scale timetables.

5.3 Resource Assignment in high school timetabling

This paper [7] describes a method for the assignment of resources, especially teachers and rooms. Because the scope of this master thesis does not cover the assignment of resources, we found it interesting how this could be done if we wanted to implement this as well.

In most European instances, teachers are preassigned. In Australian instances, events are linked in a complex way, which makes it impractical to assign teachers before the events are scheduled. For resource assignment, the *global tixel matching* is used. This is a large bipartite matching model.

This model is a kind of market with demand and supply The term *tixel* is introduced as 'time pixel' and represents the supply unit of one resource at one time. So there is a supply tixel for every combination of resource and time. A demand

tixel is the request of one resource at a certain time unit. A bipartite graph is created by connecting a number of demand tixel nodes with a number of supply tixel nodes, whenever a resource is suitable for assignment to a certain time. This is a maximum weighted bipartite matching problem. The solution to this problem give a lower bound to the number of unassignable tixels, which can be used to diagnose supply problems.

This model can also be used for resource assignment, after the assignment of all events times. Unavailability times can be modelled by adding an unavailability demand tixel, representing the time that a certain resource is unavailable. Satisfying this demand ensures that the corresponding supply tixel is not assigned to another demand tixel. Likewise, workload limits of resources can be modelled.

Optimization techniques are used to improve the efficiency of the global tixel matching technique. Several algorithms are described that try to minimize the number of unsatisfied demand tixels, given a schedule in which events are assigned to times.

6 Approach and method

6.1 Programming

In this section, the work on programming of the scheduling program is described.

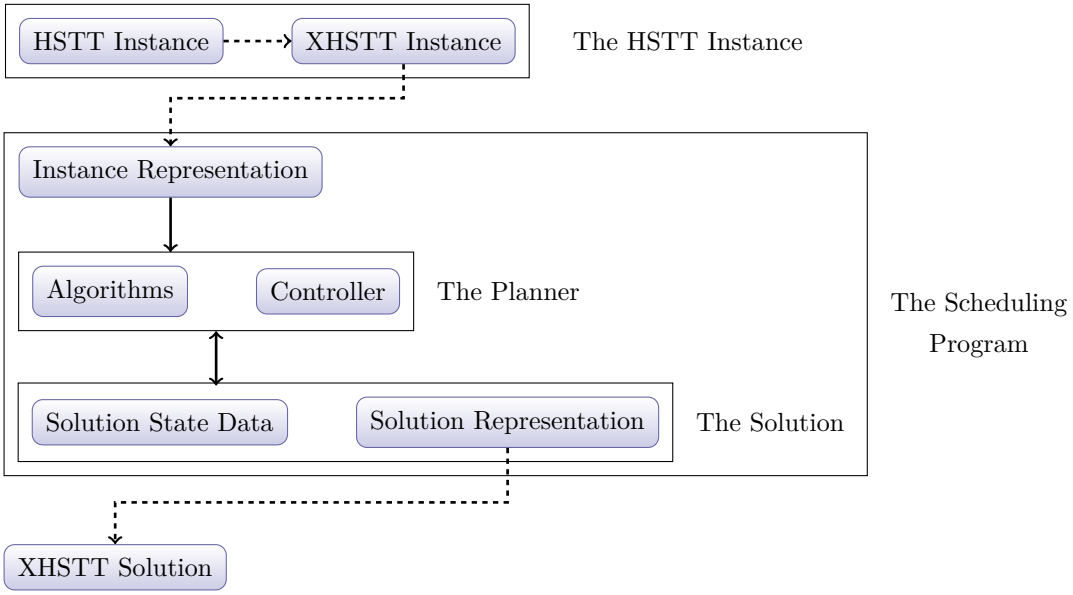


Figure 33: Structure of the Scheduling Program

Several components are used by the scheduling program. These are the instance representation, the planner and the solution. (See Figure 33) At the start of this project, only a framework of a scheduling program was available. It was possible to load the data of an instance into the program. The programming work that was to be done during this project, is the writing of the planner and the solution structure.

The first component, the planner ,consists of algorithms and a controller. Several algorithms have been written that can be executed on each instance. The metaheuristics that have been investigated are the following:

- The SplitEvents algorithm (Section 6.2)
- The Grading algorithm (Section 6.3)
- The Hillclimbing algorithm (Section 6.4)
- The Simulated Annealing algorithm (Section 6.5)
- The Reschedule Resource algorithm (Section 6.6)
- The Ejection Chain algorithm (Section 6.7)

The order in which algorithms are started, the length of their run time and the settings of parameters are configured by the controller.

The second component, the solution, is the component that keeps track of planner actions. For example, if the assigned times of two events are interchanged by an algorithm, the result is stored in the solution. So this component stores the current solution state.

6.2 The SplitEvents algorithm

This algorithm finds an optimal splitting of each Event into a number of SolutionEvents, with respect to the SplitEvents constraint and DistributeSplitEvents constraint. Usually, these constraints leave some freedom to the planner regarding the splitting process. By default, this algorithm maximizes the number of split events. Motivation is that, for some instances, smaller events are easier to fit into a schedule than large ones. However, a larger number of events can lead to a larger computation time. Some instances get a better solution when the instance has a small number of large events instead of a larger number of small events. This can for instance occur due to the SpreadEvents constraint. In that case, the algorithm can be configured to minimize the number of split events instead. See Algorithm 1.

Algorithm 1 The SplitEvents algorithm

Require: An XHSTT instance
  for $I = 0 \rightarrow NumberOfEvents - 1$ **do**
    $TheEvent \leftarrow Event[I]$
    $SplittingList \leftarrow$ The different possibilities for splitting this event
    $TheBestCost \leftarrow$ large number
    for $J = 0 \rightarrow SplittingList.Count - 1$ **do**
      $TheCurrentSplitting \leftarrow SplittingList[J]$
      if CalculateEventCost($TheCurrentSplitting$) $< TheBestCost$ **then**
        $TheBestSplitting \leftarrow TheCurrentSplitting$
      end if
    end for
    Split $TheEvent$ into $TheBestSplitting$
  end for

6.3 The Grading algorithm

This algorithm can be used in two ways. The algorithm can be used for attempts in finding good initial solutions. The algorithm can also be used as a reschedule algorithm, in case the solution process is stuck in a local optimum.

This algorithm operates as follows. First, we create a $ToDoList$, a list of events with which a positive feasibility or objective cost is associated. These can be for example events that have not been assigned a time yet, or events of which the assigned resources attend more than one event at once. We unassign the assigned times of these events. We skip events that have a preassigned time.

Next, we give each event a grade. A grade is an integer 'score'. With this score, we do an attempt to give a measure to an event, which tells how difficult it is to schedule this event. By difficult we mean, that the event is conflicting many times with other events. Conflicting means, that the event is (partly) the cause of a feasibility cost associated with another event. Initially, each event starts with grade 0.

For each event in the $ToDoList$, we try to find a best time for the event. We define a $CostValue$ for the assignment of a time to the event. This $CostValue$ is the cost for assignment of TheSolutionEvent to the time, plus the grades of conflicting events aleady assigned to that time. We assign the time to the best time with the lowest $CostValue$. Conflicting events are unassigned from their times and added to the $ToDoList$. The fact that they blocked the current time assignment and thus had to be unassigned to make way for the current event, gives us a reason to increase their grade. We add 1 to the grade of the conflicting events. This way, events that are removed many times, get a higher grade than others, which gives the planner the preference for removing 'less difficult' events. See Algorithm 2.

Algorithm 2 The Grading algorithm

Require: An XHSTT instance

```
Unassign times of events that have positive Feasibility or Objective costs
TheToDoList ← a random permutation of SolutionEvents
Set Grades of SolutionEvents initially to 0
for I = 0 → NumberOfSolutionEvents − 1 do
  TheSolutionEvent ← TheEventList[I]
  TheBestCostValue ← largenumber
  TheBestTime ← nil
  for J = 0 → NumberOfTimes − 1 do
    TheTime ← Time[J]
    TheConflictingEvents ← The events that cause this event positive Feasibility cost
    TheCostValue ← Cost for assignment of TheSolutionEvent to TheTime
    for K = 0 → NumberOfConflictingEvents − 1 do
      TheConflictingEvent ← TheConflictingEvents[K]
      TheCostValue ← TheCostValue + TheConflictingEvent.Grade
    end for
    if CostValue < TheBestCostValue then
      TheBestCostValue ← TheCostValue
      TheBestTime ← TheTime
    end if
  end for
  Move TheSolutionEvent to TheBestTime
  Remove TheSolutionEvent from TheToDoList
  for J = 0 → NumberOfConflictingEvents − 1 do
    Unassign time of TheConflictingEvent
    Add TheConflictingEvent to TheToDoList
    TheConflictingEvent ← TheConflictingEvents[J]
    TheConflictingEvent.Grade ← TheConflictingEvent.Grade + 1
  end for
end for
```

6.4 The Hillclimbing algorithm

In this algorithm, a local optimum is found by interchanging the times of events. Each iteration, the added cost of interchanging pairs of events is evaluated. On occurrence of a decrease in cost, the events are switched. This is continued until no further decrease in cost is possible. This is the 2-OPT version. There is also a 1-OPT version implemented, which is not presented. See Algorithm 3.

Algorithm 3 The Hillclimbing algorithm

Require: ASolution with any number of SolutionEvents scheduled

```
IsImprovement ← True
while IsImprovement do
  IsImprovement ← False
  for I = 0 → NumberOfSolutionEvents − 2 do
    for J = I + 1 → NumberOfSolutionEvents − 1 do
      SolutionEvent1 ← ASolution.SolutionEvents[I]
      SolutionEvent2 ← ASolution.SolutionEvents[J]
      Switch SolutionEvents
      Evaluate AddedCost
      if AddedCost < 0 then
        Update CostSpecification
        IsImprovement ← True
        Break (continue with outer for-loop)
      else
        Switch SolutionEvents back
      end if
    end for
  end for
end while
```

6.5 The Simulated Annealing algorithm

This algorithm is based on the simulated annealing algorithm proposed in [6]. For each resource that has multiple events scheduled at a certain time, a random event is picked. If a better time is found, such that the total cost of the timetable is improved, the move is made. Otherwise, the event is moved with a certain probability, depending on the increase in total cost. This is the 1-OPT version. There is also a 2-OPT version implemented, which is not presented. See Algorithm 4.

Algorithm 4 The Simulated Annealing algorithm

Require: ASolution with any number of SolutionEvents scheduled
  $TheInitialTemperature \leftarrow T_i$
  $TheTresholdTemperature \leftarrow T_t$
  $TheCurrentTemperature \leftarrow T_c$
  while TheCurrentTemperature > TheTresholdTemperature **do**
    $RandomResources \leftarrow$ random permutation of Resources
    for $I = 0 \rightarrow NumberOfResources - 1$ **do**
      $TheCurrentResource \leftarrow RandomResources[I]$
      for $J = 0 \rightarrow NumberOfTimes - 1$ **do**
        if TheCurrentResource.NumberOfEventsAtTime[J] ≤ 1 **then**
          Resource has no clashes at this time, pick next time J
        end if
        {Resource has a class at time J, find better time for the resource}
        $TheNewTime \leftarrow$ a time with lowest NumberOfEventsAtTime
        {Pick a random TheCurrentResource.SolutionEvent to be moved}
        $ASolutionEvent \leftarrow TheCurrentResource.SolutionEventAtTime[J][Random]$

        Move ASolutionEvent to TheNewTime
        Calculate AddedCost
        if $AddedCost < 0$
        or $\left(AddedCost \geq 0 \textbf{ and } exp\left(\frac{-AddedCost}{TheCurrentTemperature}\right) > Random[0,1]\right)$
        then
          Update CostSpecification
        else
          Move ASolutionEvent back to time J
        end if
      end for
    end for
  end while

6.6 The Reschedule Resource algorithm

This algorithm reschedules, for each resource, the attended events. Each iteration, a resource that has the highest cost is picked. This is the cost that is associated with the resource by the constraints. For an explanation of costs associated with elements, see Section 4. For this resource, all attended events are unscheduled. For each of these events, a best time is assigned in a greedy way. See Algorithm 5.

Algorithm 5 The Reschedule Resource algorithm

Require: ASolution with any number of SolutionEvents scheduled
  $IterationCounter \leftarrow 0$
  while $IterationCounter < N_i$ **do**
    Increase($IterationCounter$)
    $TheCurrentResource \leftarrow$ The resource with highest Cost
    $AnEventList \leftarrow$ The events attended by TheCurrentResource
    for $I = 0 \rightarrow AnEventList.Count - 1$ **do**
      $TheOldTimesList[I] \leftarrow AnEventList[I].Time$
      MoveEventToTime(nil) {Unassign the assigned time}
    end for
    for $I = 0 \rightarrow AnEventList.Count - 1$ **do**
      $TheCurrentSolutionEvent \leftarrow AnEventList[I]$
      $TheBestTime \leftarrow TheOldTimesList[I]$
      for $J = 0 \rightarrow NumberOfTimes - 1$ **do**
        Calculate $AddedCost$ for moving $TheCurrentSolutionEvent$ to time J
        if $AddedCost < 0$ **then**
          $TheBestTime \leftarrow Time[J]$
        end if
        MoveEventToTime($TheBestTime$)
      end for
    end for
  end while

6.7 The Ejection Chain algorithm

The Ejection Chain algorithm is focused on removing of feasibility violations. Ejection chains combined with tabu search seem to be very appropriate for improving schedules [5]. A *ToDoList* is created in which all SolutionEvents are listed that are involved in the clashes of all Resources. In each iteration, the first SolutionEvent in the list is picked. The SolutionEvent is then assigned to the *BestTime* that maximizes the decrease in cost. If any other SolutionEvents which are scheduled at that time are involved in a clash, they are added to the *ToDoList*.

A tabu list is used to avoid cycling. The motivation for also implementing an algorithm using tabu search is, that tabu search based methods [8] often offer the best know solutions to many timetabling problems, when compared to other metaheuristics. Right after a SolutionEvent is assigned to a time, this assignment is declared tabu and added to the tabu list. If the tabu search list becomes too large, the assignments that have stayed the longest in the list are removed. See Algorithm 6.

Algorithm 6 The Ejection Chain algorithm

Require: ASolution with any number of SolutionEvents scheduled
  Initiate ToDoList with events that have positive feasibility cost or are attended by a resource that attends more than one event at the same time.
  while $ToDoList.Count > 0$ **do**
    $TheSolutionEvent \leftarrow ToDoList[0]$
    Remove TheSolutionEvent from ToDoList
    if TheSolutionEvent is not the cause of some feasibility cost **then**
      Pick next event {Event should not be moved}
    end if
    $TheOldTime \leftarrow TheSolutionEvent.Time$
    $BestTotalCost \leftarrow largenumber$
    {Find the best Time $\neq TheOldTime$ to move $TheSolutionEvent$ to}
    for $I = 0 \rightarrow Times.Count - 1$ **do**
      if Time I is tabu for $TheSolutionEvent$ **then**
        Continue with next time
      end if
      Move $TheSolutionEvent$ to Time I
      if $Solution.Cost < BestTotalCost$ **then**
        $BestTotalCost \leftarrow Solution.Cost$
        $BestTime \leftarrow$ Time I
      end if
      Move $TheSolutionEvent$ back to $TheOldTime$
    end for
    Move $TheSolutionEvent$ to $BestTime$
    Add Time I to this event's tabu list
    Add events that are assigned to time I and are attended by a resource that attends more than one event at the same time to the $ToDoList$
  end while

7 The hyperheuristic

In this section, we present the hyperheuristic which we implemented and tested on several HSTT instances.

7.1 About hyperheuristics

Several metaheuristics that are used in this project have been described in Section 33. However, the succes of a metaheurstic is usually limited to particular problems, or even to particular stages of the solution process. Good performance on one problem does not guarantee good performance on another. On top of that, most metaheuristics can be tuned by several parameters. Which parameters are best, depends on the instance. Also, many businesses are not interested in optimal solutions, but rather good quality solutions in a short amount of time for a wide range of problems [9].

This gives motivation for the introduction of a hyperheuristic as introduced by Cowling et al. [10]. They defined a hyperheuristic as "'an approach that operates at a higher level of abstraction than metaheuristics and manages the choice of which low-level heuristic method should be applied at any given time, depending upon the characteristics of the region of the solution space currently under exploration". This means that, given the instance of a problem, a hyperheurstic decides which metaheuristic should be applied at a certain stage of the solution process.

There are different types of hyperheuristics. Some are based on random selection, some are greedy or peckish, some are metaheuristic-based or combined with learning. The hyperheuristic presented here is based on the metaheuristic GA (genetic algorithm) combined with learning [9].

7.2 Description of the hyperheuristic

Here follows the description of a hyperheuristic which we implemented an with which we did several experiments.

The hyperheuristic starts with the initialization of individuals. An individual is defined as a chromosome of metaheuristics, called a column. Each gene is a fixed metaheuristic. The metaheurstics that are used are described in Section 6. Each column has a length and the order of the genes in the column defines the order in which metaheuristics are executed on application of the column in the hyperheuristic. An example of a column is illustrated in Figure 34). First, the Grading algorithm is executed, then the 2-OPT version of Simulated Annealing, then the Reschedule Resource algorithm and last the Hillclimbing.

To start the algorithm, each individual of the first generation is run.(See Algorithm 7). The process is started with an initial solution. For each individual, the algorithms corresponding to the genes are subsequently run on the initial solution. Each algorithm continues with the resulting solution of the previous algorithm. The

performance of a column is measured by the objective and feasibility values of the final solution. The goal is to find the best performing individuals.

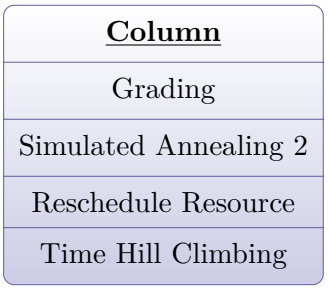


Figure 34: A column

Algorithm 7 The Genetic hyperheuristic algorithm

Require: An XHSTT instance
  Create an initial generation
  for $I = 0 \rightarrow Individuals.Count - 1$ **do**
    $TheIndividual \leftarrow Individuals[I]$
    for $J = 0 \rightarrow TheIndividual.Genes.Count - 1$ **do**
      $TheGene \leftarrow Genes[J]$
      Run algorithm corresponding to gene J
    end for
    Update best columns
  end for
  Select best columns and start reproduction

The next generation is generated by a combination of reproduction (intensification) and new individuals (diversification). The reproduction is done by mutations (Figure 35) and crossovers (Figure 36) of the best performing columns. In a mutation, one or two of the genes are replaced by a randomly chosen other gene. In a crossover, new columns are formed by combining the first half of one column and the second half of another. Diversification is done by performing a crossover between a good column and a randomly generated one. Besides that, some random new columns are added to the new generation.

The overall algorithm of finding a solution to an instance is described in Algorithm 8 and illustrated by a diagram of an experiment that uses this algorithm in Figure 37 of Section 8. This algorithm has a number of stages. Each stage is a certain region of the solution space. The first stage starts with some initial solution. A fixed number of generations of the hyperheuristic is run on this solution. The best column is the one that resulted in the best solution thus far and is used as column in the second part of the stage. The second part is run for a number of iterations. At the first stage, a new solution is created every iteration. Each iteration, the best

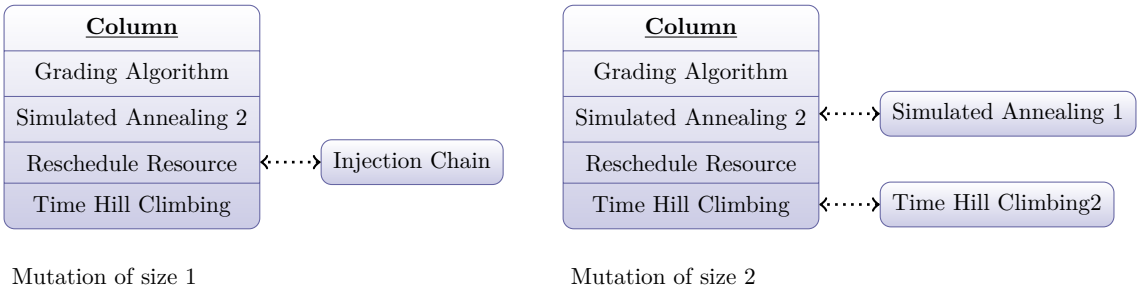


Figure 35: The Mutation of columns

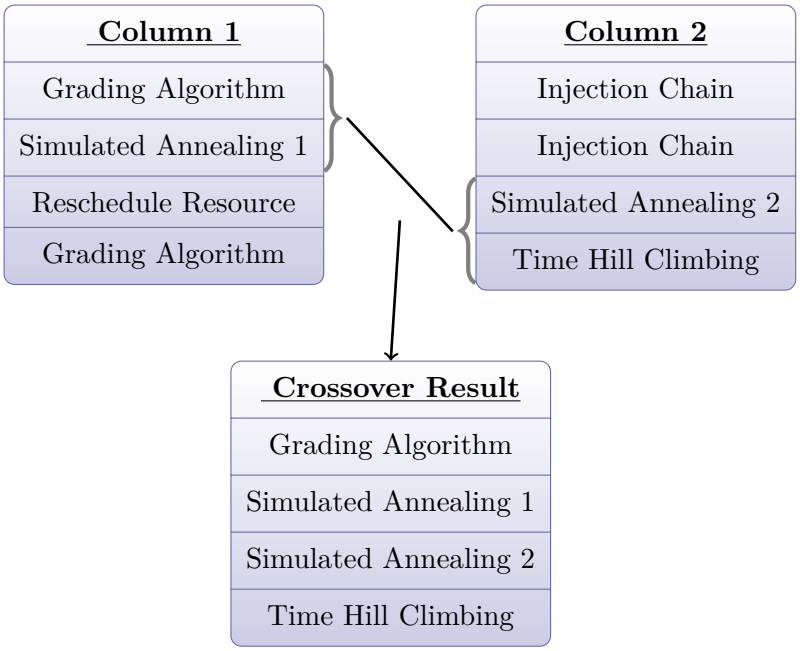


Figure 36: The CrossOver of columns

column is repetitively run, as long as the current solution improves itself during an iteration step.

Algorithm 8 The Overall algorithm

```
Require: An XHSTT instance
  Initialize BestSolution
  for I = 0 → NumberOfStages − 1 do
    Start genetic hyperheuristic algorithm
    for J = 0 → NumberOfIterations − 1 do
      CurrentSolution ← BestSolution
      while IsImprovement do
        IsImprovement ← False
        Run best column on CurrentSolution
        if CurrentSolution has improved then
          IsImprovement ← True
        end if
        Update BestSolution
      end while
    end for
  end for
```

7.3 The next stages

After the first stage, the process is continued with the next stage. The whole process is repeated, with the exception that every initial solution is the best solution obtained in the previous stages. So each next stage, the hyperheuristic finds the column that best improves the best solution of the previous stages. In the second part of the stage, the best column of this stage is repetitively run. Each iteration is initialized with the best solution of previous stages.

7.4 The configurations

The algorithms in Section 6 are used as genes. The Hillclimbing and Simulated Annealing have a number of configurations. Each gene is configured using one of a few predefined configurations. Each gene has a time limit of five seconds. The hillclimbing configurations vary the number of improvements that can be made. The simulated annealing configurations vary the initial and treshold temperature and cooling rate. Both the 1-OPT and 2-OPT versions of the hillclimbing and simulated annealing algorithms are candidates to become a gene. For the hillclimbing and the simulated annealing algorithms there are a number of different types of genes. Each type together with some configuration is one of the genes that can be the realization of a randomly chosen gene in each of the random columns. For hillclimbing, the following combinations of types and configurations are created (See Table 1):

Individual	0	1	2	3	4	5	6	7
Name	H1(0)	H1(1)	H1(2)	H1(3)	H2(0)	H2(1)	H2(2)	H2(3)
Type	1-OPT	1-OPT	1-OPT	1-OPT	2-OPT	2-OPT	2-OPT	2-OPT
Configuration	0	1	2	3	0	1	2	3
Max improvements	10	20	30	120	10	20	30	120

Table 1: The eight different hillclimbing individuals

The motivation is that, it might be better to do just a few hillclimbing improvements with at the cost of a small amount of computation time, than many hillclimbing improvements at the cost of a large amount of computation time.

For simulated annealing, the following configurations have been used (See Table 2):

Individual	0	1	2	3	4	5	6	7	8	9
Name	S1(0)	S1(1)	S1(2)	S1(3)	S1(4)	S2(0)	S2(1)	S2(2)	S2(3)	S2(4)
Type	1-OPT	1-OPT	1-OPT	1-OPT	1-OPT	2-OPT	2-OPT	2-OPT	2-OPT	2-OPT
Configuration	0	1	2	3	4	0	1	2	3	4
Initial Temperature	5	50	500	50	500	5	50	500	50	500
Treshold Temperature	1	1	1	1	1	1	1	1	1	1
Cooling Rate	0.9	0.9	0.9	0.99	0.99	0.9	0.9	0.9	0.99	0.99

Table 2: The ten different simulated annealing individuals

8 Computational results

In this section is described the computational results of a test of the hyperheuristic on a number of instances.

8.1 The Setup

The following instances have been tested:

- Brazil 4
- Brazil 6
- Brazil 7
- Italy 1
- Italy 4
- Greece Preveza 2008
- Greece Patras 2010
- Greece Western 3
- Greece Western 4
- Greece Western 5
- Finland Secondary School 1
- Finland High School
- Finland College
- Kosova 1
- South Africa

Each instance is of different size and uses different types of constraints. For an overview, see Table 3.

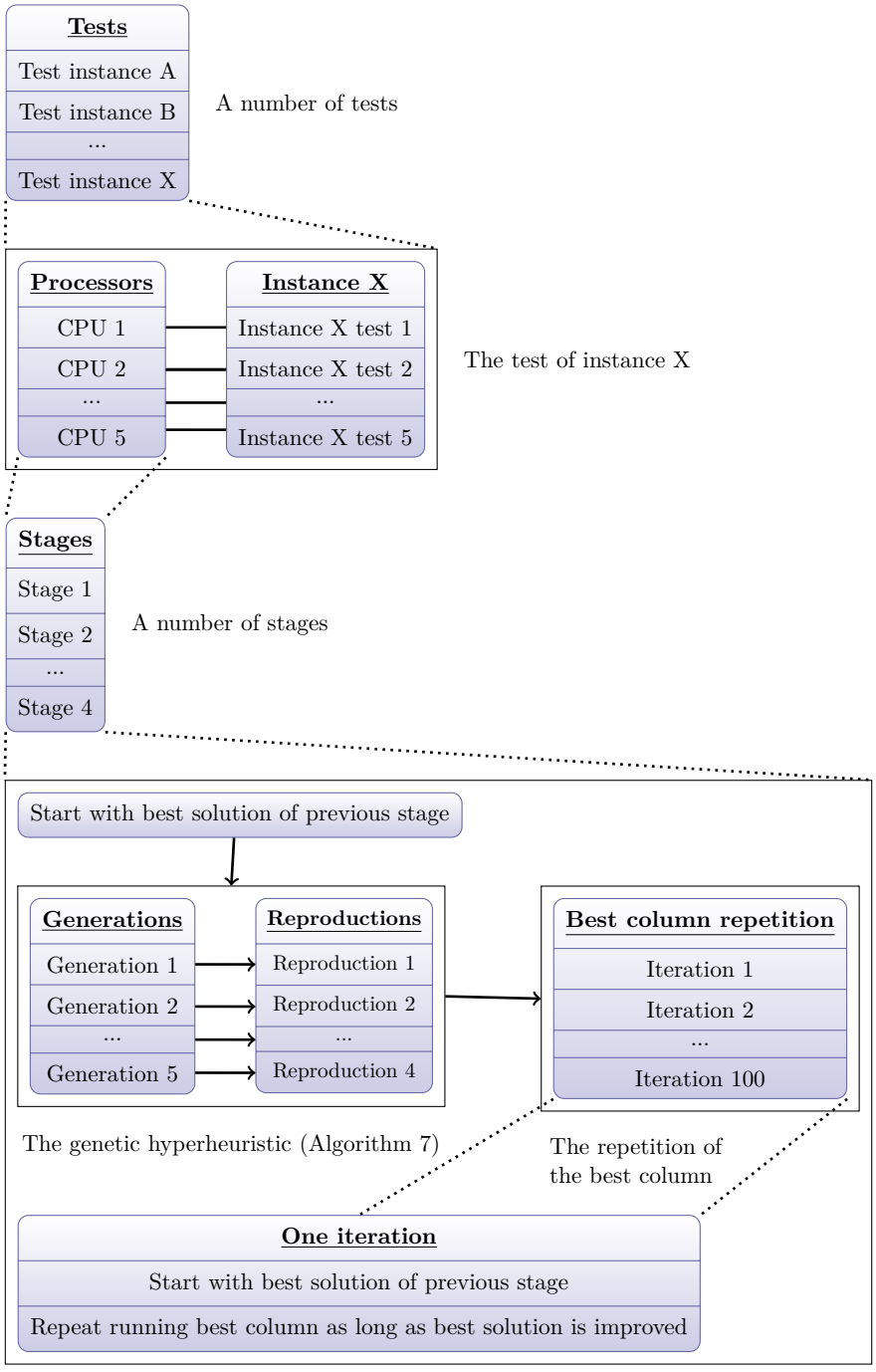
Each instance is run five times, on five different processors. Each run consists of 4 stages. Columns are generated up to the fifth generation using reproduction. The first generation is started with 100 individuals. The best 20 individuals are remembered for the next generation. The next generation is constructed by a mutation of size two of each of the 20 best columns and 20 crossovers between the first 10 best columns. Additionally, there are 20 crossovers between the last 10 of the 20 best columns and 10 random new columns. This results in a generation of size 60. After each column of the fifth generation is executed, the best column of these five

	Number of Resources	Number of Events	Number of Times	Split Events	Distribute Split Events	Assign Resource	Assign Time	Prefer Resource	Prefer Time	Avoid Clashes	Avoid Unavailable Times	Spread Events	Link Events	Limit Idle Times	Cluster Busy Times	Limit Busy Times
Brazil 4	35	127	25	x	x		x		x	x		x		x	x	
Brazil 6	44	140	25	x	x		x		x	x		x		x	x	
Brazil 7	53	205	25	x	x		x		x	x		x		x	x	
Italy 1	16	42	36	x			x		x	x	x	x	x	x		x
Italy 4	99	748	36	x			x		x	x	x	x		x		x
Greece Preveza 2008	97	164	35	x			x		x	x	x	x	x	x		x
Greece Patras 2010	113	178	35	x			x		x	x	x	x	x	x		x
Greece Western 3	25	210	35				x		x	x	x		x			x
Greece Western 4	31	262	35				x		x	x	x	x	x			x
Greece Western 5	24	184	35				x		x	x	x		x			x
Finland Secondary School 1	64	280	35	x			x		x	x	x	x		x		x
Finland High School	41	172	35	x			x		x	x	x	x		x		x
Finland College	111	387	40	x			x		x	x	x	x		x		x
Kosova 1	164	809	62	x	x		x		x	x	x	x		x		x
South Africa	70	278	42	x			x		x	x			x			

Table 3: The elements and constraints used by each instance

generations is used in 100 iterations. Each iteration, the column is repeated until no improvement is registrated after the last algorithm of this column, relative to the solution cost at the start of the first algorithm of the column. After the last iteration, the next stage is started. (See Figure 37).

The first part of each stage is limited by a computation time of four hours. The computation time of the second part is not limited and depends on the number of improvements of each iteration, but takes usually four hours as well. It is part of the setup to take this amount of time (about 24 hours) per instance. Much short computation time results (one second to a few hours) have been achieved in the past, but this test is focused on long computation time results, in order to see what can be achieved in a time limit of about 24 hours.



One stage of the overall algorithm (Algorithm 8)

Figure 37: The test plan

8.2 The results

We present the results by the solution after each stage and the column that is used to get there. Results on two instances are elaborately shown here. Whenever a column is left empty, it means that the corresponding stage did not find any best column or improvement anymore. For each stage part, the computation time and best result of that part is shown. For the first part, the column that found the best result is shown, together with an integer which tells in which generation that best column was found. There are a number of codes used as abbreviations of the used algorithms. They refer to the following ones:

- G: Grading (Algorithm 2)
- H1(x): Hillclimbing 1-OPT, configuration x (Algorithm 3)
- H2(x): Hillclimbing 2-OPT, configuration x (Algorithm 3)
- S1(x): Simulated Annealing 1-OPT, configuration x (Algorithm 4)
- S2(x): Simulated Annealing 2-OPT, configuration x (Algorithm 4)
- R: Reschedule Resource (Algorithm 5)
- I: Ejection Chain (Algorithm 6)

An overview of all results is summarized in Table 14.

Brazil 7

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1480	1795	2132	
Best Column	G	I	$S2(2)$	
	G	$H1(0)$	I	
	S2(0)	$S2(4)$	I	
	H2(0)	I	H1(1)	
	H1(3)	$S2(0)$	S1(2)	
	H2(0)	$H1(1)$	S2(0)	
Found in generation	4	1	1	
Best Result	(0,430)	(0,320)	(0,290)	

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	3198	4040	4014	
Best Result found in iteration	33	1	1	
Best Result	(0,326)	(0,320)	(0,290)	

Table 4: Brazil 7: test 1

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1480			
Best Column	I G S2(2) G S1(2) S2(0)			
Found in generation	5			
Best Result	(0,455)			

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	6047			
Best Result found in iteration	44			
Best Result	(0,353)			

Table 5: Brazil 7: test 2

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1462	1506	2181	
Best Column	S2(4) G S2(0) I R S2(0)	H2(0) I R S1(3) H2(0) H1(3)	S2(2) I S1(3) I S2(3) G	
Found in generation	5	5	1	
Best Result	(0,458)	(0,350)	(0,346)	

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	8960	312	4069	
Best Result found in iteration	61	1	1	
Best Result	(0,371)	(0,350)	(0,346)	

Table 6: Brazil 7: test 3

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1844	2060	2910	2310
Best Column	H1(0)	R	I	R
	G	H1(2)	S2(0)	S1(2)
	G	H1(2)	G	I
	S2(0)	H1(0)	R	H2(1)
	R	R	S1(2)	I
	G	G	S1(0)	H1(1)
Found in generation	5	1	5	5
Best Result	(0,438)	(0,368)	(0,388)	(0,286)

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	2916	148	2126	576
Best Result found in iteration	43	1	21	2
Best Result	(0,370)	(0,520)	(0,298)	(0,274)

Table 7: Brazil 7: test 4

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1521	2046		
Best Column	G	S2(0)		
	H2(0)	H2(0)		
	S1(3)	H2(0)		
	H2(1)	H2(2)		
	G	I		
	S2(0)	I		
Found in generation	5	2		
Best Result	(0,456)	(0,335)		

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	4415	2363		
Best Result found in iteration	84	58		
Best Result	(0,356)	(0,308)		

Table 8: Brazil 7: test 5

Italy 1

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1991	1387	1991	1454
Best Column	I	R	S2(0)	H2(1)
	S1(2)	I	I	S2(3)
	S2(0)	H1(3)	G	H1(3)
	H1(1)	I	G	R
	I	H1(2)	H2(2)	I
	I	H2(2)	I	G
Found in generation	3	3	2	1
Best Result	(0,81)	(0,35)	(0,32)	(0,28)

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1472	17	763	2090
Best Result found in iteration	86	1	4	1
Best Result	(0,45)	(0,34)	(0,29)	(0,28)

Table 9: Italy 1: test 1

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1780	1922	1376	
Best Column	H1(3)	H1(3)	H2(3)	
	I	I	S2(0)	
	H1(3)	S2(4)	S1(3)	
	H2(0)	G	R	
	H1(0)	S2(4)	R	
	S2(0)	S2(2)	S1(0)	
Found in generation	5	1	4	
Best Result	(0,76)	(0,45)	(0,44)	

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1616	6022	757	
Best Result found in iteration	25	1	1	
Best Result	(0,46)	(0,45)	(0,44)	

Table 10: Italy 1: test 2

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1535	1803	1977	1228
Best Column	S1(2)	S2(4)	S2(0)	H2(3)
	I	S2(0)	I	H2(3)
	H1(3)	H1(2)	G	H1(0)
	H2(2)	S2(0)	G	G
	G	H1(1)	I	G
	G	H2(3)	S1(2)	H2(2)
Found in generation	1	4	1	1
Best Result	(0,97)	(0,43)	(0,32)	(0,28)

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	223	3961	745	104
Best Result found in iteration	2	40	3	1
Best Result	(0,66)	(0,34)	(0,312)	(0,28)

Table 11: Italy 1: test 3

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1347	1742	1716	
Best Column	I	S2(2)	S2(0)	
	H1(3)	S1(1)	H1(0)	
	S2(0)	S1(2)	S2(0)	
	I	G	S2(0)	
	H1(1)	G	S1(0)	
	G	I	H2(1)	
Found in generation	5	2	3	
Best Result	(0,47)	(0,42)	(0,35)	

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	1291	2024	1927	
Best Result found in iteration	41	23	1	
Best Result	(0,44)	(0,40)	(0,35)	

Table 12: Italy 1: test 4

Part 1	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	2000	1269	1619	
Best Column	G	S1(0)	H1(2)	
	H2(0)	S2(0)	S1(3)	
	S2(2)	I	G	
	H1(3)	S1(0)	H1(1)	
	S2(0)	G	G	
	H1(0)	H1(0)	R	
Found in generation	4	20	1	
Best Result	(0,50)	(0,40)	(0,39)	

Part 2	Stage 1	Stage 2	Stage 3	Stage 4
Computation time (sec.)	3966	713	96	
Best Result found in iteration	75	1	1	
Best Result	(0,41)	(0,40)	(0,39)	

Table 13: Italy 1: test 5

Overview

Instance	Result						
	CPU 1	CPU 2	CPU 3	CPU 4	CPU 5	Best	Best known
Brazil 4	(6,163)	(4,165)	(5,161)	(6,214)	(5,177)	(4,165)	(0,121)
Brazil 6	(0,177)	(0,193)	(0,172)	(0,170)	(0,169)	(0,169)	(0,209)
Brazil 7	(0,290)	(0,353)	(0,346)	(0,274)	(0,308)	(0,274)	(0,330)
Italy 1	(0,28)	(0,44)	(0,28)	(0,35)	(0,39)	(0,28)	(0,28)
Italy 4	(0,205)	(0,402)	(0,378)	(0,176)	(0,185)	(0,176)	(0,84)
Greece Preveza 2008	(0,134)	(0,146)	(0,150)	(0,131)	(0,150)	(0,131)	(0,28)
Greece Patras 2010	(0,121)	(0,151)	(0,151)	(0,147)	(0,144)	(0,121)	(0,0)
Greece Western 3	(0,13)	(0,22)	–	–	–	(0,13)	(0,7)
Greece Western 4	(0,31)	(0,28)	–	–	–	(0,28)	(0,8)
Greece Western 5	(0,11)	(0,11)	–	–	–	(0,11)	(0,0)
Finland Secondary School 1	(0,144)	(0,120)	(0,139)	(0,154)	(0,157)	(0,120)	(0,106)
Finland High School	(0,42)	(0,58)	(0,70)	(0,74)	(0,80)	(0,42)	(0,1)
Finland College	(0,119)	(0,160)	(0,84)	(0,81)	(0,150)	(0,81)	(0,0)
Kosova 1	(0,270)	(0,282)	(0,279)	(0,264)	–	(0,264)	(0,952)
South Africa	(2,16)	(4,8)	(2,20)	(3,19)	(1,19)	(1,19)	(0,2)

Table 14: Overview of computational results

9 Conclusion and recommendations

9.1 Conclusion

The first goal of this master project was to extend a progam in order to facilitate the application of algorithms on instance data. We extended the program and it is now possible to write any algorithm or planner and implement it easily into the code. We created a solution structure, which keeps track of any action of the planner and the algorithms. The result of the actions is represented by the solutions. We implemented the evaluation of constraints, such that the quality and feasibility of any solution can be evaluated.

The second goal was the creation of algorithms. We have written a SplitEvents algorithm, which does some preliminary work. It splits the events and links them. We have created a Grading algorithm, which tries to assign all events to a promising time. We have created two Hillclimbing and two Simulated Annealing algorithms. Beside the Simulated Annealing algorithms, we have thought of other algorithms that could help escape local optima. We have written the Reschedule Resource and Ejection Chain algorithm.

The third goal was to combine these algorithms together with a planner. We have created a Hyperheuristic algorithm, which can manage the choice of which heuristic method should be applied at any given time. It also manages the parameters of each algorithm.

We generated computational results on a number of instances. We have the following conclusions on these results:

- Solution cost
  We got a feasible solution to almost every instance. Brazil 4 and South Africa got near-feasible, in the sense that only a few required constraints were violated. Together with the objective values, the cost of the solutions got close to known solutions. For illustration, the solution process of the Kosova 1 instance starts it's initial solution with cost (1912,0), gets cost (66,29007) after the first generation, but ends with a solution cost of (0,264). The best known solution thus far is (0,952). The solution process of instance Italy 4 gets a solution cost of (1,14884) after generation 1, (0,3413) after generation 2, (0,486) after generation 3 (0,483) after generation 4 and ends with (0,176). The best known solution is (0,84).

  We see that for each instance, the final solution costs are not very different from different runs on the same instance. This is a sign that for a single run, the probability of gaining a reasonable perfomance is very high.

  Some solutions have improved the best known thus far, others got very close to the best known solution, some got somewhat worse and others even did not get feasible. We do not know why these solutions did not get feasible, but

since this occured in every run of these instances, we think that the algorithms are just nog good enough to get these solutions feasible.

- Columns
  We can not conclude much on the found columns. It is a good thing that they are different at each stage of the solution process, this shows the ability of the hyperheuristic to adapt the current algorithm combination to the position of the solution process in the solution space. On the other hand, the columns are also quite different from other runs on the same instance. This tells us, that with the current collection of algorithms, it does not matter very much which algorithms are used in a column. It seems that having just a combination of diverse algorithms, both deterministic and stochastic, is a good choice for proceeding to better solutions in the solution process, regardless the kind of mixture and order of the algorithms. However, it is striking that the Grading algorithm is chosen to be part of the best column of the first generation many times. We expected this, because this algorithm was created with the intention to create a good, hopefully feasible, initial solution. We observed in other experiments that it is indeed useful to include this algorithm at the beginning of the solution process.

- The hyperheuristic
  We observed that for every instance, we could just start the program without any manual configurations or adaptations to the program. Although each instance had a different number of resources, events and times and had a different collection of constraints, the hyperheuristic could adapt automatically to the instance, yielding the presented result. This is an advantage, since this means that a user of the program can get solutions of reasonable quality without any knowledge of the program or the instance. However, for some instances, there may be some adaptations to the algorithm configurations possible that can not be done by the hyperheuristic. We do not expect that this hyperheuristic with the current algorithms will outperform heuristics that are created specifically for a certain instance, making use of every possible advantage of the instance specific information.

9.2 Recommendations

We have the following recommendations for improving the solution process of the HSTT problem using a hyperheuristic.

First, the hyperheurstic could use some better algorithms. The current algorithms are pretty straightforward. The perform reasonably well, but they make no use of certain instance specific information. An example is the use of weight factors of events, where each weight factor is measure of the priority of scheduling certain events. This can be used to distinguish between difficult and less difficult events. An event can for example be difficult if a difficult resource is assigned to it, that is a resource that is assigned to many events but is only available at very few hours.

Besides that, there could be experimented with more and different predefined configurations. Currently, the hyperheuristic can choose from only a few configurations, which are just chosen by common sense but there may be better ones. The simulated annealing may be improved by different cooling schedules.

Secondly, the hyperheuristic itself could be tested in more ways. In the experiments in this master thesis, we gave the hyperheuristic a lot of time (24 hours) to do calculations for a single run. We know that for a very short time, in the sense of a few minutes, already some reasonable results are achieved. Research can be done on the performance of the hyperheuristic given the amount of computation time available. In combination with some better algorithms, we may get good results in a short amount of time. There could also be experimented with the length of the columns.

Thirdly, we think that the current hyperheuristic settings is too much about intensification. Every next generation continues with the schedule of the previous generation. At some point, the algorithms always get stuck, in the sense that even a combination of algorithms can not improve the schedule any more. A suggestion for diversification is to introduce a solution tree. The root of the tree is the highest level of the tree, it represents the solution in which nothing is assigned yet. The first level is the nodes connected to the root, they represent the first solutions. The second level consists of nodes that are connected to nodes at the first level and represent the solutions that are the result of continuing with the solutions of the first level, and so on. If we are stuck at a certain level of the tree, we can always go back and continue with a previous found solution from a higher level. If we are stuck at all, we can start at the root again. We give the computation on every node of the tree only a short computation time, in order to search at much different parts of the solution space. We might even combine good parts of different solutions, but it may be hard to find good parts that are not conflicting whenever they are combined into one solution.

References

[1] "Benchmarking project for (High) School Timetabling." `http://www.utwente.nl/ctit/hstt/`. Accessed: 19/06/2012.

[2] G. Post, S. Ahmadi, S. Daskalaki, J. Kingston, J. Kyngas, C. Nurmi, and D. Ranson, "An XML Format for Benchmarks in High School Timetabling," *Annals of Operations Research*, vol. 194, no. 1, pp. 385–397, 2010.

[3] A. Schaerf, "A survey of automated timetabling," *Artificial Intelligence Review*, vol. 13, pp. 87–127, 1995.

[4] E.-J. Krijgsman, B. Rorije, S. van Veldhoven, and B. de Wilde, "Schoolroosters maken met Tabu Search." Bachelor assignment, June 2006.

[5] P. De Haan, R. Landman, G. Post, and H. Ruizenaar, "A case study for timetabling in a dutch secondary school," in *Proceedings of the 6th international conference on Practice and theory of automated timetabling VI*, PATAT'06, (Berlin, Heidelberg), pp. 267–280, Springer-Verlag, 2007.

[6] D. Zhang, Y. Liu, R. M'Hallah, and S. C. Leung, "A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems," *European Journal of Operational Research*, vol. 203, no. 3, pp. 550 – 558, 2010.

[7] J. Kingston, "Resource assignment in high school timetabling," *Annals of Operations Research*, vol. 194, no. 1, pp. 241–254, 2010.

[8] H. Santos, L. Ochi, and M. Souza, "An efficient tabu search heuristic for the school timetabling problem," in *Experimental and Efficient Algorithms*, vol. 3059 of *Lecture Notes in Computer Science*, pp. 468–481, Springer Berlin / Heidelberg, 2004.

[9] K. Chakhlevitch and P. Cowling, "Hyperheuristics: Recent developments," in *Adaptive and Multilevel Metaheuristics* (C. Cotta, M. Sevaux, and K. Sörensen, eds.), vol. 136 of *Studies in Computational Intelligence*, pp. 3–29, Springer Berlin / Heidelberg, 2008.

[10] P. I. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, (London, UK, UK), pp. 176–190, Springer-Verlag, 2001.