A Timeslot-Filling Heuristic Approach to Construct High-School Timetables

Michael Pimmer and Günther R. Raidl

Abstract This work describes an approach for creating high-school timetables. To develop and test our algorithm, we used the international, real-world instances of the *Benchmarking project for (High) School Timetabling*. Contrary to most other heuristic approaches, we do not try to iteratively assign single meetings (events) to timeslots. Instead, we repeatedly choose a not entirely occupied timeslot and aim at simultaneously assigning the most suitable set of meetings. To improve and diversify the solutions, a heuristic that deletes and reassigns certain timeslots, events or resources is applied and combined with a hill-climbing procedure to find suitable parameters for grading constraints. Experimental results indicate the competitiveness of this new approach.

1 Introduction

The task of high-school timetabling is to assign events – normally class–teacher meetings – to rooms and timeslots of a weekly schedule. Dependent on the country and institution of origin, the requirements and specifics of the problem vary drastically. As no commonly accepted, international benchmark instances existed until recently, most scientists have been working with specific local instances or strongly simplified models as the classical Class-Teacher Timetabling Problem (CTTP) described by Gotlieb [8] in 1974. Thus, most of this work is hard to compare or of limited practical relevance.

In 2007, the **Benchmarking Project for (High) School Timetabling** was launched to settle this issue. Based on an XML file format and a well-defined evaluation function, it currently provides more than 20 real-world instances from various coun-

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9-11/1861, A–1040 Vienna, Austria
e-mail: michael@pimmer.info, raidl@ads.tuwien.ac.at

tries. The XML-format describes *resources* (e.g. teachers, students, rooms) which can be part of *events*. An event is a meeting that usually requires some resources and should be assigned to a timeslot. If any resource out of a set of resources – e.g. any English teacher – shall be assigned to an event, this is called *open role*. Further information about the project is available in [16] and on the project website[1].

While most existing heuristics to solve school timetabling problems are based on an iterative process that assigns single events to timeslots, we follow here the concept of repeatedly choosing a not entirely occupied timeslot and assigning a promising larger set of suitable meetings at the same time. In addition an improvement procedure that deletes and reassigns timeslots, events, or resources is applied in combination with a hill-climbing procedure for adapting parameters controlling the grading of constraints. This new approach was specifically developed with the instances of the Benchmarking Project for (High) School Timetabling and general applicability in mind.

The next section gives an overview on related work. Our approach is described in Section 3, Section 4 presents and discusses achieved results, and conclusions are drawn in Section 5.

2 Related Work

Concerning the problem complexity, the classical CTTP was shown to be NP-complete when any unavailabilities are given [7]. Relaxing the restrictive definition of meetings and adding some common constraints from real-world school timetabling problems introduces significant additional complexity. Kingston and Cooper [10] identified five NP-hard subproblems, and Willemen [20] extended these by two more.

A broad range of approaches has been applied to high-school timetabling. For extensive information on existing methods, the reader is referred to well-known surveys [3, 17, 14] and to the international conferences *Practice And Theory of Automated Timetabling (PATAT)*[2] [4, 2, 12] as well as to the EURO working group on automated timetabling *EURO-WATT*[3].

The most common solving strategy is to iteratively assign single events, combined with a backtracking on dead ends. The less explored alternative we consider in this work is to create a timetable by repeatedly filling selected timeslots. This approach was first mentioned by Schmidt and Ströhlein [18] in 1972. Unfortunately, their formulation of a CTTP as a vertex coloring problem is not always applicable to real-world instances, because many constraints have to be considered additionally to the coloring-problem. For example, problems arise with events of longer duration requiring multiple timeslots, especially if it is not determined how an event can

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

[2] http://www.asap.cs.nott.ac.uk/patat/patat-index.shtml

[3] http://www.asap.cs.nott.ac.uk/watt/

be split, or in case of multiple open roles requiring the same resource. Considering the vertex coloring formulation, not only the weights and possible colors of edges would change during the solving procedure, but nodes and edges may even appear or disappear.

In 2002, Abraham and Kingston [1] applied the timeslot-filling approach to an Australian instance. They pre-calculate possible compatible sets of events. As there are too many sets, smaller events which are supposed to be easier to schedule are omitted. Then, a set is chosen for each timeslot in a way to cover all required events. Finally, the omitted events are scheduled. Instead of continuing this work, effort was put in the creation of the KTS High School Timetabling System [9], which yielded better results.

The KTS can be considered a hybrid approach. Such approaches lie in between assigning single events and filling timeslots at once. The goal is to break down the problem into easier subproblems by grouping events, without losing too much flexibility later on. This can be done by pre-assigning events to days, e.g. as described in [6], before assigning them to specific timeslots. Kingston [9] breaks down the problem by grouping events to tiles. The tiles are scheduled separately and joined afterwards to obtain the final timetable. This approach yields good results for Australian real-world instances in short runtimes and can to some degree be extended to be applicable to instances of other countries.

Some very Large-Scale Neighborhood Search Techniques (VLNS) delete and reassign a major part of the solution. They were already applied to timetabling problems, e.g. by Meyers and Orlin [13]. The refilling-strategies described in chapter 3.3 also belong to this kind of search technique.

3 Timeslot-Filling Heuristic

The central building block of our algorithm is to fill a timeslot with a suitable set of events. Consequently, we call it *Timeslot-Filling Heuristic* (TFH). We iteratively fill non-full timeslots as shown in Figure 1 and further detailed in the subsequent sections.

The higher-level strategies based on this building block are described in Section 3.3.

Because of the multitude of constraints considered in the benchmark project's instances and thus in our algorithm, we cannot present all details here but refer to the first author's master thesis [15] for an exhaustive description.

Fig. 1 Repeatedly filling timeslots

3.1 Grading of Events

The grade of an event represents the favorability of holding it in a chosen timeslot. It is defined as the sum of the grades of the constraints that apply to this event, that apply to any resource it requires (or that can fill any of its open roles), and that apply to any event group this event belongs to. Moreover, we calculate and add grades that are not directly related to constraints, but instead aim at completely filling the timetable (*timetable-filling grades*).

3.1.1 Constraint-Related Grades

When dealing with constraint-related grades, we try to maintain a direct connection to their weight and cost function. If assigning an event would entail a penalty by violating a soft-constraint, a negative grade of exactly this penalty will be added to the grade of this event. If an assignment helps avoiding future constraint-violations, positive grades are assigned. When having positive as well as negative grades, they are added up. The benchmark instances allow soft-constraint weights of up to 1000. To represent the urge of avoiding hard-constraint violations – which originally do not have weights themselves – we assign such constraints a weight of 10000. The balance between hard- and soft-constraints is maintained with a parameter *soft-constraint level*. All positive and negative grades arising from soft-constraints are multiplied with this value. This allows adjusting the influence of soft-constraints to the grades with one parameter, as we will further explain in Section 3.3.

We will now demonstrate the evaluation of a constraint on the example of the **SpreadEventsConstraint**, which is – besides the resource availability – the most important of the 15 existing constraints. First we will calculate a **ratio**, which expresses the urgency of an assignment independent of the weight of the constraint or whether it is hard or soft. The second step is to transform the ratio to the final grade of this constraint by considering the weights and other parameters such as the soft-constraint level.

Calculating the Ratio

The SpreadEventsConstraint is supposed to limit the usage of a set of events within certain timegroups. Typically timegroups represent days and the events form a course so that no more than one event of this course should be held on each day. The constraint allows defining a *minimum* and a *maximum* number of events per time group (day). We will only discuss the *maximum* here, where an event with a duration of more than one timeslot still counts as one assignment.

As *current time group* we understand the intersection of the time groups the constraint is applied to and the time groups the timeslot we are currently grading

belongs to, which normally is the time group representing the current day (i.e., the day the timeslot we are currently grading belongs to).

Table 1 explains the variables we are going to use for calculating the ratio.

Table 1 Variables used for calculating the SpreadEventsConstraint-ratio

Variable	Explanation
maximum	number of allowed events per time group
cA	**c**urrent **A**ssignment: number of existing event-assignments within the current time group
pendingAssignments	nr of pending (open) event-assignments of the constrained event group
possibleAssignments	possible assignment in all time groups (also current) without violating the *maximum*
pACG	**p**ossible **a**ssignments in **c**urrent time**g**roup permitting maximum-violations

Positive grades are only applied if the maximum is not yet reached ($cA \geq maximum$). Equation (1) shows the calculation of the ratio. The left part $\frac{pendingAssignments}{possibleAssignments}$ is independent of the current time group and can be considered as a general urgency/pressure: It compares the number of pending assignments with the slots they can be assigned to without violating the maximum. The right part $\min\left(\frac{maximum-cA}{pACG}, 1\right)$ represents the urgency of assignments in the current time group: It relates the assignments missing to reach the maximum with the number of timeslots that are available for such assignments.

$$ratio = \frac{pendingAssignments}{possibleAssignments} \cdot \min\left(\frac{maximum - cA}{pACG}, 1\right) \tag{1}$$

If there are several possibilities (”sub-events“) of assigning the duration of an event, we will calculate the ratio using the highest number of events. Having more *possibleAssignments* makes it more difficult to not violate the *maximum*, so we anticipate this case.

Example

Assume $maximum = 1$, a total number of four events, and the event availabilities as given in Figure 2. White slots indicate that at least one of the events is available, black slots mark unavailability, and grey slots indicate that an event of this group is already assigned to the respective timeslot. The value of *pendingAssignments* is two, because two of the four events are already assigned (We-1 and Th-3). We have a *possibleAssignment* of two, one on Monday and one on Tuesday. This would imply the following ratios:

Mo-1: $ratio = \frac{2}{2} \cdot \min(\frac{1-0}{5}, 1) = 0.2$
Tu-1: $ratio = \frac{2}{2} \cdot \min(\frac{1-0}{2}, 1) = 0.5$
Th-1: $ratio = \frac{2}{2} \cdot \min(\frac{1-1}{4}, 1) = 0$

As the maximum for Th-1 is already reached with $cA \geq maximum$, we would not apply any positive grade anyhow.

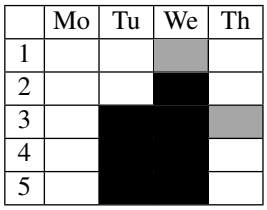

Fig. 2 Event availabilities

Calculating the final grade

The final grade combines the ratio with the weight and type (soft or hard) of the constraint. It is calculated as follows:

$$\begin{aligned} \text{hard-constraint.: } & grade = ratio^{exponent} \cdot 10000 \cdot externalWeight \\ \text{soft-constraint: } & grade = ratio^{exponent} \cdot weight \cdot externalWeight \cdot softConstraintLevel \end{aligned} \tag{2}$$

Ratios usually are values between 0 and 1. The differences of a ratio can be either emphasized (stretched) by applying an $exponent < 1$, or reduced (squeezed) by $exponent > 1$. The *weight* is given by the constraint within the instance, whereas $externalWeight$ is defined by us to adjust the overall importance of each constraint. These parameters will be adjusted by a hill-climbing procedure, see Section 3.3.

Continuing our example, assume having an external weight of 0.5, an exponent of 2, and a hard SpreadEventsConstraint. For Mo-1 this leads to $grade = 0.2^2 \cdot 10000 \cdot 0.5 = 200$, and to $grade = 0.5^2 \cdot 10000 \cdot 0.5 = 1250$ for Tu-1. When grading these timeslots, each event of the constrained event group will have the grade of the respective timeslot added.

3.1.2 Timetable-Filling Grades

To assist creating complete timetables, we calculate three more grades:

Bin-Packing When having tight resource-assignments, a problem similar to bin-packing arises. We calculate a grade that aims at gaplessly assigning such resources. This is done by favoring events and event-durations that maintain the possibility of gaplessly assigning resources.

Unassignment-Bonus Resources and events that lately failed to be assigned get an additional grade.

Course-Urgency The instances do not provide something equally to a course: Courses can be either represented by single events that have to be split or by a set of multiple events. We therefore group events that require the same resources to courses. Then, a grade is calculated favoring courses that have fewer possibilities left to assign their events, considering their duration and other constraints that impede assignments to certain timeslots.

3.2 Clique-Search

Having graded all events for the chosen timeslot, we are now looking for the most favorable set of events that can be held together. We first construct a weighted graph out of the graded events. The nodes correspond to the events having their grade as weight, and nodes are connected by edges if the events can be held simultaneously, which usually means that they do not have any resource in common. Additionally to the weight we store the *depth* of nodes, which is the number of teachers and rooms an event requires. Having open roles leads to *resource limits*, which is the number of resources of a certain type that each solution has to respect to be valid. For example, a feasible set of events must not require more gym-rooms than there are available in the given timeslot.

Apart from the resource limits and the depth, the search for the most favorable set of events corresponds to the **maximum-weight clique problem**. Because of the additional constraints and the instances' graphs with densities of up to 0.96% and 950 nodes, we use a custom heuristic.

A *peer* of a clique is a node that is not part of the clique but connected to all nodes of the clique. Our approach is to repeatedly *expand* cliques, which means we create new cliques by adding peers to an existing clique. When expanding a clique, we will create one new clique for each of its peers. We start at cliques of size 1, each containing one single event. Obviously, the order of selecting the clique to extend next is crucial. Condition (3) shows the basic idea of how we sort our cliques. The *depth* of a clique is the sum of the depth values of all its nodes, and the *peerDepth* of a clique is the total depth of all its peers. As long as the condition holds, a clique can still exceed the currently known maximum.

$$grade + \frac{grade}{depth} \cdot peerDepth > currentMax \tag{3}$$

This condition is incorporated into our *internGrade*, which determines the order of choosing cliques to extend. Equation (4) shows the calculation of the *internGrade*. The smaller the cliques are, the more the *internGrade* tends to over-estimate the reachable maximum, which we try to compensate with parameter c. As condition (3) aims at finding the optimum, we introduce *gradeMultiplier* (default set to 10) to focus on good-graded cliques instead of cliques having many peers. This again prevents expanding too many small cliques, and thus to expand many cliques that are highly unlikely of being further chosen later on.

$$internGrade = \left(grade \cdot gradeMultiplier + \left(peerDepth \cdot \frac{grade}{depth} \right) \right)^{1+c \cdot depth} \tag{4}$$

The cliques are stored in an array of heaps, so the *internGrade* is only calculated for the highest graded clique of each depth. At any moment, we only allow expanding cliques of certain depths, e.g. cliques that contain three to six normal-sized events. With increasing runtime, this range is shifted towards higher depths. Factor c is

adapted automatically with the goal of equally choosing the cliques of the depths we currently permit; it is increased when too many cliques are chosen from low depths and decreased in case of too many cliques are chosen from high depths.

After having found a promising clique with this heuristic, the open roles have to be closed. We construct a bipartite graph out of the open roles of the chosen events and the resources that possibly fill these roles. Then, a maximum-cardinality maximum weight matching is determined. If there are resources that belong to multiple resource-groups, e.g. the sets rg_1 and rg_2, filling all open roles can be impossible when $(rg_1 \not\supseteq rg_2) \wedge (rg_1 \not\subseteq rg_2)$ and $rg_1 \cap rg_2 \neq \emptyset$. In such cases, we reduce the resource limits and repeat the search until filling all open roles succeeds. Finally, the events are assigned to the chosen timeslot.

3.3 Higher-Level Strategies

Apart from the grading parameters, we modify the search by varying how to choose the next timeslot that is filled: simply incremental or by choosing the first not entirely full timeslot of the day that has the fewest events assigned. When the pending workload of a resource gets unassignable, a local backtracking is applied. Whether or not these methods are used is controlled by parameters, too. Together with the grading parameters, they form a **parameter set**.

A given parameter set is tested by repeatedly applying **refilling strategies**, as shown in Figure 3. We implemented several different strategies, each having a different focus: Refilling timeslots that have few events assigned, consecutively deleting and refilling timeslots or days, re-assigning resources, events or event-groups (courses) that cause high penalty or that we were unable to assign completely.

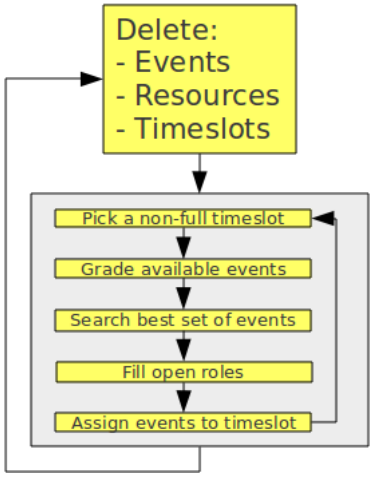

Fig. 3 Refilling strategies

After initially filling the timetable, we iteratively select and apply one of these strategies on a random basis. Hereby, each strategy has an individual selection probability which is adapted according to its success in previous applications.

On top of the refilling strategies, a **hill-climbing** procedure is applied. The goal of the hill-climbing is to find the most suitable parameter set and soft-constraint level for a given instance. To focus on the most relevant parameters at a given stage of the solving procedure, we introduced the *soft-constraint level* already mentioned in Section 3.1. All grades arising from soft-constraints get multiplied with this parameter. All hard-constraints and grades that assist in creating a completely filled timetable

(e.g. the bin-packing and the course-urgency) are not affected by the soft-constraint level. Setting the soft-constraint level to zero completely deactivates considering any soft-constraint – the grades will exclusively aim at creating a complete timetable. Increasing the soft-constraint level leads to a higher influence of soft-constraints to the overall grades. This helps avoiding penalty, but complicates creating completely filled timetables. The hill-climbing then consists of two parts:

The **first part** is to find suitable parameter sets as starting points for the later hill-climbing. We try to find hard-constraint parameters that reliably fill the timetable while maximizing the extent of considering soft-constraints. To do so all soft-constraint parameters are set to default values; when creating a random parameter set these values are forbidden to be changed. Instead, all hard-constraint parameters are allowed to change. We start at a soft-constraint level of zero and create a random parameter set. This set is tested by applying the before-mentioned refilling strategies. If we achieve a valid timetable without any hard-constraint violations, we increase the soft-constraint level. Otherwise, a new random parameter set is created and evaluated.

The **second part**, the hill-climbing procedure, inverts the parameters that are allowed to change: The hard-constraint parameters and the soft-constraint level are fixed, we only allow changes of soft-constraint parameters at this stage. As starting point we pick the parameter set with which the lowest-penalized full timetable was created during the first part. Then, the climbing is started by iteratively changing some of the parameters and evaluating the changed parameter set. If we were not able to create a full timetable in the first part, we continue ignoring the soft-constraints by only allowing changes of hard-constraint parameters.

4 Results

We compare the timeslot-filling heuristic (TFH) described above with the more common event-assignment heuristic (EAH) which iteratively assigns the most urgent event. We simulate the latter by grading all timeslots and assigning the event that achieved the highest grade to the respective timeslot. This allows us to apply exactly the same grading procedure and higher-level strategies to both algorithms. Normally, event-assigning first picks the most urgent event (only considering hard-constraints), and then assigns it to the most suitable timeslot determined by the soft-constraints. Contrary to this we consider both the hard- and soft-constraints at the same time. We did not implement any kind of additional backtracking for EAH. The algorithms were implemented in Python. All runtimes were measured using an Intel Core 2 Duo with 2.55GHz (with the program running on one core only) with 3GB RAM.

We tested our algorithms on the real-world instances of the Benchmarking Project and on artificial HDTT (hard timetabling) instances from the OR Library as discussed in the next paragraphs.

Real-World Instances

Table 2 shows the best results we were able to achieve. As these are probably the first published results for the benchmarking project instances, the only source of solutions are the solutions delivered with the instances, which are listed in column *Existing Solution*. These solutions originate from previous scientific work and usually were provided by the contributor of the respective instance. The columns TFH and EAH present the results for the timeslot-filling heuristic and event-assigning heuristic, where the first value denotes the hard-constraint violations, and the latter value in parenthesis indicates the penalty arising from soft-constraint violations.

The column *runtime TFH* presents CPU-times of the **initial filling** of the timetable using TFH. The runtime of evaluating a parameter-set depends on the – stochastically chosen – refilling strategies. For 20 refilling-rounds, it usually is 20 to 30 times the initial filling time. The implementation of EAH has a lot of overhead because we use much code that we implemented for TFH: For assigning one event, all events in all timeslots are graded without applying any delta-functions, which makes runtime-comparisons valueless.

Table 2 Results for real-world instances

Instance	*Country*	*TFH*	*EAH*	*Existing Solution*	*Runtime TFH*
Brazil1	Brazil	**0 (1)**	0 (101)	0 (104)	1.8s
Brazil4	Brazil	**4 (1728)**	18 (1070)	-	46s
Brazil5	Brazil	**0 (2375)**	0 (5054)	-	59s
Brazil6	Brazil	**0 (2218)**	0 (2376)	-	40s
Brazil7	Brazil	0 (6581)	**0 (6277)**	-	85s
FinHigh	Finland	0 (248)	**0 (193)**	-	20s
FinSec	Finland	**0 (216)**	0 (279)	-	40s
FinColl	Finland	5 (424)	**4 (813)**	-	222s
GreeceHigh	Greece	**0 (0)**	**0 (0)**	-	90s
Patras	Greece	0 (163)	0 (30)	**0 (0)**	74s
Preveza	Greece	0 (138)	0 (62)	**0 (0)**	68s
Italy1	Italy	0 (138)	0 (134)	**0 (28)**	7s
Lewitt	South-Africa	**0 (36)**	0 (144)	0 (58)	163s
GEPRO	Netherlands	0 (19751)	36 (54157)	**1 (566)**	2330s
KT2003	Netherlands	0 (33565)	27 (77148)	**0 (1410)**	1800s
KT2005	Netherlands	23 (13530)	98 (20588)	**0 (1078)**	1850s
StPaul	England	0 (81996)	62 (76782)	**0 (18444)**	1550s

All results were achieved using the instances of version XHSTT-2011.2. We improved two out of the nine existing solutions, but up to now we are not able to explain the rather poor results of the larger instances from the Netherlands and England. Because of the better results for smaller instances, and also for Lewitt which is comparable in size, we do not think that this is a more general problem of our approach, but rather a flaw of the grading procedure which we were unable to detect until now.

We did not include the Australian instances in our tests for two reasons: The definition of the limitWorkload constraint was problematic at the time of implementing, but is fixed already. This constraint is one of the key points of the Australian instances. Second, our algorithm is not well suited for dealing with the multitude of open roles that these instances incorporate. Closing the open roles after searching the maximum weight clique decreases control and influence on the search procedure. Having chosen a set of events, we have to fill their open roles, no matter how preferable assigning some of the resources is. The situation gets even worse when having hard avoidSplit-constraints: These define that all events of a course have to use the same resource to fill a certain open role. We already close the open role of the whole course when assigning the first event losing a lot of flexibility this way. However, this is an implementation-specific issue we became aware of too late, and not an inherent feature of our approach.

Artificial HDTT-Instances

The HDTT-Instances stem from the OR Library[4], and are very basic: Every event has one teacher, one school-class and one room assigned. There are 30 timeslots, and every resource has to be occupied in every timeslot to get all events assigned. There are only two hard-constraints, AssignTime and AvoidClashes, and no soft-constraints. We converted these instances to the XML-format of the Benchmarking Project.

Table 3 Results for the artificial HDTT instances

Method	*HDTT4*	*HDTT5*	*HDTT6*	*HDTT7*	*HDTT8*
SA1	-	0 (0.7)	0 (2.5)	2 (2.5)	2 (2.5)
SA2	**0 (0)**	0 (0.3)	0 (0.8)	0 (1.2)	0 (1.9)
TS	0 (0.2)	0 (2.2)	3 (5.6)	4 (10.9)	13 (17.2)
GS	5 (8.5)	11 (16.2)	19 (22.2)	26 (30.9)	29 (35.4)
NN-TT2	0 (0.1)	0 (0.5)	0 (0.8)	0 (1.1)	0 (1.4)
NN-TT3	0 (0.5)	0 (0.5)	0 (0.7)	0 (1.0)	0 (1.2)
CPMF	5 (10.7)	8 (13.2)	11 (18.7)	18 (25.6)	15 (28.6)
DWTAN	**0 (0)**	0 (0.4)	0 (1.65)	0 (2.1)	0 (3.25)
SA3	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0.4)**
EAH	**0 (0)**	2 (5.4)	6 (7.9)	9 (12.0)	13 (15.1)
TFH	**0 (0)**	0 (0.6)	0 (2.1)	0 (2.5)	0 (3.1)
TFH: time	107s	168s	220s	432s	697s

Table 3 compares our algorithms TFH and EAH with existing results: The first six methods are explained or cited in [19]. *SA* stands for Simulated Annealing, *TS* denotes Tabu Search, *GS* a Greedy Search and *NN-TT* are Hopfield Neural Net-

[4] http://people.brunel.ac.uk/ mastjjb/jeb/orlib/tableinfo.html

works. *DWTAN* and *CPMF* are neural network approaches too, described in [5]. The currently best method *SA3* again is a Simulated Annealing variant [11].

The first value of each cell indicates the hard-constraint violations of the best result achieved, followed by the *average* hard-constraint violations of 20 runs in parentheses. The row *TFH: time* denotes the average runtime of our TFH algorithm. As we can observe, TFH yields on these instances solutions that are competitive to those of the leading existing methods, while EAH's solutions are significantly worse.

Here, we did not apply the full hill-climbing procedure. Instead, we create and test random parameter sets, which – for TFH – was sufficient to create valid solutions. Applying hill-climbing results in better average results, but causes runtimes much higher than the ones stated in existing work.

4.1 Discussion

In general, the success of our algorithm highly depends on a suitable **grading procedure**. Dealing with several NP-complete subproblems, good grading functions are not easy to design and implement. Although we tried to maintain the relation between the grades and the constraints' penalties as directly as possible, the parameters for finding the best results are much more diverse than we originally expected. Despite the introduction of the soft-constraint level, the number of parameters often is still too high to lead to a clearly directed hill-climbing.

Regarding the **refilling-strategies**, it turned out that iteratively deleting and refilling timeslots and days helps to get all events assigned and therefore is more suitable for hard-to-fill instances. The other strategies – refilling resources, events and event-groups – help keeping the penalty low at the cost of having more difficulties assigning all events.

The results of the **maximum-weight clique search** are satisfying. The event-assigning heuristic EAH completely bypasses the clique search, and can therefore be used as an indicator for flaws in the clique-search. Surprisingly, especially for larger instances – where we assume that our clique-search yields worse results due to the NP-complete nature of the problem – TFH performs much better than EAH. This may partly be caused by the lack of backtracking, but it still indicates that the current bottleneck of TFH is not the clique-search, but rather the grading procedure.

Although theoretically applicable, our approach is not equally well suited for all instances. The Italian instance requires that at least one out of a set of multiple events is assigned to each of a certain group of timeslots. As TFH picks and fills single timeslots, we can either favor assigning all or none of those events. Although we can avoid assigning too many events using the resource limits of the clique-search, we lose fine-grained control at this point.

Interpreting the **final results** is not easy because of their diversity, and because of few existing solutions. Altough the timeslot-filling heuristic TFH often yields better results than EAH, it is not justified to declare it as the better approach in

general. The whole grading procedure was developed and implemented having TFH in mind. Also, the event-assigning heuristic EAH lacks backtracking. We assume that most of the existing solutions were achieved with an event-assigning approach, so the suitability of this approach for solving high-school timetabling problems is out of question. The performance of an approach strongly depends on the specifics of the given instance, as shown by the results of the artificial HDTT-instances. When developing our algorithm we did not adapt it to these instances in some more specific way. We were surprised by the good results of TFH, which are competitive to other, tailor-made algorithms. Completely filling each timeslot turned out to be much more appropriate than EAH to cope with this kind of problems. For TFH, a good solution of one timeslot occupies all resources (when not having any soft-constraints), which is exactly the bottleneck of the HDTT instances.

5 Conclusion

In this work we described a timeslot-filling heuristic (TFH) for creating high-school timetables. This heuristic is based on iteratively filling selected timeslots with sets of events. The more common approach, with which we compared our heuristic, is to iteratively assign single events. These approaches were evaluated using artificial instances as well as the real-world instances of the Benchmarking Project for (High) School Timetabling.

Although our algorithm is sometimes outperformed by tailor-made algorithms for particular instances, we demonstrated the general aptitude of the timeslot-filling heuristic. The suitability of an approach strongly depends on the characteristics of the instance it is applied to, which inhibits stating a clear winner. As the timeslot-filling approach is by far less explored, further investigation will be necessary to evaluate it in more detail. In particular, a comparison to other leading school timetabling algorithms that are flexible enough to handle the instances' constraints is required.

The main challenge definitely lies in the development of a suitable grading function. It has to maintain the balance between the various soft- and hard-constraints, between events of differing size, and the urge of creating a completely filled timetable. Future work may focus on more advanced concepts for the grading function. Self-adapting parameters or switching between various grading functions for one constraint are possible improvements. This could be done by applying local search algorithms to timetables created by the timeslot-filling heuristic. One could then focus on adapting the parameters or grading functions of the part (constraint-violation) the local search was able to improve. This would make the search for suitable parameter sets and inappropriate grading functions more efficient.

We want to thank the authors and contributors of the Benchmarking Project for their effort, and believe that their work will both help to organize and structure past and future scientific effort, and to revive the field of (real-world) high-school timetabling.

References

1. D. J. Abraham and J. H. Kingston. Generalizing bipartite edge colouring to solve real instances of the timetabling problem. In E. K. Burke and P. D. Causmaecker, editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Springer Lecture Notes in Computer Science*, pages 288–298, 2002.
2. E. K. Burke and M. Gendreau, editors. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, Montreal, Canada, 2008.
3. E. K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266–280, 2002.
4. E. K. Burke and H. Rudova, editors. *The Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*. Springer, 2007.
5. M. P. Carrasco and M. V. Pato. A comparison of discrete and continuous neural network approaches to solve the class/teacher timetabling problem. *European Journal of Operational Research*, 153(1):65–79, 2004. Timetabling and Rostering.
6. P. de Haan, R. Landman, G. Post, and H. Ruizenaar. A four-phase approach to a timetabling problem in secondary schools. In E. K. Burke and H. Rudová, editors, *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 423–425, Sept. 2006.
7. S. Even, A. Itai, and A. Shamir. On the complexity of timetabling and multicommodity flow problems. *SIAM Journal of Computation*, 5:691–703, 1976.
8. C. C. Gotlieb. The construction of class-teacher time-tables. In C. M. Popplewell, editor, *Proc. IFIP Congress 62*, volume 4 of *Information Processing*, pages 73–77. North-Holland Publishing Co., 1963.
9. J. H. Kingston. The kts high school timetabling system. In E. K. Burke and H. Rudova, editors, *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2006.
10. J. H. Kingston and T. B. Cooper. The complexity of timetable construction problems. In E. K. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 283–295. Springer, 1996.
11. Y. Liu, D. Zhang, and S. C. H. Leung. A simulated annealing algorithm with a new neighborhood structure for the timetabling problem. In *GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 381–386, New York, NY, USA, 2009. ACM.
12. B. McCollum, E. Burke, and G. White, editors. *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)*, Belfast, Northern Ireland, 2010.
13. C. Meyers and J. B. Orlin. Very large-scale neighbourhood search techniques in timetabling problems. In *Practice and Theory of Automated Timetabling VI (Sixth International Conference, PATAT2006, Brno, Czech Republic, August 2006, Selected Papers)*, volume 3867 of *Springer Lecture Notes in Computer Science*, pages 24–39, 2007.
14. N. Pillay. An overview of school timetabling research. In B. McCollum, E. Burke, and G. White, editors, *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)*, pages 321–335, 2010.
15. M. Pimmer. A timeslot-based heuristic approach to construct high-school timetables. Master's thesis, Vienna University of Technology, Nov. 2010.
16. 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*, pages 1–13, Feb. 2010.
17. A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.
18. G. Schmidt and T. Ströhlein. Timetable construction - an annotated bibliography. *The Computer Journal*, 23(4), 1979.

19. K. A. Smith, D. Abramson, and D. Duke. Hopfield neural networks for timetabling: formulations, methods, and comparative results. *Comput. Ind. Eng.*, 44:283–305, February 2003.
20. R. J. Willemen. *School timetable construction: Algorithms and complexity*. PhD thesis, Technische Universiteit Eindhoven, May 2002.