A Timetabling System for the German "Gymnasium"

Martin Löhnertz

Institut für Informatik V*, University of Bonn

Abstract. We present a timetabling program consisting of a batch solver and an interface for manual improvements. The batch solver is based on a tabu search metaheuristic which respects symmetry aspects of the solution space. We show how this can be combined with a graph coloring algorithm.
Model and interface are based on a set system structure. This allows a fast input of constraints and a customizable presentation. We show how to exploit idle phases of the interactive interface for better decision support.

1 Structure of our Model

1.1 The German "Gymnasium"

Timetabling for a german "Gymnasium" is a mixture of class-teacher timetabling and full course timetabling as it can be found at universities. Our model allows to represent a course timetabling problem but we will take some advantage of the simpler structure of the actual problem in Sec. 2.3.
The curriculum of a german "Gymnasium" consists of three phases. The classes 5 to 7 are taught in a class - teacher mode. In the classes 8-10 special courses are formed from groups of pupils from different classes. Classes 11-13 are taught in a course based system with arbitrary groups of students.
There are numerous soft constraints, e.g. teacher preferences or lessons requiring two subsequent timeslots.
We assume that the assignment of the teachers to the courses has already been done. We concentrate on the placement of the lessons in a limited number of timeslots and the allocation of suitable rooms and other resources.

1.2 Sets, Groups and Lessons

Our model is similar to that of Cooper and Kingston [2]. The founding principle of our system is that of a set. A set may include objects, timeslots or other sets. The sets are created from the base objects, i.e. teachers, classes rooms etc. This can be done manually or automatically based on attributes like room size or teaching abilities. A set may be contained in a different set as an element or as a

* this work was supported by Institut III

subset. The only restriction is that no set may contain itself cyclically. Typically there are some timeslot-only sets, e.g. the set "Monday" containing all timeslots which lie on a monday.
A problem specification is a list of "groups" each consisting of requests for some numbers of objects from specified sets. E.g. a group may consist of a request for teacher "A", a request for class "B" and a request for some chemistry room. Each group has a multiplicity describing how many timeslots have to be assigned to that group. The batch-solver assigns a timeslot to each (replication of a) group. Then is assigns objects to each request in that group by weighted matching. The results are called lessons.

1.3 Constraints

Using the set-concept constraints and preferences can be specified efficiently as they can be applied to all elements of a set simultaneously. There are constraints which apply to single objects only (whose handling is straightforward) and constraints which apply to the relations between objects.
Here we focus on relations like "teacher A likes Room 512". For each pair of sets (x, y) a value $w(x, y)$ can be specified which describes in seven grades whether objects of these sets or of subsets of these sets should be allocated together or not. For the other sets these values are derived from the relation-values of their respective supersets. Requesting that the set-subset relation is acyclic values can be propagated using the following formula:

$$w(x,y) := \frac{\left(\sum_{z \supset y} w(z,x)\right) + \left(\sum_{z \supset x} w(z,y)\right)}{\left(\sum_{z \supset y \wedge w(z,x) \neq 0} 1\right) + \left(\sum_{z \supset x \wedge w(z,y) \neq 0} 1\right)}$$

($\supset$ means direct relations and not the transitive hull). Values equaling zero represent "ignore" and are therefore not included in the averaging.

2 The Batch-Solver

2.1 General Structure

Our solver consists of two combined parts: A tabu-search metaheuristic and a weighted matching algorithm. The tabu search places (replications of) groups in the timeslots. Whenever it considers a new placement it has to evaluate a quality function to find out how good that placement is.

In order to do this the matching algorithm is invoked to assign objects to the resource requests in the groups. First a complete bipartite graph is created. The requests form one set of vertices and the objects form the other. Each edge is weighted by a value describing how good the respective object fulfills the corresponding request. Then a maximum weight matching is constructed and the requests are replaced with a set containing the respective object they have

been matched with. By this process groups are transformed to lessons, which contain only singleton sets.

The lessons then can be checked for clashes and fulfillment of constraints and preferences. This results in a value which is returned to the tabu search as the quality of the current placement.

2.2 Tabu Search

The solution space of a timetabling problem with only "hard" constraints (i.e. physical limitations) is symmetric in several aspects, especially:

1. There will be at least (#timeslots)! solutions which result from permutating the timeslots.
2. There often is a large number of identical lessons (multiplicity of groups).

Therefore an algorithm should operate on equivalence classes of these symmetries [4][5]. Unfortunately the soft constraints often have no symmetry at all.

We decided to use Tabu Search[6] as it utilizes three functions, the "ordinary" cost function, the "tabu" function and the "aspiration" function. The first and last represent the real cost of a timetable including soft constraints. We have put a strong emphasis on the hard constraints by giving each clash a weight of 1000 while preferences have an influence in the order of at most 100. These values are subject to further experiments. The base operation in our solver defining the "neighborhood" is the movement of a single lesson. This neighborhood turned out to be too large so we restrict our search to a subset of the neighborhood. Like [3] we evaluate moves from a single timeslice only. We select this timeslice based on an ordering by the sum of the number of clashes and the number of steps since that timeslice has been considered previously.
The "tabu"-function is used to make the search leave the current equivalence class. It is implemented as a "tabu list". When moving a lesson A created by group $G(A)$ we chose another lesson B from the "source" slot at random and add the unordered pair of their groups $(G(A), G(B))$ to the tabu list. We then forbid to place a lesson from $G(A)$ into any timeslot containing lessons from $G(B)$. On the one hand this prevents lesson A from being moved back to the old timeslice. On the other hand it prevents it from being inserted into timeslices which have the same structure as that timeslice. That the lesson B can still "follow" A to the new timeslot allows complete blocks to move. This furthermore implies that a timetable different to the current but in the same equivalence class can in principle still be reached, but this has become somewhat harder. If no more moves are possible we shrink the tabu list by dropping older entries.
The aspiration function overrides the veto of the tabu function if the solution considered is better than the best one found so far.

2.3 Hybridization

The unconstrained class-teacher timetabling problem can be solved by decomposing a bipartite graph into matchings (Koenig 1918 [7]). The problem is modeled by a bipartite multigraph. Teachers and classes are interpreted as the two

vertex sets, and each lesson is represented by an edge connecting its teacher with its class. Two lessons can be placed in the same timeslot if they do not share a teacher or class, i.e. when the corresponding edges are independent. Let l be the largest number of lessons a single teacher or class is involved in, i.e. the maximum degree of a vertex in that graph. One can show that there always exists a set of independent edges (matching) which involves all teachers and classes which participate in l lessons. If one places these in a timeslice and recursively considers the remaining problem the value of l decreases by one for this subproblem. Therefore l timeslices are sufficient (and necessary) to place all lessons. Using flow techniques[1] or Galvin's list-edge coloring algorithm[8], one can include some additional constraints.
We will use Koenig's method as a subroutine. Let $G = (A \dot{\cup} B, E)$ be a bipartite Graph. Let $c : E \to \mathbb{R}_+$ be an arbitrary weight function and $M := 1 + \sum_{e \in E} c(e)$. Denote by $\tilde{\Delta}$ the set of maximum degree vertices. We define a new cost function by

$$c'((v,w)) := M \ |\{v,w\} \cap \tilde{\Delta}| + c((v,w)) \tag{1}$$

for edges $(v,w) \in E$. A matching covering all vertices of maximum degree is "heavier" than any matching not doing so. For a pair of matchings covering the same number of maximum degree vertices the one which is better according to c is preferred. Using weighted matching algorithms a matching covering $\tilde{\Delta}$ and approximately fulfilling the requests coded in c can be found.
While most school timetabling problems are not simple class-teacher problems, most lessons still can be associated with at least one teacher and one "class". Therefore we can generalize this method by temporarily ignoring other objects and reconsidering them when analyzing the quality of a plan.
The methods described above can be used to construct a timetable modification subroutine which can be applied to any given timetable:

1. strip the lessons of all objects except for one teacher and one class
2. model each lesson by an edge in a bipartite multigraph using the classes and teachers as vertices
3. Decompose the graph into matchings by executing the following steps for each timeslice
   (a) set all edge weights c to 0
   (b) set $c(e)$ to 1 iff the lesson represented by edge e was placed in this timeslice in the original timetable
   (c) calculate weights c' according to formula (1)
   (d) find a maximum weight matching in the graph
   (e) place the lessons corresponding to matching edges in the current timeslot
   (f) remove the edges contained in the matching from the graph
4. reassign the objects stripped in step 1 by weighted matching

This routine has the interesting properties that:

1. Every timetable is mapped to a timetable which is feasible according to the chosen teachers and classes.

2. Feasible timetables are not modified.
3. For every timeslot a feasible matching is chosen which is as similar to the input as the request for feasibility and the slots already scheduled allow.

A straightforward application of this method would be to invoke it in the context of the quality function. Then the tabu search algorithm actually would search for the best input for this subroutine. Unfortunately this turned out to be too time consuming, as the quality function must also be evaluated for all the moves the tabu-search decides not to perform.
We decided to apply this method to the intermediate solution of the tabu search algorithm each 500 iterations. This is motivated by the idea that the tabu search moves out of the "attraction" range of one minimum into that of a different one. Then the coloring algorithm is used to jump fast towards that new minimum. Figure 1 shows the number of resource clashes over the number of iterations for a german school with 590 lessons which had to be placed in 32 timeslots (starting with a random placement). Note that each lesson can cause several clashes. The figure compares the intermediate solutions found by the plain algorithm with those found by the hybrid-algorithm. The peaks are caused by diversification steps. In the run displayed the best solution was actually found by the plain algorithm but on average the hybrid variant shows better values.

Fig. 1. Plain Algorithm vs. Hybrid Algorithm

3 Interactive Improvements

3.1 General Structure of the User Interface

The groups/lessons are displayd in user modifiable tables based on the set-subset relation. Here we differentiate between sets as elements and sets as subsets of other sets. When a set is chosen as a table index, the row (or column) headings are set to the names of the sets which are contained as elements. If there is a real subset it's name is not printed. Instead the scheme is applied recursively to it's elements and subsets. For example the sets "monday" and "tuesday" could be contained in the set "days" as an element. If "days" is chosen as a table index then there will be rows "monday", "tuesday" and so on. On the other hand a set "pupils" could contain all sets representing classes as subsets. Choosing "pupils" as a table index will list all names of the pupil-sets being elements of the classes-sets. The lessons are inserted into each field for which there is an nonempty intersection between an index set and a set contained in the lesson. Any two triples of sets can be chosen to form the indices of the table (three for the rows and three for the columns), allowing us to create almost any view on the data. The elements of the second and third sets are repeated for each element of the preceding sets.

Copy-operations are adjusted according to the context of each cell. Whenever a group (lesson) is cut all objects causing this group to be displayed at the given position are removed. This is best understood by looking at the following screenshot (Fig. 2): We have chosen a tuple for the column headings and a single set for the rows. The clipboard shows lessons striped from timeslots and some teachers, which will be added again by reinsertion into the table. In the top left corner there is a C++ lesson given by teachers F and AE to class I in room R14 on saturday (german: Samstag) morning. This lesson also contains the sets F and "Samstag.1". These are not displayed as they are implicitly given by the table structure. When cutting the lesson they will be removed. Upon reinsertion a minimal hitting set is calculated by a greedy heuristic for all sets defining the new position which do not include one of the inserted sets. For example when inserting in a table position with "Dienstag" (Tuesday) at the top and "2." to the left there is a set contained in "Dienstag" and "2." which is "Dienstag.2". This will be added to the lesson making it appear at the desired position. The same presentation and modification methods are available for the groups allowing a comparison of requests and created lessons with only a small context switch.

3.2 Decision Support

The time spent thinking by the user is completely idle time for the program and most probably for the whole system, because mostly single-user PCs are used for practical timetabling. We try to exploit this time to calculate some suggestions for the proceeding of the manual process.

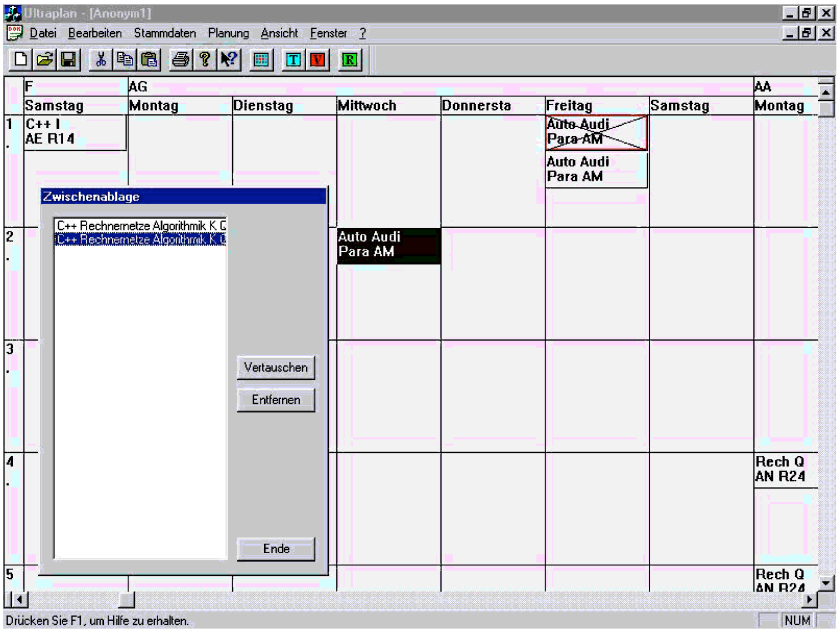

Fig. 2. The Clipboard & Inserting; Zwischenablage=Clipboard,vertauschen=exchange

It didn't seem reasonable to calculate a (necessarily) bounded sequence of steps which does not lead to a clash. Furthermore the resulting moves often cannot be understood by the user. We therefore chose to warn the user about moves which provably do not lead to a solution. We do this by an exhaustive search whose search depth increases the more the longer the user does nothing. We assume that the user wants to insert the elements of the clipboard into the plan one by one. The clipboard is organized as stack. When clicking on a empty cell the topmost (or the selected) clipboard entry is popped and pasted.
In iteration one we mark all fields "forbidden" in which the topmost entry of the clipboard cannot be placed without a clash. If the user is still waiting we also mark the fields in which the topmost entry of the clipboard cannot be placed without making a clash free inserting of the second clipboard entry impossible. Then we use a lookahead of $3, 4$ and so on. In the time we tended to think about our next move the program usually reached level three or four. So we are warned all insertions which will lead to a no chance situation within the next four placements.

Figure 3 gives an example: It consists of 3 objects (o1,o2,o3) and two timeslices (t1,t2). Four lessons consisting of one or two objects ((o1),(o2),(o3),(01,02)) have to be placed into the two timeslots. The insertion of the lesson containing object $o1$ in timeslot $t1$ would make the placing of $o1, o2$ impossible and is therefore forbidden (diagonal line).

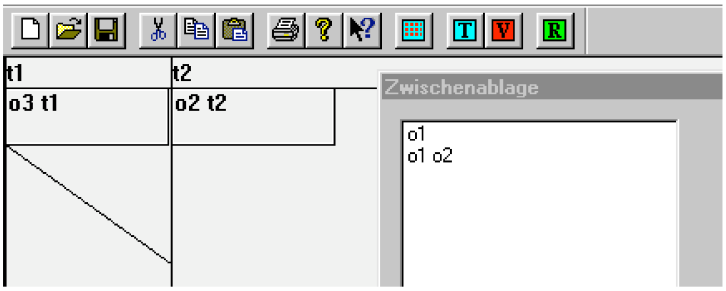

Fig. 3. Warning the user ("Zwischenablage"=Clipboard)

References

[1] N. Chahal and D. de Werra. An interactive system for constructing timetables on a PC. *European Journal of Operational Research*, 40:32–37, 1989.

[2] Tim B. Cooper and Jeffrey H. Kingston. A program for constructing high school timetables. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 132–143, 1995.

[3] D. Costa. A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research*, 76:98–110, 1994.

[4] A.E. Eiben, J.K. Van der Hauw, and J.I. Van Hemert. Graph coloring with adaptive evolutionary algorithms. *J. Heuristics*, 4:25–46, 1998.

[5] W. Erben. A grouping genetic algorithm for graph colouring and exam timetabling. In *Proceedings of the third conference on practice and theory of automated timetabling (PATAT 2000)*, page 397, 2000.

[6] A. Hertz. Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics*, 35:225–270, 1992.

[7] Denes König. *Theorie der Graphen.* Chelsea, 1918.

[8] Martin Löhnertz. Theorie und Praxis der automatischen Stundenplanerstellung. Diplomarbeit, Institut für Informatik Bonn, 1999. http://www.loehnertz.de/Diplom.html.