Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Term Project at IFOR
(Institute for Operations Research)

Timetabling Problem for a High School

Roland Erwin Kurmann

Summer 2000

Supervisors:
Dr. Eleni Pratsini
Dr. Maurice Cochand
Prof. Hans-Jakob Lüthi

This report was written with LyX, LaTeX 2$_\varepsilon$, KOMA-Script and xfig.

Abstract

This term project investigates the timetabling problem of a high school. The timetabling problem consists of assigning courses, which are taught by teachers and taken by classes to time periods and class rooms. The objective in a timetabling problem is to eliminate the number of conflicts due to lessons taking place simultaneously, involving common classes or teachers and to produce a compact schedule. Real data from "Kantonsschule Sargans" was used and the project considered real-world restrictions like availability of teachers and classes, blocking of time periods, splitting of courses into consecutive lessons, respecting double lessons and lunch breaks.

Tabu search techniques were used for optimizing. The tabu search program was implemented in C++ and improved for speed. Numerical experiments were made with the data from Sargans. These tests show that conflicts can be resolved in 12 seconds and a more compact schedule can be generated in 1 to 2 minutes.

Contents

1. Introduction

Every year a timetable has to be constructed for the teachers and pupils at a high school. The timetabling problem consists of assigning courses, which are taught by teachers and taken by classes to time periods and class rooms. Obtaining a feasible schedule is a hard problem as there can be a large number of teachers, classes, courses to be taught, classrooms with specific features and restricted teacher availability. In many high schools the yearly timetable is still done by hand.

Tabu search is a metaheuristic first described by F. Glover, and is one of the most efficient heuristics for handling large optimization problems. A. Hertz [1] considered timetabling and the grouping subproblem in his work. He adapted the tabu search techniques for timetabling and combined it with a global approach in order to handle the subproblems. These subproblems were addressed in one global approach and therefore treated in the same iteration process. He applied his algorithm for the exam plan of ETH and the course schedule for the Faculty of Economics of the University of Geneva. In this term project mainly the ideas of A. Hertz are used.

The main task of this project was to adapt Hertz's ideas to a real world problem. As I had no previous experience in applying tabu search, this project also implied a good understanding of the method and experimentation with the various parameters of the technique.

The data and requirements of the "Kantonsschule Sargans" were used. The requirements are described in section 2.2, the data is described in section 5.1.

2. Timetabling Problem

2.1. Description

The classical timetabling problem consists of assigning time periods to teachers and classes. The objective is to eliminate conflicts among lessons taking place simultaneously involving common students or teachers. In reality the problem becomes more difficult since additional constraints have to be taken into account: the availability of teachers, mixing half classes, the length of lessons may not be uniform, sufficient time should be provided for students to move from one building to another if necessary, compactness or splitting of courses into consecutive lessons. Moreover, some courses may be optional courses and we want to respect the students' freedom of choice.

There may also be other problems associated with the timetabling problem, for example the grouping problem, or the room problem. Courses taken by a large number of students have to be repeated several times during the week. Hence the grouping of students into course sections needs to be carried out so as to create a timetable with as few conflicts as possible. In the room problem there must be rooms associated to lessons. Some lessons have special requirements for rooms, e.g. chemistry can only be taught in a chemistry lab. These problems may be strongly related and have to be considered in one global approach.

The requirements vary from school to school. Restrictions can be divided into *hard* and *soft* ones. Hard restrictions must be met while soft restrictions should be met if possible.

In large scale applications, the timetabling problem is NP-hard [9]. Therefore it cannot be solved with standard optimization procedures and must be addressed with heuristics.

A number of approaches and papers on the timetabling problem exist. A. Hertz [1, 2] uses the metaheuristic tabu search, while R.C. Carlson and G.L. Nemhauser [8] formulated the problem as an assignment problem. The global approach idea was proposed by J. Aubin and A. Ferland [6]. Suggestions on the division of restrictions can be found in the paper of H.A. Eiselt and G. Laporte [5].

2.2. Real-World Problem from Sargans

The "Kantonsschule Sargans" is a Swiss high school with 39 classes, 120 teachers and more than 840 pupils. Several courses are given in Sargans, from economics, math,

new languages and latin to preschool courses for nurses (DMS), teacher education (Lehrerseminar) and intermediate economics school (WMS).

The problem is to assign a time period and a class room to lessons taught by a given teacher to given classes. The school preassigns which teacher teaches a certain subject to certain classes. A course consists of one ore more lessons. The assignment of time periods and rooms is restricted in several ways. The restrictions were found through discussions with the school administrator.

1. Teachers are only available at certain days and times.
2. Each teacher is involved in at most one lesson in a time period.
3. Each class is involved in at most one required lesson in a time period. A class is together for required courses but can be divided for optional courses.
4. Courses have to be split into single or double lessons. If a course is split, the two parts cannot be taught the same day. For example, four hours of mathematics should be taught twice a week in two consecutive time periods.
5. It must be possible to block lessons. Blocking is needed for special courses which have additional constraints and have to be scheduled "by hand". A time period is provided in advance for blocked lessons.
6. A lunch break must be respected. A lunch break should have the length of two time periods.
7. Certain courses require special classrooms, e.g. chemistry.
8. A lesson can be taken by pupils from more than one class. For example, introductory biology can be taught to two or three half classes. Additional lessons on this course can be given to the various classes separately.
9. The schedule should be compact, have only a few gaps and no long afternoons.
10. All lessons have the same time length.

All the data for this term project is taken from the school year 99/00. There are a total of 1650 preassignments. Of these 486 are blocked and the other 1164 need to be assigned by the optimization program. An extraction of the preassignments for the year 99/00 can be found in the appendix B. The preassignments were originally stored in a Microsoft Excel 5.0 for Macintosh file. There are 13 time periods per day for Monday through Friday, and 5 periods on Saturday, giving a total of 70 time periods per week. The time periods are specified by a unique 3 digit number. The first digit takes values of 1 to 6 indicating the day in the week, while the last tow digits indicate the time period in the day having values 01 to 13. For example 208 is the 8th time period on Tuesday. The 6th time period is the common lunch break. Information about class rooms doesn't exist in a useful form and has to be gathered. Currently, the administrator only uses his experience for assigning class rooms.

Teachers may have part time jobs, so some of them are very restricted in time. When generating a new schedule, schedules from previous years can be consulted, however, different teacher availability and changing curriculums frequently require a significant change of the schedule from year to year.

The timetable is currently created “by hand” on a wall using a gantt chart. The size of the wall is approximately 1.5m x 3m. Four hundred and eighty hours (5 Jahreswochen-lektionen) are invested every year in creating the timetable. The people in charge of the schedule pin up small pieces of paper representing teachers, classes and rooms on the wall. They have to remember all the details for the whole timetable. This job needs a lot of experience and they use “intuitive” heuristics for assigning the objects, e.g. they start with hard cases.

3. Tabu Search

3.1. Introduction

[1]Tabu search is a metaheuristic designed for getting a global optimum to a combinatorial optimization problem. It is basically an iterative method used for finding in a set X of feasible solutions a solution s which minimizes an objective function f. A neighborhood $N(s)$ is defined for each solution s in X. The procedure starts from an initial feasible solution s_{init}; whenever a feasible solution s has been reached, a collection V^* of solutions s_i in $N(s)$ is generated, and a move made to the best s_i in V^*. This best neighbor is denoted s^*.

In oder to prevent cycling to some extent, it is not permitted to turn back to solutions which were visited in the previous k steps, where k is a given number. More precisely, we introduce a so-called tabu list T of length $|T| = k$ (fixed or variable) which is used as a queue: whenever a move from a solution s to s^* is made, we introduce s at the end of T and remove the oldest solution from T. So all moves back to s are now forbidden for the next $|T|$ iterations; s is a *tabu solution* and any move going back to s is a *tabu move*.

Deciding that some moves are tabu moves may be too absolute. Therefore we introduce an additional feature to cancel the tabu status of a move when this move gives a considerable improvement. An aspiration function $A(z)$ is defined for every value z of the objective function. If a move to a neighbor solution s_i is a tabu move but gives $f(s_i) \leq A\,(z = f(s))$ then the tabu status of this move is dropped and s_i is considered a normal member of the sample V^*.

Stopping rules have to be defined; in many cases a lower bound f^* for the minimum value of f is known. The iterative process may then be interrupted when we are close enough to f^*. However in general f^* is not available with sufficient accuracy and we give a maximum number $nbmax$ of iterations: as soon as $nbmax$ iterations have been performed without improving the best solution obtained so far, the whole procedure is terminated.

If there are subproblems around, then these subproblems can be solved in a global approach as described by A. Hertz [1] or J. Aubin and A. Ferland [6]. First initial solutions for all subproblems are generate; then an iterative procedure is used which adjusts the timetabling subproblem and the other subproblems successively until no more improvement of the objective function can be obtained. At each iteration two procedures are used:

[1]The introduction is mainly taken from A. Hertz [1].

Algorithm 1 The general tabu search technique.

s :=initial solution in X;
$nbiter := 0$; *//current iteration*
$bestiter := 0$; *//iteration, when the best solution has been found*
$bestsol := s$; *//best solution*
$T := \emptyset$;
initialize the aspiration function A;
while $(f(s) > f^*)$ **and** $(nbiter - bestiter < nbmax)$ **do**
    $nbiter := nbiter + 1$;
    generate a set V^* of solutions s_i in $N(s)$
        which are either not tabu or such that $A(f(s)) > f(s_i)$;
    choose a solution s^* minimizing f over V^*;
    update the aspiration function A and the tabu list T;
    if $f(s^*) < f(bestsol)$ **then**
        $bestsol := s^*$;
        $bestiter := nbiter$;
    endif
$s := s^*$;
endwhile

(a) given a solution of the other subproblem generated during the preceding iteration, the timetable is modified to reduce the number of conflicts,

(b) with this new timetable, the other subproblem is modified to reduce the number of conflicts.

These subproblems stop when a local minimum is reached. Tabu search techniques have been especially designed for avoiding being trapped in local minima and are generally much more powerful than other methods.

There are several documents describing the tabu search techniques. An edition of *Annals of Operations Research* [3] is dedicated to tabu search. A good introduction ("user's guide") is given in this edition by F. Glover, E. Taillard and D. de Werra [4]. F. Glover, the inventor of tabu search wrote several papers on the method. In "Tabu Search - Part I" [10], the fundamentals were discussed, whereas in "Tabu Search - Part II" further information, like parallel processing implementations, are mentioned. The paper [12] of A. Hertz and D. de Werra gives a survey of different applications of tabu search. The article of F. Glover "Tabu Search - Wellsprings and Challenges" [13] gives the history of tabu search as well as explanations of the name.

3.2. Elements of Tabu Search

The following structures are required by tabu search:

- X: the set of feasible solutions;
- f: the objective function;

Algorithm 2 The main algorithm (global approach).

$u^* :=$ initial timetable;
$v^* :=$ initial for the other subproblem;
$fbest := f(u^*, v^*)$;
$continue := true$;
while $continue$ **do**
    $f' := fbest$;
    (a) use timetabling subalgorithm with a fixed $v = v^*$ for getting
        an optimal timetable u';
    if $f(u', v^*) < fbest$ **then**
        $fbest := f(u', v^*)$;
        $u^* := u'$;
    endif
    (b) other subalgorithm with a fixed $u = u^*$ for getting
        an optimal solution for the subproblem v';
    if $f(u^*, v') < fbest$ **then**
        $fbest := f(u^*, v')$;
        $v^* := v'$;
    endif
    if $fbest = f'$ **then**
        $continue := false$;
    endif
endwhile

- $N(s)$, $s \in X$: the neighborhood of a solution s in X;
- f^*: if available, a lower bound for the objective function f.

The following structures are used by tabu search:

- V^*: a collection of solutions s_i in $N(s)$, where s is the current solution;
- T: tabu list with forbidden moves;
- $\mid T \mid$: the size of the tabu list;
- A: aspiration function, which allows tabu moves if and only if some improvement is achieved;
- $nbmax$: the maximum number of iterations without improvement.

With tabu search it's possible to approach a problem step by step. So it's possible to adapt a rudimentary prototype implementation to encompass additional model elements, such as new types of constraints and objective functions. Similarly, the method itself can be evolved to varying levels of sophistication.

This property allows the programmer to start with a basic version of the optimization algorithm. This gives early feedback and the program can be enhanced in an "interactive" way, restriction by restriction.

4. Model

The problem environment with its components and their relationships is shown in the diagram 4.1.

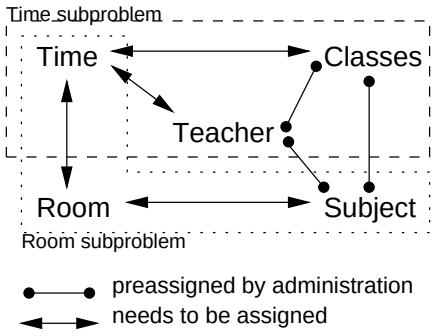

Figure 4.1.: Time tabling schema with the time and room subproblems.

There are various objects in a time table which have to be treated: classes, rooms, teachers, times, subjects. These objects have different relationships. In our case the school administration preassigns which teacher teaches a certain subject to certain classes. Such a preassignment is fixed and denoted as *stc* (subject-teacher-classes).

The problem can be split up in two subproblems: the time assignment problem and the room assignment problem. In the time subproblem each *stc* is assigned to a time period, while in the room subproblem each *stc* is assigned to a room. The problem can be summarized by the following table.

preassigned by school		need assignment
subject-teacher-classes (stc)	$\rightarrow$	time slot
	$\rightarrow$	class room

The time problem can be modeled as an *assignment problem*. Consider a set S containing the preassignments of subject-teachers-classes (*stc's*). Also consider the set of all available times, T. Each element in S needs to be assigned to exactly one element in T. A *stc* is the basic object, which has to be scheduled by the program. An assignment can be described by a partition ($T_1, \ldots, T_k$) of the S into a fixed number k of subsets, where k is the number of available times.

Restrictions can be achieved either by penalizing it in the objective function, these are denoted as *soft constraints* or by the feasibility of a solution, these are denoted as

hard constraints. If too many terms are in the objective function f, then the process is too slow. Also, finding an initial solution can be difficult if feasibility is hard to achieve. There are various ways of handling restrictions [1, 5, 7]. The approach of the state-space graph of A. Hertz [1] is used in this term project. The solutions of the optimization problem can be viewed as nodes in a state-space graph G. The idea is to find a well connected state-space graph in order to enable the algorithm "to easily move through" the different solutions. In a low or even not connected state-space graph the algorithm is captured in a single graph component and only finds local optima. It's possible to define a connected state-space graph containing nodes which are not necessarily feasible timetables. An initial solution can thus be generated in an easier way and paths exists from any initial solution to the best feasible timetable.

The *restrictions* from Sargans given in section 2.2 are modeled in the following way: The restrictions 1 (availability), 4 (course splitting), 5 (blocking), 6 (one lesson lunch break, 6th time period of a day) and 8 (half classes) are considered as *hard constraints* as they are easy to satisfy. Therefore, an initial feasible solution can easily be obtained and a lunch break is guaranteed. Courses will be split in the initial solution into consecutive lessons and during optimization only considered as consecutive lessons. All classes which take the same lesson (restriction 8) are in the same *stc*, because a *stc* consists of the preassignment subject, teacher and classes. The restrictions 2 (teacher conflict), 3 (class conflict), 6 (second lessons lunch break, 7th time period of a day) and 9 (compactness) are considered as *soft constraints*. Restriction 2 (teacher conflict) and 3 (class conflict) are difficult to satisfy and are denoted as conflicts in the objective function. This model gives a state-space graph G which is well connected. Compactness can only be achieved through penalizing. The second lunch break is a desire and is therefore treated as soft constraint. Restriction 7 (special class rooms) belongs to the room problem which isn't considered in this term project. Restriction 10 (same time length) allows abstraction of real times. Finally, in the following text the term solution always denotes a feasible solution.

A *neighborhood* is defined by single moves. A move is obtained when new consecutive time periods are assigned to the *stc's* of consecutive lessons. A move corresponds to a move of a *stc* in a gantt chart from one to another position. A solution s_a is a neighbor of another solution s_b if s_a can be reached through one consecutive lesson move from the solution s_b. If s_a is a neighbor of s_b, then s_b is also a neighbor of s_a.

The *objective function* f measures the conflicts and the compactness of a solution s, which contains an assignment for every element of S to one element of T. Compactness is a property of a class schedule and is therefore summed over all classes for a solution s. The goal of the compactness is to reduce gaps in a schedule of a class and to end as early as possible in the afternoon. Nevertheless, up to three gaps beside the daily lunch breaks in a week are desired for a class. C is the set of classes.

$$f(s = (T_1, \ldots, T_k)) = \frac{1}{2} \sum_{\substack{x,y \in S \\ x \neq y}} d(x, y) + \sum_{c \in C} e(c)$$

$$d(x,y) = \begin{cases} p_0 & \text{if x and y are in the same } T_i, \\ 0 & \text{else.} \end{cases}$$

$$e(c) = p_1 \max\left(|FP(c)| - 3, 0\right) + p_1 |PP(c)|$$

$FP(c)$: set of current free periods of a class c of a week, which are not free morning or afternoon periods and 7th periods

$PP(c)$: set of 7th and 13th periods of a class c of a week, which are not free periods and not blocked

Originally, the timetabling problem was modeled as a *constraint semi-assignment problem* (C-SAP) [15, 16], as a frame work for this problem class already exists. However some restrictions like teacher availability couldn't be modeled as C-SAP constraints and therefore this approach was dropped.

5. Implementation

5.1. Input Data

The original data from the “Kantonsschule Sargans” was available in a Mircrosoft Excel 5.0 for Macintosh file. This so-called “Arbeitsbank” contains all the preassignments. For the tabu search optimizing program the file was converted to a comma separated text file (csv). This new format can be used easier than the Excel format. The conversion can be done directly in Excel. The classes and the teachers were extracted out of the “Arbeitsbank”. The teacher availability was given only on paper and the data had to be entered. The availability can be given in a teacher input file. A list of unavailable time periods can be given for each teacher. The availability of classes can be listed in a file as well. Lessons, which aren’t intended to be assigned by the program must be blocked. If a lesson is blocked, then the provided time period of this lesson must be blocked by the involved teacher and the involved classes. Thus, the teacher and the classes are not available at this time period, and the blocked *stc* has to be deleted from the “Arbeitsbank”. The data and the data format are given in the appendix B.

5.2. Tools

As a programming language for the implementation *C++* is used. C++ generates efficient code, allows efficient programming and structuring is supported through object orientation. The C++ *Standard Template Library* (STL) is used for some data structures. The *GNU Compiler Collection 2.95.2* (g++) and *Borland C++ Compiler 5.5* (bcc32) were used as C++ compilers. Both are optimizing, high quality compilers and are freely available.

The program is developed to be platform independent and can be run on UNIX (e.g. Sun Solaris), Linux and Windows. It wasn’t possible to compile the program at ETH for Apple G3 because the STL of C++ there was not up to date.

Java was used for graphically viewing the timetable. Java is better than C++ for graphical representation as there is a good program library and speed is not important. Furthermore, Java is platform independent.

5.3. Data Structures

The goal for the data structures was efficiency, ease of handling during programming and safety.

5.3.1. Integer Representation

In this implementation all input information is converted into integers. Integers are the basic data structure of the processors, so they are fast and easily handled in programs.

- Numbers are directly converted into integers.
- Text information is also converted into integers. Four characters can be placed in one integer on a 32 bit processor, so the maximum length for a text is four characters. In the input format given in appendix B only subject, teacher and classes names having a length four characters or less are allowed. This representation needs no additional encoding for text information and increases the safety of the program, because at every program step text information can be printed on the display. The integer value may differ on different computer platforms due to the internal encoding of integers, denoted as little endian encoding versus big endian encoding. This difference in representation is not a problem for the algorithm.

5.3.2. Containers

Containers are the data structures for storing sets of data. For efficiency an array based container with random access is needed. Because normal C++ arrays do not offer any bound checking, they aren't suitable during development. In the STL of C++ *vector* is an alternative. Vector itself doesn't offer bounds checking either, but vector can be inherited and supplemented with bound checking. The enhanced *CheckedVector* and *CheckedMatrix* of U. Breymann [18] are used. The use of these data structures is very simple due to operator overloading in C++. *CheckedVector* and *CheckedMatrix* can be used with array notation. *Maps* and *sets* are also used from the STL. STL *set* supports the basic set operations. STL *map* is a set of mappings name to value.

5.3.3. Model Representation

- The *stc's* are stored in a *CheckedMatrix stcs.* A single *stc* is stored in a row of the *CheckedMatrix stcs* and is accessible through the row number. The availability of teachers and classes is combined to the availability of a *stc*. A solution s is stored in *CheckedVector*. This *CheckedVector* contains the assigned time periods of the corresponding *stcs* rows.
- The tabu list is a *CheckedMatrix* with columns for time periods and *stc* row numbers.

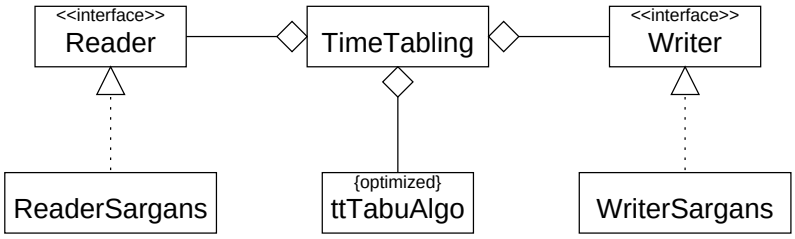

Figure 5.1.: UML class diagram for the timetabling program.

- The information for the aspiration function is stored in a *CheckedVector*. This *CheckedVector* contains an index for every objective function value. The size of the *CheckedVector* depends on the initial value of the objective function.

5.4. Program Design

The timetabling program for generating a feasible timetable is drafted as a command line program. The algorithm can be controlled through program parameters. The input data is provided in text files.

Flexibility in the sense of changing the input data format, e.g. data from another school, should be achieved through program design.

The program is designed as an object oriented program. The UML diagram is shown in figure 5.1. An interface, in the Java sense, is provided for input reading with the name *Reader*. The same is for writing files with interface *Writer*. The concrete data reading and writing classes implement these interfaces. Readers and writers use the *Strategy* design pattern described by Gamma et al. [20]. The class *TimeTabling* is the main class which processes the command line parameters, starts the reading, the optimizing process and the writing.

The class *ttTabuAlgo* contains the tabu search algorithm. The class is optimized for speed. This class has all ingredients for tabu search: the aspiration function, objective function, neighborhood calculation, test functions and other auxiliary functions.

5.5. Tabu Search Algorithm

5.5.1. Initial Solution

An initial solution s_{init} depends on the definition of *feasibility*, because an initial solution must be feasible. In this case the feasibility isn't difficult to achieve. In the first implementation an initial solution was built in a simple deterministic way for better and easier checking of program changes. In a second step the initial solution was built at random which produces better initial solutions according to the objective function f.

Another task of the initial solution is *splitting courses* into consecutive lessons. Consecutive lessons are always moved together.

5.5.2. Neighborhood

The neighborhood V^* is an important part of a tabu search implementation, first for the quality of a solution and second for the program speed. In this implementation V^* contains only the moves to the neighbor solutions, not the neighbor solutions themselves. The generation of neighbors is a directed process. The *stc's* with the highest number of conflicts are considered first. For every *stc* the potential improvement of the objective function is calculated and stored in a special data structure. This information is then transformed in a list and sorted after the potential improvement of a *stc*. In this list there are different *levels* of improvement, which all have members of the same improvement. The members of the first five levels are randomly permutated with elements of the same level. This approach avoids cases where the same *stc's* are reconsidered again and again. Then beginning with the first item on the list, m random feasible time periods are created for each *stc*. A feasible time period is a time period which is available for all consecutive lessons. These potential moves are inserted into V^*, if they aren't *tabu* or can *aspire*.

Since the whole neighborhood of one solution is too big, the neighborhood has to be decreased in a clever way. In decreasing the neighborhood, the real improvement and the potential improvement of a move are compared. If the potential improvement is the same as the real improvement then the look for further neighbors is stopped. In a next step, the first best neighbor is taken out of V^*. Since the *stc's* are sorted in decreasing order of number of conflicts, and randomly permutated within each level, selecting the first neighbor is a good strategy. While conflicts exist, *stc's* with only a bad compactness value aren't considered. An upper limit for the number of neighbors is introduced in order to end the search at a certain point. In literature [1] $|V^*| = \frac{1}{2}n$ is a good value where n is the problem size, which is in our case the number of *stc's*. It is possible that no neighbor is available for a *stc*. If this occurs the *stc* is left out and the next *stc* on the list is taken.

5.5.3. Tabu List

The tabu list stores the last moves in order to prevent cycling. Whenever a *stc* x is moved from T_i to T_j, the pair (x, i) becomes tabu and then x cannot return to T_i for $|T|$ iterations. This pair is stored in a *CheckedMatrix* as a cyclic list. The tabu list has to be updated after every move. The tabu list length is important in the tabu search metaheuristic for speed and quality of a solution. In the literature [4], it is suggested to take either $|T| = 7$ or vary randomly the length of $|T|$ in the range of 5 to 10 every 20 iterations. Through the randomization of the tabu list length the search of neighbors may lead to other parts of the neighborhood. The lower and upper bound of $u_1 \leq |T| \leq u_2$ can be given as parameters in this implementation.

5.5.4. Aspiration

This function cancels the tabu status of a move, if it generates a better solution. The best objective function improvement from the current objective function value is stored in an aspiration function $A(z)$, defined for every value z of the objective function. If a move to a neighbor solution s_i is a tabu move, but gives $f(s_i) \leq A(z = f(s))$ then the tabu status of this move is dropped and s_i is considered as a normal neighbor. The aspiration function is initialized with $A(z) := +\infty$ for every value z of the aspiration function.

The aspiration function has to be updated whenever we move from a solution s to the best solution s^* in V^*:

- $A(f(s)) := \min\{A(f(s)), f(s^*) - 1\}$;
- $A(f(s^*)) := \min\{A(f(s^*)), f(s) - 1\}$.

5.5.5. Objective Function

The objective function f measures the number of conflicts and the compactness of a timetable. The objective function is the most time consuming[1] part of the program. Therefore a lot of development time was spent to improve the objective function calculation speed. Two principals were used: calculate only changed parts and maintain special data structures.

The objective function f given in the model in section 4 gives the mathematical idea. The calculation in the program is improved for speed. The objective function has two parts: penalizing conflicts and measuring compactness. First the *conflict* part is considered. The full objective function is only calculated at the beginning as start value and at the end of the program as security comparing value. Otherwise only the difference is calculated to the current objective function value. The difference calculation of the objective function with consecutive lessons gives several conditional cases due to overlapping lessons. In the first implementation the objective function looked for a time conflict with *every* other *stc*. Then a list for every *stc* was built in advance with *stc's* involving common teachers or classes. With this special data structure the objective function must only check time conflicts within the listed *stc's*. This program change improved the speed by a factor of ten. One problem was to calculate the objective function for neighbors. The calculation of the objective function for neighbors is important because the best neighbor has to be chosen. Because the evaluation of the objective function of neighbors must be done before a move, you cannot move, change the solution and use the normal objective function. So a move must be simulated considering all the conditional cases for the calculation of the objective function for a neighbor.

The *compactness* part is now considered. A special data structure for the free periods of a class will be built and maintained. This data structure is updated after every move.

[1] This fact can be seen in the profiling information of the timetabling program which can be obtained with programming tools like *gprof*.

For updating this data structure another one is used. The second data structure contains the current *stc's* which are conflicting in time. The updating of these data structures is an important part of the program because other iterations depend on these. Since the compactness is a property of a class, a data structure with the free periods of a class is a good choice. The measuring of the compactness is implemented as "if . . . then . . . " rules.

As a matter of fact it can be said that a trade off between development time and program speed exists.

5.5.6. Stop Rule

As a stop rule there are two possibilities: a lower bound f^*, or a threshold for the number of iterations without improvement. Since there is no lower bound f^*, the algorithm is stopped by the number of iteration without any improvement, and $nbmax$ is the threshold. In mathematical terms, the algorithm stops if $nbiter - bestiter > nbmax$, where $nbiter$ is the current number of iterations and $bestiter$ the iteration with the best solution so far.

5.5.7. Parameters

Several parameters can be set in the algorithm. These parameters can be easily changed by the user, see details in the appendix A.

- $|V^*|$, upper neighborhood limit
- m, number of randomly generated alternative time periods for one *stc*
- $nbmax$, the number of iterations without improvement
- $u_1 \leq |T| \leq u_2$, lower and upper limits of the tabu list length
- p_0 and p_1, the penalty values for conflicts and uncompactness

5.5.8. Program Output

In order to use a generated solution, the solution must be stored as a file. The same format is used for both input and output. The data format is given in the appendix B. The program collects several informations during a run like development of the objective function, the number of tabus, the number of aspirations, time to resolve conflicts, time to end solution, compactness of first conflictless solution, etc. This information is stored in a report file and can be viewed later. A solution file and a report file contain the start date and time in the name. This is good for two reasons, first it allows several starts without user interaction and facilitates the identification of a solution, at a later time.

5.5.9. Computational Efficiency

For this combinatorial optimization problem the speed had to be improved. Two approaches were considered, the use of clever data structures and the improvement of the compiling quality, e.g. function inlining, reordering expressions. The compiling quality can be affected by optimization levels of the compiler and function inlining. It can be said that the use of special data structures gives bigger improvements in speed than compiling optimizations.

5.5.10. Testing

Once an algorithm is implemented it has to be tested. Considerable time was invested for program verification. The two classic methods were used for testing: *white box testing* and *black box testing*.

White box test means testing with knowledge of the implementation of a function. For almost every function of tabu search a white box test function was written. This test function calls the ordinary function with normal and extreme values and tests the different cases. Only thinking about writing test functions helped detecting errors, because all the extreme case were considered. Due to the use of array based data structures a lot of indices occurred and all these indices have a lower and an upper bound. If the program had to be enhanced, then the previous written test function could be enhanced too.

Black box tests work only on input and output and consider everything between as a black box. The viewer program described below was used for black box testing. The graphical representation of a solution could be compared with the expected solution.

5.6. Viewer Program

For viewing a timetable the solution file must be represented as a graph. A viewer program was developed which shows a solution as a *gantt chart*. This gantt chart shows the timetables for classes and teachers. The time is represented on the horizontal axis of the gantt chart, the classes and teachers are listed on the vertical axis. Conflicts with teacher and classes are drawn in red color. Lessons taking place at blocked time periods are drawn in dark gray color. Free periods are white rectangles. Blocked periods are light gray rectangles. The different gantt charts can be compared by the school administrator. This viewer program can also be used to check the *correctness* of a solution as well as for analysis. The viewer program allows to store timetables as images.

6. Results

6.1. Parameter Tests

Tests and experiments were made with the tabu search program. The parameters considered were the upper bound for neighborhood $|V^*|$, the neighbors to generate per stc m, the stop criteria $nbmax$, the lower u_1 and the upper u_2 bound of the tabu list length. The goal of the experiments was to see the influence of these parameters and to find optimal values of the parameters. Another question of the testing was whether it is better to make a lot of fast runs or to make only a few slow, more precise runs.

The tests were made on a Sun Ultra 10 Workstation with a 333 MHz UltraSPARC-IIi CPU and 256 MBytes RAM from Sun Microsystems. The program was compiled with gcc 2.95.2 with optimization level -O3. A shell script was used to start the program several times. The program was started on several computers in parallel.

Forty runs were made for each parameter experiment. Two different tests were conducted. In the first test, shown in table 6.1, the tabu list length was fixed and the numbers of neighbors and the stop criteria were investigated. In the second test shown in table 6.2 the tabu list length was investigated and the other parameters were fixed. The tests started with the working parameters as used in the project.

Explanation of the parameters used in the following tables:

- $|V^*|$: upper bound for neighbors produced in one iteration,
- m: neighbors per stc generated,
- $nbmax$: stop criteria, the numbers of iterations without improvement,
- u_1: lower bound of the tabu list length,
- u_2: upper bound of the tabu list length,
- $\varnothing\ f$: average of objective function values of conflictless runs,
- $min\ f$: minimum objective function value (the most important parameter for real usage),
- $dev\ f$: standard deviation of objective function values of conflictless runs,
- $\#C$: the number of runs which ended with conflicts,
- $\varnothing t_C$: average time to resolve conflicts of runs which ended conflictless,

$\|V^*\|$	m	$nbmax$	$\varnothing f$	$min\ f$	$dev\ f$	$\#\ C$	$\varnothing t_C$	$\varnothing t$	$dev\ t$
550	10	400	47	6	18	4	12s	54s	25
550	10	400	42	12	18	1	12s	66s	26
550	10	400	44	22	13	1	12s	56s	24
400	8	400	44	12	18	1	10s	43s	15
400	10	400	44	12	17	4	8s	41s	16
400	20	400	47	8	18	2	18s	44s	46
200	5	400	41	6	16	3	8s	32s	11
200	10	400	44	14	16	1	8s	29s	9
200	20	400	46	23	17	3	8s	29s	11
650	10	400	46	10	19	1	13s	64s	27
800	20	400	50	6	19	1	12s	63s	28
1	1	400	-	-	-	all	-	17s	10
550	10	200	48	31	15	3	10s	31s	9
400	10	600	47	13	17	1	8s	51s	21
550	10	600	45	12	20	2	13s	68s	28
1100	20	600	42	7	17	2	16s	104s	40

Table 6.1.: Investigation of parameters $|V^*|$, m and $nbmax$, where $u_1 = 5$, $u_2 = 8$, $p_0 = 5$ and $p_1 = 1$ were fixed. Forty runs were made per experiment.

u_1	u_2	$\varnothing f$	$min\ f$	$dev\ f$	$\#\ C$	$\varnothing t_C$	$\varnothing t$	$dev\ t$
5	5	48	12	20	6	13s	48s	19
6	6	41	11	19	6	15s	55s	27
7	7	40	11	17	1	9s	51s	19
8	8	45	7	17	2	10s	48s	19
9	9	37	12	12	2	9s	54s	22
10	10	40	13	14	0	11s	56s	20
3	7	48	14	20	1	13s	57s	27
3	12	39	11	17	2	16s	64s	25
5	6	43	16	15	5	13s	49s	21
5	7	46	9	17	3	12s	53s	22
5	9	42	9	18	4	13s	57s	29
5	10	39	10	14	2	10s	55s	19
6	7	43	8	21	7	13s	53s	25
6	8	42	10	19	0	12s	54s	21

Table 6.2.: Investigation of parameters u_1 and u_2, where $|V^*| = 550$, $m = 10$, $nbmax = 400$, $p_0 = 5$ and $p_1 = 1$ were fixed. Forty runs were made per experiment.

- $\varnothing t$: average time used by the tabu search program,
- $dev\ t$: standard deviation for the time used by the tabu search program.

Only the conflictless runs were used for the evaluation of $\varnothing f$, $dev\ f$ and $\varnothing t_C$, because the runs with conflicts give much bigger numbers which would destroy the meaning of these parameters. The parameters p_0 and p_1 were not tested because the values of these parameters were not so important in the program.

These experiments can only give an idea of the influence of the parameters since 40 runs are too few per experiment in order to eliminate random effects. There was no time left in the project to increase the number of runs. The creation of a compact solution is harder than the creation of a conflictless solution. The upper and the lower bound of the tabu list length are not so important since there is no significant difference. So far it cannot be said which of the two strategies (many fast runs vs a few slow runs) is the better.

Some ideas for improving the tabu search program resulting from tests: The problem of compactness should be treated differently than the resolving of conflicts. The use of different parameters for the conflict resolving phase and the compactness improving phase may help in a first step.

The following optimal parameters were chosen from the results of these tests:

$$u_1 = 5, u_2 = 8, |V^*| = 600, m = 15, nbmax = 600$$

$\varnothing f$	$min\ f$	$dev\ f$	$\#\ C$	$\varnothing t_C$	$\varnothing t$	$dev\ t$
45	8	14	2	11s	69s	28
44	8	18	2	12s	71s	29
45	8	18	3	11s	72s	31

∅ iterations	∅ total neighbors	∅ tabus	∅ aspirations
2656	1400920	59985	1810
2700	1443545	76742	2033
2723	1451237	66938	1934

$\varnothing f_{StartConflicts}$	$\varnothing f_{StartCompactness}$	∅ conflictless iter.	$\varnothing f_{FirstConflictless}$
3138	450	490	173
3108	450	516	171
3135	445	490	172

Table 6.3.: Runs made with the optimal parameters. Hundered runs were made per experiment.

Parameter tests with the optimal parameters are shown in table 6.3. The parameters ∅ iterations, ∅ total neighbors, ∅ tabus, ∅ aspirations and $\varnothing f_{StartConflicts}$ have a very high standard deviation. The number of aspirations is the number of tabu moves where the tabu status was cancelled. The number of tabus gives the number of tabu

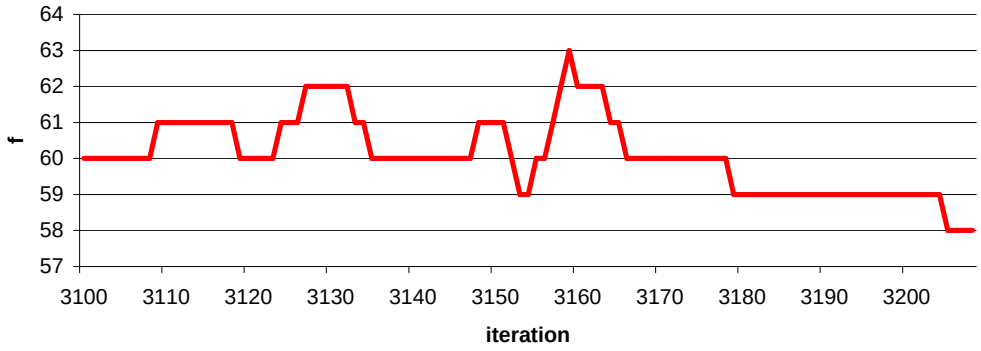

Figure 6.1.: A typical objective function development in a run.

neighbors which couldn't aspire. As the aspired neighbors are normal neighbors they are counted in the total number of neighbors, and correspondingly the tabu neighbors are not counted in the total number of neighbors.

A typical development of the objective function f is shown in the figure 6.1. The ability of tabu search to get out of local optimas is easily seen.

6.2. Comparison with the Existing Solution

The timetable obtained by the tabu search is compared to the one generated by Sargans. A qualitatively good solution fulfills all restrictions given in section 2.2. The compactness is an intuitive measure and depends on the involved persons. The table 6.4 gives some criteria and shows an overview.

	Criteria	**Administrator Sargans**	**Tabu Search Program (*TT*)**
a	restriction 1 (availability)	sometimes relaxed	always respected
b	restriction 2 (feasiblity)	respected	respected
c	restriction 3 (feasiblity)	respected	respected
d	restriction 4 (course splitting)	flexible splitting, usually double lessons	splitting into double lessons
e	restriction 5 (blocking)	respected	respected
f	restriction 6 (lunch break)	flexible	less flexible, but respected
g	restriction 7 (class room)	respected	dropped
h	restriction 8 (several classes)	respected	respected
i	restriction 9 (compactness)	too compact	room for improvement
j	restriction 10 (same length)	respected	respected
k	special courses	first done	blocked
l	time to create a timetable	480 h	1 run = 1 - 2 min
m	choice of timetables	no	yes, with serveral runs

Table 6.4.: Criteria and overview of the two timetables.

Assumptions made in this comparison:

- Every subject is treated the same. No difference is made between main or optional courses.

The tabu search program is denoted as *TT*. Comments on the comparison given in the table are listed below:

Criteria (a) If the school administrator has problems in placing *stc's* and does not see another possibility he speaks with the involved teacher in order to relax the

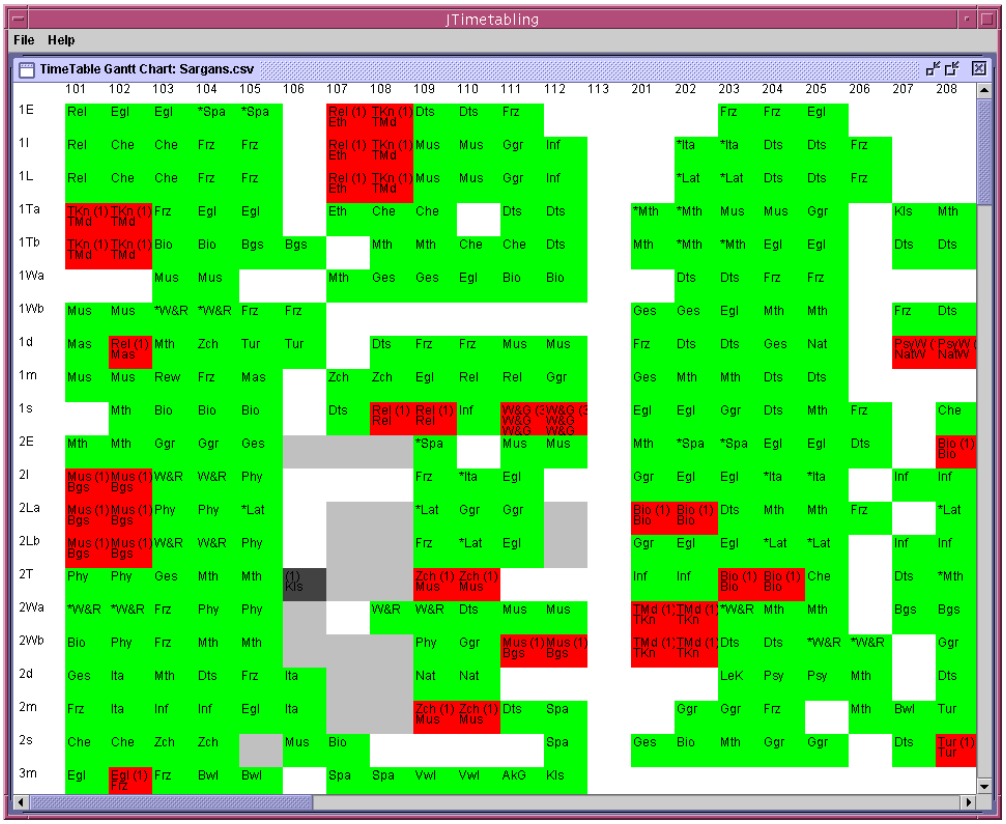

Figure 6.2.: Original solution form Sargans.

availability restriction. *TT* respects all restrictions on availability as given in the input file.

Criteria (b) & (c) There is no problem to produce a feasible timetable with *TT*. Some feasibility checks were done in Sargans with Excel Macros. But as the school administrator said it sometimes happened that their timetable had conflicts. These conflicts were detected in the first week of each school year. However, for the most part the schedule of Sargans is feasible.

Criteria (d) Course splitting is implemented in *TT* as a rule to always split courses into double courses. Sargans usually splits courses into double lessons, but relaxes this rule, if they do not see another possibility or if there are other reasons. They say placing double lessons is more complicated than placing single lessons. It would be no problem to allow the setting of specific splitting points in the input files for *TT*. Both solutions do not place splitted courses the same day.

Criteria (e) In both solution approaches blocking of lessons is no problem. The blocking feature is widely used in *TT* in oder to handle special courses.

Criteria (f) Lunch time is handled more flexible in the solution of Sargans. In *TT* a quite simple rule is used: the 6th lesson is blocked and the 7th lesson is punished

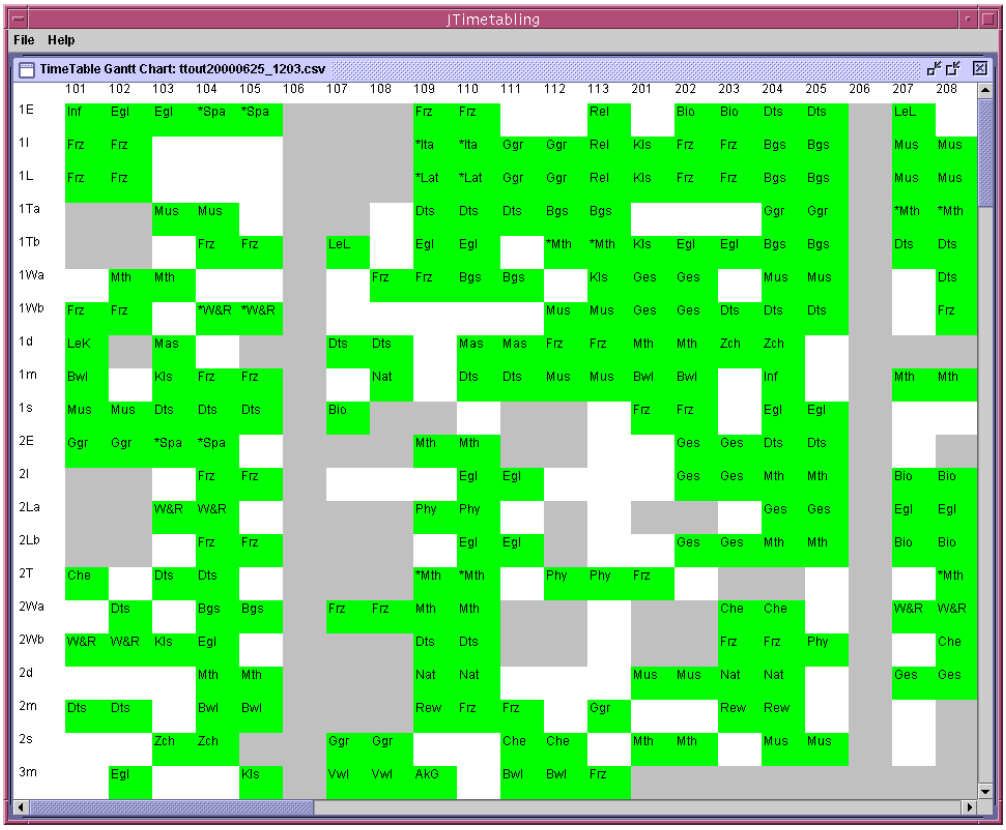

Figure 6.3.: A conflictless, non compact solution of the tabu search program.

in the objective function. This rule allows to fulfill the lunch time restriction in every case, but gives some extra flexibility. There's room to experiment with the lunch time in *TT*. A useful and simple enhancement could be made if either the 5th or the 7th periods are considered as desirable lunch times.

Criteria (g) There was no remaining time to work on the room subproblem in this term project. A global approach can be taken for including the room problem in *TT*. Sargans assigns class rooms on a intuitive criteria.

Criteria (h) Up to three classes can be included in a *stc* in *TT*. If members from more than three classes are taking a course, such courses have to be blocked. Cases with more than three classes occur normally in special courses, which are blocked anyway. The school administrator handles this intuitively.

Criteria (i) The compactness of timetables of *TT* can be improved. More sophisticated rules can be introduced into the objective function. Further analysis and experiments should be done for more compact timetables. Timetables produced by *TT* usually starts too late in the morning, since this is not punished based on a wrong assumption. Free morning periods should be treated different than free afternoons periods. As mentioned in (f), lunch time should be made more flexible in order to achieve better compactness. As the school administrator said their timetable is usually too compact in the sense that there are not enough gaps.

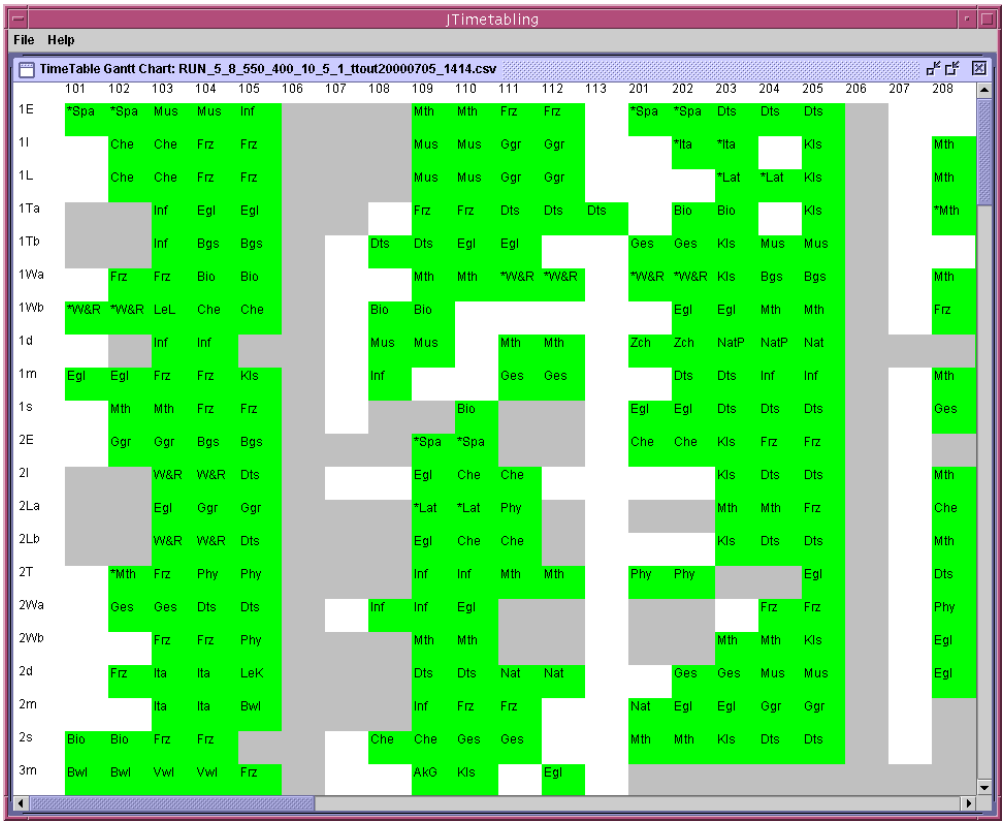

Figure 6.4.: A more compact solution form the tabu search program.

He said it's desired to have about three gaps per class in a week. The gaps are normally used for private lessons, e.g. a pupil learning to play an instrument. Private lessons are directly scheduled between the teacher and the pupil. TT tries to respect this desire of having three free periods in a week and does not penalize the first three gaps.

Criteria (j) The time length restriction is respected by *TT* and Sargans.

Criteria (k) Special courses, which require special knowledge can be handled by *TT* and in Sargans. They do it in a similar way. A special timetable for special courses is done first in Sargans, because special course assignments require full freedom of choice on the timetable. Some parts of a special timetable are not even done by the school administrator, but by the involved persons themselves. Assigned special courses are considered as fixed. The same blocking can be used in *TT* and therefore blocking is a very natural operation. A special timetable has to be done in every case.

Criteria (l) A significant amount of time is invested in Sargans for building a timetable. *TT* can create a timetable in a very short time. For obtaining good results several runs of *TT* are necessary. Several runs are supported by *TT* in using scripts and reporting the essential information to the user. A special timetable must still be done in both approaches.

Criteria (m) Several runs can be made in a short time with *TT*. This gives the possibility to create a few good timetables. These timetables can be compared and the best can be taken by the administrator. Another possibility of an approach with a computer is the variation of objective function strategies as well as changing the input data.

For comparison the timetable of Sargans is measured with the compactness objective function used in *TT*. Sargans achieved a compactness penalty of 91. This is a big value compared with *TT*. This can be explained by not differentiating between blocked lessons and gaps as well as the inflexibility of the compactness measure used in the objective function.

One part of the original schedule of Sargans is shown in the image 6.2. The meaning of the colors is given in the description of the viewer program in section 5.6. The red conflicts appear in the timetable of Sargans because classes are divided into half classes for optional courses. Dark gray conflicts with blockings appear because some optional courses had to blocked in the input file since these data could not be directly handled. A conflictless solution which is not compact is shown in the image 6.3. A more compact solution is shown in the image 6.4.

Summarized it can be said that the current version of *TT* can be used as a guide for the school administrator. If the room problem is included and the compactness improved, then *TT* should be fully usable.

7. Future Work

In this section I would like to show some interesting approaches and improvements that could be done.

Different compactness objective functions can be introduced and compared. The compactness evaluation is quite simple and can be improved for better satisfying the intuitive quality measures of an experienced school administrator. For example morning gaps and afternoon gaps can be treated differently or the lunch break could be made more flexible.

The room subproblem was dropped because there wasn't enough time for both the time and the room problem. The room subproblem can be done in a similar way like the time subproblem. These subproblems can be considered together in the proposed global approach.

The neighborhood is an important part of the tabu search metaheuristic and has to be carefully chosen. The neighborhood can be expanded for example by introducing exchanges between *stc's*. Special moves like moving all afternoon lessons can be introduced for compactness.

More aspects of the timetabling problem of a high school can be considered, e.g. the grouping problem. The grouping problem considers the grouping of pupils for optional courses. The grouping problem can also be included in a global approach.

Bibliography

[1] A. Hertz, "Tabu search for large scale timetabling problems", *European J. of Operational Research*, 54/1, 39-47, 1991.

[2] A. Hertz, "Finding a feasible course schedule using Tabu Search", *Discrete Applied Mathematics*, 35, 255-270, 1992.

[3] *Annals of Operations Research*, Tabu search, 41, 1993, Editor in chief: Peter L. Hammer.

[4] F. Glover, E. Taillard and D. de Werra, "A user's guide to tabu search", *Annals of Operations Research*, Tabu search, 41, 12-37, 1993, Editor in chief: Peter L. Hammer.

[5] H.A. Eiselt and G. Laporte, "Combinatorial optimization problems with soft and hard requirements", *Journal of the Operational Research Society*, 38, 785-795, 1987.

[6] J. Aubin and A. Ferland, "A large scale timetabling problem", *Computers and Operations Research*, 16/1, 67-77, 1989.

[7] G. Laport and S. Desroches, "The problem of assigning students to course sections in a large engineering school", *Computers and Operations Research*, 13, 387-394, 1986.

[8] R.C. Charlson and G.L. Nemhauser, "Scheduling to minimize interaction cost", *Operations Research*, 14, 52-58, 1966.

[9] S. Even, A. Itai and A. Shamir, "On the complexity of timetable and multicommodity flow problems", *SIAM J. Comput.*, 5, 691-703, 1976.

[10] F. Glover, "Tabu Search - Part I", *ORSA Journal on Computing*, 1, 190-206, 1989.

[11] F. Glover, "Tabu Search - Part II", *ORSA Journal on Computing*, 2, 4-32, 1990.

[12] A. Hertz and D. de Werra, "The tabu search metaheuristic: how we used it", *Annals of Mathematics and Artificial Intelligence*, 1, 111-121, 1990.

[13] F. Glover, "Tabu Search - Wellsprings and Challenges", *European Journal of Operational Research*, 106, 221-225, 1998.

[14] W. Domschke, R. Klein und A. Scholl, "Tabu Search", `http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/tabu/index.htm`, 1996.

[15] M. Burkard, M. Cochand and A. Gaillard, “A Dynamical System Based Heuristic for a Class of Constrained Semi-Assignment Problem”, `http://www.ifor.math.ethz.ch/~burkard/index.en.html`, 1998.

[16] M. Burkard, “A Heuristic Method for the C-SAP”, `http://www.ifor.math.ethz.ch/~burkard/index.en.html`, 1999.

[17] B. Stroustrup, “Die C++ Programmiersprache”, Addison-Wesley, 2. Auflage, 1992.

[18] U. Breymann, “Die C++ Standard Template Library”, Addison-Wesley, 1996.

[19] J. Ortega and A. Grimshaw, “C++ and Numerical Methods”, Oxford University Press, 1999.

[20] Gamma et al., “Design Patterns”, Addison-Wesly, 1995.

[21] Sun Microsystems, “Optional Package Advanced Imaging for Java”, `http://java.sun.com/products/java-media/jai/`.

A. Program Documentation

A.1. Tabu Search Program

Several parameters can be set by the user. These parameters must be given in the command line.

```
TimeTabling stcFile subjectsFile teachersFile ↩
classesFile timesFile lowerTabu upperTabu neigh nbmax ↩
m p0 p1 reportName quiet
```

Example:

```
TimeTabling ../data/stc6.csv ../data/subjects.csv ↩
../data/teachers6.csv ../data/classes6.csv ↩
../data/times.csv 5 8 550 400 10 5 1 ../out/RUN_A_ 6
```

Description of the program parameters:

- `stcFile`: file name of the file containing the *stc's*
- `subjectsFile`: file name of the file containing the subjects
- `teachersFile`: file name of the file containing the teachers and their availability
- `classesFile`: file name of the file containing the classes and their availability
- `timesFile`: file name of the file containing all time periods
- `lowerTabu`: lower bound of the tabu list length (model: u_1)
- `upperTabu`: upper bound of the tabu list length (model: u_2)
- `neigh`: maximum number of neighbors to generate (model: $|V^*|$)
- `nbmax`: number of iterations without improvement, used as stop rule (model: $nbmax$)
- `m`: number of neighbors generated for one *stc* (model: m)
- `p0`: penalty for conflicts (model: p_0)
- `p1`: penalty for uncompactness (model: p_1)
- `reportName`: file name beginning for report and summary files

- `quiet`: level of output information (0: no information; 5: information at end of run; 6: objective function value updated on screen; 10 full information)

There is a script for automating multiple runs.

A.2. Viewer Program

As the viewer program is a Java program it needs for running a Java Virtual Machine (JVM). Java Platform 2 Version 1.3 (J2SE JRE1.3) is need. For making images of timetables the optional package "Java Advanced Imaging 1.0.2" is need as well. The package can be obtained by Sun Microsystems [21].

The program can be started with the following parameters:

```
java -Xmx100m JTimeTabling
```

In the file `JTimetabling.properties` the files for teachers, classes and times has to be set:

```
DEFAULT_DIR = ../out/
CLASSES_FILE = ../data/classes6.csv
TEACHERS_FILE = ../data/teachers6.csv
TIMES_FILE = ../data/times.csv
```

Solution files can be chosen for viewing in the menu Files→Open.

B. Data Files

B.1. Preassignments: Subject-Teacher-Classes

The data format of file containing the preassignments (*stc's*) is given in Extended Backus Nauer Form (EBNF):

```
stcFile = header {stc}.
header = {characters} newline.
stc = time ‘;‘ room ‘;‘ teacher ‘;‘ subject ‘;‘ class1 ‘;‘ class2 ‘;‘ class3 ‘;‘ newline.
time = number.
room = {character}.
teacher = character [character [character [character]]]
        | ‘"‘ character [character [character [character]]] ‘"‘.
subject = character [character [character [character]]]
        | ‘"‘character [character [character [character]]] ‘"‘.
class1 = character [character [character [character]]]
        | ‘"‘ character [character [character [character]]] ‘"‘.
class2 = character [character [character [character]]]
        | ‘"‘ character [character [character [character]]]‘"‘.
class3 = character [character [character [character]]]
        | ‘"‘ character [character [character [character]]]‘"‘.
```

The beginning of the preassignment file is shown below with 216 of 1164 preassignments (*stc's*).

```
"time";"room";"teacher";"subject";"class1";"class2";"class3";
110;14;"Ca";"Spa";"4sA";"4sB";"4m";
112;14;"Ca";"Spa";"2m";"2s";"3t";
206;14;"Ca";"Spa";"5gA";"5gB";"3wA";
207;14;"Ca";"Spa";"5gA";"5gB";"3wA";
210;14;"Ca";"Spa";"2m";"2s";"3t";
407;14;"Ca";"Spa";"4sA";"4sB";"4m";
408;14;"Ca";"Spa";"2m";"2s";"3t";
409;14;"Ca";"Spa";"5gA";"5gB";"3wA";
101;23;"Ga";"Rel";"1E";"1I";"1L";
506;23;"Hp";"Rel";"5gB";"3wB";"3t";
507;23;"Hp";"Rel";"5gB";"3wB";"3t";
211;62;"Kr";"EglW";"6gA";"6gB";"4t";
212;62;"Kr";"EglW";"6gA";"6gB";"4t";
207;84;"Ul";"FrzW";"4wA";"4wB";"6gB";
208;84;"Ul";"FrzW";"4wA";"4wB";"6gB";
406;"Med";"Hr";"Aut";"5sA";"5sB";;
508;46;"Gd";"Bgs";"1L";"1I";;
509;46;"Gd";"Bgs";"1L";"1I";;
304;24;"Li";"Bio";"1L";"1I";;
303;24;"Li";"Bio";"1L";"1I";;
402;29;"Zm";"Bio";"2Lb";"2I";;
403;29;"Zm";"Bio";"2Lb";"2I";;
102;31;"Sk";"Che";"1L";"1I";;
```

```
103;31;"Sk";"Che";"1L";"1I";;
308;31;"Fr";"Che";"2Lb";"2I";;
307;31;"Fr";"Che";"2Lb";"2I";;
401;3;"Ze";"Dts";"1L";"1I";;
402;3;"Ze";"Dts";"1L";"1I";;
302;6;"Ze";"Dts";"1L";"1I";;
205;72;"Ze";"Dts";"1L";"1I";;
204;72;"Ze";"Dts";"1L";"1I";;
404;84;"Vt";"Dts";"2Lb";"2I";;
301;84;"Vt";"Dts";"2Lb";"2I";;
502;84;"Vt";"Dts";"2Lb";"2I";;
301;61;"Ki";"Egl";"1L";"1I";;
504;61;"Ki";"Egl";"1L";"1I";;
503;61;"Ki";"Egl";"1L";"1I";;
209;61;"Ki";"Egl";"1L";"1I";;
203;15;"Ro";"Egl";"2Lb";"2I";;
202;15;"Ro";"Egl";"2Lb";"2I";;
111;15;"Ro";"Egl";"2Lb";"2I";;
309;3;"Kr";"EglW";"4wA";"4wB";;
308;3;"Kr";"EglW";"4wA";"4wB";;
307;41;"Am";"Eth";"1I";"1E";;
307;49;"Lu";"Eth";"1L";"1Ta";;
206;15;"Ro";"Frz";"1L";"1I";;
505;15;"Ro";"Frz";"1L";"1I";;
104;15;"Ro";"Frz";"1L";"1I";;
105;15;"Ro";"Frz";"1L";"1I";;
506;14;"Ul";"Frz";"2Lb";"2I";;
212;84;"Ul";"Frz";"2Lb";"2I";;
109;84;"Ul";"Frz";"2Lb";"2I";;
309;13;"Hs";"Ges";"1L";"1I";;
310;13;"Hs";"Ges";"1L";"1I";;
210;49;"Hs";"Ges";"2Lb";"2I";;
211;49;"Hs";"Ges";"2Lb";"2I";;
111;43;"Ks";"Ggr";"1L";"1I";;
502;43;"Ks";"Ggr";"1L";"1I";;
501;43;"Ks";"Ggr";"2Lb";"2I";;
201;41;"Ks";"Ggr";"2Lb";"2I";;
112;18;"Vo";"Inf";"1L";"1I";;
407;18;"Vo";"Inf";"1Ta";"1Tb";;
301;18;"Vo";"Inf";"1Wa";"1Wb";;
207;18;"Mü";"Inf";"2Lb";"2I";;
208;18;"Mü";"Inf";"2Lb";"2I";;
102;12;"Wb";"Ita";"2d";"2m";;
106;6;"Wb";"Ita";"2d";"2m";;
209;15;"Es";"Ita";"2La";"2Lb";;
311;16;"Es";"Ita";"2La";"2Lb";;
211;11;"Wb";"Ita";"2s";"2T";;
407;5;"Wb";"Ita";"2s";"2T";;
307;16;"Es";"Ita";"2Wa";"2Wb";;
210;15;"Es";"Ita";"2Wa";"2Wb";;
310;16;"Es";"Ita";"3sA";"3sB";;
212;15;"Es";"Ita";"3sA";"3sB";;
306;16;"Es";"Ita";"5gA";"5gB";;
211;15;"Es";"Ita";"5gA";"5gB";;
311;31;"Sk";"Kls";"1L";"1I";;
312;6;"Hs";"Kls";"2Lb";"2I";;
206;11;"Ak";"LatW";"3wC";"3t";;
305;4;"Es";"LeL";"1L";"1I";;
210;17;"Re";"Mth";"1L";"1I";;
404;14;"Re";"Mth";"1L";"1I";;
211;17;"Re";"Mth";"1L";"1I";;
403;14;"Re";"Mth";"1L";"1I";;
505;17;"Wo";"Mth";"2Lb";"2I";;
504;17;"Wo";"Mth";"2Lb";"2I";;
406;60;"Wo";"Mth";"2Lb";"2I";;
302;21;"Wo";"Mth";"2Lb";"2I";;
```

```
109;99;"Bs";"Mus";"1L";"1I";;
110;99;"Bs";"Mus";"1L";"1I";;
102;96;"Kt";"Mus";"1Wb";"1m";;
101;96;"Kt";"Mus";"1Wb";"1m";;
105;37;"Gl";"Phy";"2Lb";"2I";;
309;37;"Gl";"Phy";"2Lb";"2I";;
310;37;"Gl";"Phy";"2Lb";"2I";;
510;28;"Hp";"Rel";"5gB";"3wB";;
509;28;"Hp";"Rel";"5gB";"3wB";;
407;28;"Ga";"Rel";"6gA";"6gB";;
406;4;"Vg";"Spa";"3sA";"4t";;
406;14;"Ca";"Spa";"3wB";"3wC";;
104;14;"Sh";"W&R";"2Lb";"2I";;
103;14;"Sh";"W&R";"2Lb";"2I";;
308;19;"Vi";"*Ita";"1I";;;
405;19;"Vi";"*Ita";"1I";;;
203;28;"Vi";"*Ita";"1I";;;
202;28;"Vi";"*Ita";"1I";;;
401;14;"Vi";"*Ita";"2I";;;
110;19;"Vi";"*Ita";"2I";;;
205;19;"Vi";"*Ita";"2I";;;
204;19;"Vi";"*Ita";"2I";;;
203;13;"Ht";"*Lat";"1L";;;
507;13;"Ht";"*Lat";"1L";;;
202;13;"Ht";"*Lat";"1L";;;
405;11;"Ht";"*Lat";"1L";;;
410;11;"Ak";"*Lat";"2La";;;
109;11;"Ak";"*Lat";"2La";;;
105;11;"Ak";"*Lat";"2La";;;
208;11;"Ak";"*Lat";"2La";;;
305;11;"Ak";"*Lat";"2Lb";;;
205;11;"Ak";"*Lat";"2Lb";;;
110;11;"Ak";"*Lat";"2Lb";;;
204;11;"Ak";"*Lat";"2Lb";;;
202;22;"Bn";"*Mth";"1Ta";;;
508;22;"Bn";"*Mth";"1Ta";;;
201;22;"Bn";"*Mth";"1Ta";;;
503;11;"Gt";"*Mth";"1Tb";;;
203;6;"Gt";"*Mth";"1Tb";;;
202;6;"Gt";"*Mth";"1Tb";;;
301;17;"Re";"*Mth";"2T";;;
208;17;"Re";"*Mth";"2T";;;
209;17;"Re";"*Mth";"2T";;;
401;4;"Vg";"*Spa";"1E";;;
303;62;"Vg";"*Spa";"1E";;;
104;12;"Vg";"*Spa";"1E";;;
105;12;"Vg";"*Spa";"1E";;;
203;14;"Ca";"*Spa";"2E";;;
405;14;"Ca";"*Spa";"2E";;;
202;14;"Ca";"*Spa";"2E";;;
109;14;"Ca";"*Spa";"2E";;;
505;3;"Ee";"*W&R";"1Wa";;;
504;3;"Ee";"*W&R";"1Wa";;;
409;3;"Ee";"*W&R";"1Wa";;;
408;3;"Ee";"*W&R";"1Wa";;;
408;15;"Dd";"*W&R";"1Wb";;;
409;15;"Dd";"*W&R";"1Wb";;;
103;72;"Dd";"*W&R";"1Wb";;;
104;72;"Dd";"*W&R";"1Wb";;;
308;5;"Hü";"*W&R";"2Wa";;;
101;5;"Hü";"*W&R";"2Wa";;;
203;5;"Hü";"*W&R";"2Wa";;;
102;5;"Hü";"*W&R";"2Wa";;;
407;4;"Eg";"*W&R";"2Wb";;;
408;4;"Eg";"*W&R";"2Wb";;;
205;4;"Eg";"*W&R";"2Wb";;;
```

```
206;4;"Eg";"*W&R";"2Wb";;;
307;5;"Hü";"Abs";"4m";;;
205;5;"Hü";"Abs";"4m";;;
204;5;"Hü";"Abs";"4m";;;
305;60;"Ee";"AkG";"2m";;;
111;4;"Eg";"AkG";"3m";;;
410;4;"Eg";"AkG";"4m";;;
409;4;"Eg";"AkG";"4m";;;
103;21;"Si";"AMa";"3t";;;
104;21;"Si";"AMa";"3t";;;
204;21;"Si";"AMa";"4t";;;
203;21;"Si";"AMa";"4t";;;
409;46;"By";"Bgs";"1E";;;
408;46;"By";"Bgs";"1E";;;
501;46;"Gd";"Bgs";"1Ta";;;
502;46;"Gd";"Bgs";"1Ta";;;
106;46;"Cs";"Bgs";"1Tb";;;
105;46;"Cs";"Bgs";"1Tb";;;
311;46;"Gd";"Bgs";"1Wa";;;
310;46;"Gd";"Bgs";"1Wa";;;
504;46;"Gd";"Bgs";"1Wb";;;
503;46;"Gd";"Bgs";"1Wb";;;
506;46;"Gd";"Bgs";"2E";;;
505;46;"Gd";"Bgs";"2E";;;
208;46;"By";"Bgs";"2Wa";;;
207;46;"By";"Bgs";"2Wa";;;
302;24;"Li";"Bio";"1E";;;
301;24;"Li";"Bio";"1E";;;
105;29;"Fl";"Bio";"1s";;;
104;29;"Fl";"Bio";"1s";;;
103;29;"Fl";"Bio";"1s";;;
504;29;"Ac";"Bio";"1Ta";;;
503;29;"Ac";"Bio";"1Ta";;;
104;24;"Ac";"Bio";"1Tb";;;
103;24;"Ac";"Bio";"1Tb";;;
112;24;"Zm";"Bio";"1Wa";;;
111;24;"Zm";"Bio";"1Wa";;;
405;29;"Zm";"Bio";"1Wb";;;
404;29;"Zm";"Bio";"1Wb";;;
202;29;"Fl";"Bio";"2s";;;
107;29;"Fl";"Bio";"2s";;;
402;24;"Li";"Bio";"2Wa";;;
401;24;"Li";"Bio";"2Wa";;;
505;29;"Ac";"Bio";"2Wb";;;
101;24;"Ac";"Bio";"2Wb";;;
204;29;"Fl";"Bio";"3sA";;;
203;29;"Fl";"Bio";"3sA";;;
205;29;"Fl";"Bio";"3sB";;;
206;29;"Fl";"Bio";"3sB";;;
502;24;"Co";"Bio";"3t";;;
501;24;"Co";"Bio";"3t";;;
309;24;"Zm";"Bio";"3wA";;;
308;24;"Zm";"Bio";"3wA";;;
110;24;"Zm";"Bio";"3wB";;;
109;24;"Zm";"Bio";"3wB";;;
310;24;"Zm";"Bio";"3wC";;;
311;24;"Zm";"Bio";"3wC";;;
509;29;"Ft";"Bio";"4sA";;;
508;29;"Ft";"Bio";"4sA";;;
511;29;"Ft";"Bio";"4sB";;;
510;29;"Ft";"Bio";"4sB";;;


...
```

B.2. Teachers

The format of the teacher input file including the availability in EBNF:

teacherFile = header {teacher}.
header = {characters} newline.
teacher = name ‘;‘ {notAvailable ‘;‘} newline.
name = character {character} | ‘"‘ character {character} ‘"‘.
notAvailable = number.

The first four teachers with their blocked time periods were shown as example:

```
"Teacher"; "not available or blocked";
"Ac";106;206;306;406;506;601;602;603;604;605;501;502;
"Ak";106;206;306;406;506;601;602;603;604;605;101;102;103;104;508;509;510;511;512;-
513;106;112;407;408;409;108;
"Am";106;206;306;406;506;601;602;603;604;605;111;112;113;107;
"Be";106;206;306;406;506;601;602;603;604;605;111;112;113;201;202;203;204;205;206;-
207;208;209;210;211;212;213;401;402;403;404;405;406;407;408;409;410;411;412;413;-
501;502;503;504;505;506;507;508;509;510;511;512;513;111;112;
```

B.3. Classes

The format of the class input file including the availability in EBNF:

classFile = header {class}.
header = {characters} newline.
class = name ‘;‘ {notAvailable ‘;‘} newline.
name = character {character} | ‘"‘ character {character} ‘"‘.
notAvailable = number.

Four classes with their blocked time periods were shown as example:

```
 "Classes";"not available or blocked";
"1d";106;206;306;406;506;601;602;603;604;605;105;106;404;102;207;208;
"1E";106;206;306;406;506;601;602;603;604;605;108;510;511;108;510;511;107;404;405;
"1I";-106;-206;-306;-406;-506;-601;-602;-603;-604;-605;-108;-510;-511;-108;-510;-
511;-107;
"4sA";106;206;306;406;506;601;602;603;604;605;305;404;405;504;405;404;405;108;-
107;105;510;511;111;306;207;208;209;210;211;101 -102;307;308;311;312;
```