Some Results in Automatic Timetabling

Martin Löhnertz

Institut für Informatik V
University of Bonn

This is an english sketch of some results of my diploma thesis of 1999. Some parts have been presented at the PATAT 2002 conference. The figures can be found at the end of the paper.

1 Abstract

I report some results archived in two projects in the theory and practice of automatic timetabling in 1997, 1998 and spring 1999. I have built an interactive program which was tested on real data. In this paper I focus on some theoretical aspects and some design ideas. A complete description (in german) can be found in [14].
On the theoretical side, I present a new application of the weigthed bipartite matching algorithm according to Koenig/ Hall/ Kuhn and give a short description and a small extension of the theorem of Galvin in the context of timetabling.
On the practical side I discuss special features of the solution space and their effects on metaheuristics and show how the models of Cooper & Kingston[8] and those of Burke et al. [3] can be combined with an user interface to form an easily adaptable timetabling-system. Advantages and disadvantages of such a general model are discussed.
I especially thank Prof. Dr. A.B. Cremers, Dipl. Math C. Hundack and Dipl. Inform. J. Luessem from the Institut für Informatik III at the University of Bonn, who supported my work and gave many usefull advices considering my thesis and this paper.

2 Theoretical Aspects

Meanwhile most of the theoretical problems which have been used as models for the timetabling problem have turned out to be not only $\mathcal{NP}$ hard but even not approximatable to a constant factor [12]. Therefore this theoretical section is dedicated to relatively plain structured problems which build the frontier between the (theoretically) tractable and intractable problems.

2.1 Applying Bipartite Matching as a "Filter"

It is well known that the unconstrained school timetabling problem can be solved by decomposing a bipartite graph into matchings (sets of independent edges). As has been pointed out by Chahal and DeWerra [6] using flow techniques, one can also apply some edge weights and try to archive additional goals, although of course no provable success can be guaranteed. A more direct view on the same problem is to consider weighted matchings in bipartite graphs. Here the theorem of Hall and the pigeonhole principle provide the existence of a matching covering all vertices of maximum degree (what of course is implicitly used by [6]). With the following "big M"-type method it is therefore possible to include arbitrary parameters instead of setting the capacities of preceding edges in a network:
Let $c : E \to \mathrm{R}_+$ be an arbitrary weight function and $M := 1 + \sum_{e \in E} c(e)$. Now we denote by $\tilde{\Delta}$ the set of maximum degree vertices. The modified cost function is defined as

$$c'(e) := M \ |e \cap \tilde{\Delta}| + c(e)$$

Obviously a matching which covers all vertices of maximum degree is "heavier" than any matching not doing so. For a pair of matchings which cover the same number of vertices in $\tilde{\Delta}$ the one which is better according to c is preferred. Therefore applying an algorithm like the "Hungarian"[13] to this new function provides a matching covering $\tilde{\Delta}$ fulfilling the requests coded in c as good as possible.
Removing the chosen edges from the graph therefore still reduces the maximum degree by 1 allowing a decomposition in $\Delta(G)$ matchings.
Most timetabling problems are not as simple as that. They often request more than two objects (one class and one teacher) to be scheduled simul-

taneously. This can interpreted as a hypergraph edge coloring problem where the vertices V are the objects and the edges E are the lessons consisting of all objects taking part. If one can find two sets C, T satisfying $|e \cap C| \geq 1 \wedge |e \cap T| \geq 1 \; \forall e \in E$ then obviously a mapping $\phi : E \to C \times T$ can be created. That is, one simply takes one "teacher" and one "class" from each lesson. Let S denote the set of timeslots and π_i the projection on the ith component. We will call a Timetable $\psi : E \to S$ ct-feasible according to ϕ if $\psi(e) = \psi(f) \to \pi_i(\phi(e)) \neq \pi_i(\phi(f)) \forall f, e \in E \forall i \in \{1, 2\}$. Obviously a feasible timetable is ct-feasible according to any ϕ.
Now the weighted decomposition algorithm described above can be used to create a mapping F between timetables. The first step is to reduce the range of the weight function to $\{0, 1\}$ and to assign values the same way as a binary representation of the matching decomposition would be created, i.e. one sets the weight of e to 1 in the ith step if and only if e is scheduled in Timeslot i in a given "starting" timetable. By applying the algorithm this "starting" timetable is mapped to a new one. This mapping has the interesting properties that:

1. Every timetable is mapped to a ct-feasible one.
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.

It must be noted, that this decomposition is not necessarily the feasible one most similar to the input in euclidean (or generally $|.|_p$) distance but in a lexicographical distance depending on the order of the timeslots.
There are two ways to include this algorithm in the framework of a general

local search technique. The simplest, but most time consuming one is to include it in any evaluation of the quality function Q simply by using $F \circ Q$ instead of Q. The other is similar to that used in memetic [4] algorithms: After some local steps the mapping is applied to reach a ct-feasible solution near the current position. While the first one heavily smoothes the profile of the quality function the second one is - especially in the context of tabu search - much faster, because only one evaluation is needed. Further details can be found in the last section or in [14].

2.2 The Theorem of Galvin

Although the importance of graph coloring algorithms in the context of timetabling is widely accepted [9], the famous theorem of Galvin (1994) [10] seems not to have received the focus it deserves. We therefore give a short introduction to the corresponding problem and its solution:
Let $G = (V, E)$ be a graph and $F = \{1, \cdots, f\}$ be a set of colors. Additionally let $g \in F_{cl} := \{h : V \to \{X | X \subseteq F, |X| \geq cl\}\}$ be a function assigning a subset of F to each vertex of G. We will call $g(v)$ the set of admittable colors for v. The problem is to find a coloring c of G (i.e. a mapping $c : V \to F$ satisfying $(v_1, v_2) \in E \Rightarrow c(v_1) \neq c(v_2)$) with the additional restriction that $c(v) \in g(v)\ \forall v \in V$. The minimum cl allowing such a coloring for all $g' \in F_{cl}$ is called the "list-chromatic-number"cl of G.
The theorem of galvin proofs the so called list coloring conjecture of Dinitz for bipartite graphs. Let $L(G)$ denote the line-graph of G (i.e. $L(G) = (E, L)$, $(e_1, e_2) \in L \Leftrightarrow e_1 \cap e_2 \neq \emptyset$) and $\Delta(G)$ denote the maximum vertex degree.

$$G \text{ bipartite } \rightarrow cl(L(G)) = \Delta(G)$$

The proof is constructive and a corresponding algorithm can be found for

example in [16]. When considering timetabling problems, most unavailability constraints are related to objects and not to lessons. Modeling the classical school-teacher timetabling problem according to the theorem of König [9] the "availability" lists are associated with the vertices of the graph itself and not with those of the linegraph. To apply Galvin's method one has to decide which unavailability an edge representing a course inherits from the incident vertices. For simplicity we shall assume that all classes are available all the time. We therefore can simply assign the list to the teachers. Interpreted in this way, a weaker version of the Theorem of Galvin can be stated as:
Given a class-teacher timetabling problem, let t be the maximum number of timeslots and m be the maximum number of lessons any object is involved in. If one assigns $(t-m)$ unavailabilities to each teacher there exists a feasible plan respecting these. Especially this plan can be found in polynomial time. Generalizations of the theorem as found in [17] can not be applied because the degree of the "classes" in practice often equals $\Delta(G)$. But following this direction we will now consider cases in which every vertex/edge can have its own listsize i.e. we generalize to $F_{sv_1,\cdots,sv_l} := \{h : V \to \{X|X \subset F\}, |h(v_i)| = sv_i\}$.
Most of the choosability-problems on graphs have turned out to be Π_2 complete [17]. The theorem of Galvin provides an criterion for a special case of linegraph choosability. One might expect that there must be problems lying between these two extrema. Problems belonging to $\mathcal{P}$ can be found in [17]. We now shortly present a subproblem belonging to $\mathcal{NP}$, which is not known to belong to $\mathcal{P}$.
We again start with the basic structure described above: Unavailabilities are associated with the teachers. i.e. all edges incident to the same vertex in the first partition share a common list. This is relaxed to the request that these

lists are of equal size.
Our construction is based on the proof of Galvin's theorem as it is found in [10]. We give a very short sketch of the proof. For details see the original paper or [10]. The proof can be gratuitously divided into two major steps:

1. A graph with an orientation, in which every subgraph contains a kernel and $|g(v)| < \delta^+(v)$(δ^+ describing the Indegree) can be colored according to the lists. A Kernel is an independant dominating set.

2. The linegraph of bipartite graph can be given an orientation satisfying the conditions above and
   $\max_{e \in L(G)} \delta^+_{L(G)}(e) < \Delta(G)$.

We concentrate on the second step. Let F denote the left partition (teachers) and S denote the right partition (classes). Again we subdivide it in some steps:

1. Let an ordinary (i.e. non-list) coloring $\phi : E \to \{1, \cdots, \Delta\}$ be given on L(G) (Alg. König/Hall).

2. For every edge $k = (e_1, e_2) \in E(L(G))$ let the orientation be $e_1 \to e_2$ if and only if the vertex generating k is in F XOR $\phi(e_1) < \phi(e_2)$, and $e_2 \to e_1$ otherwise.

As is proved in [10] any subgraph of the resulting graph contains a kernel. We will concentrate on the indegrees effected by a fixed initial coloring. A central observation in the proof of the theorem is, that following the described construction any edge colored l is reached in the oriented linegraph only by edges with larger color adjacent via F and from edges with smaller color from S.
Therefore the Number of preceeding edges can be at most $\Delta(G) - 1$ because

there are $l-1$ smaller and $\Delta(G)-l$ larger colors.
Now assume that an edge e is colored with the smallest possible color. Then the indegree is equal to the degree of the vertex $e \cap F$. Choosing the second smallest color for another edge f emanating from that vertex the number of edges in $L(G)$ directed towards f is also bounded by $\delta(e \cap F)$ because f cannot be reached from e and at most one edge can appear from S because there is only one smaller color. Generally if only the first $j \geq \delta(e \cap F)$ colors are used the indegree effected is smaller than j.

FIG I

This together with the rest of Galvin's proof yealds:

Theorem 1 *A linegraph of a bipartite graph is choosable with listsizes* $s(e), (e \cap F = f \cap F) \rightarrow s(e) = s(f)$ *if and only if it is colorable with lists* $l(e) = \{1, .., s(e)\}$

(The reverse direction beeing implicated by the definition of chooasability.)
This Theorem contains the Theorem of Galvin as a special case: If all lists are of the same size, the coloring problem is identical to ordinary graph coloring. The colorability problem is in $\mathcal{NP}$ therefore this special choosability-problem is in $\mathcal{NP}$, too. Should this colorability problem be in $\mathcal{P}$ some more unavailability problems would become solvable. But one has to accept that choosability is much stronger than colorability in general. A coloring problem may be solvable although it is not choosable with the corresponding list sizes.

3 Practical Aspects

3.1 Some Observations on Problem Structure and Metaheuristics

Not only since the paper of [7] is known, that the problem of timetabling is hard in several aspects destroying any hope for finding exact or even approximate algorithms without proving $\mathcal{P} = \mathcal{NP}$. Therefore - if one does not want to resort to direct heuristics, which are completely based on pure experimental experience and require lots of this - one will apply one of the well known metaheuristics, whose performance has been theoretically analyzed in some cases (e.g. simulated annealing) and which are easily applied with varying success.

What metaheuristic should be chosen depends on the structure of the associated solution space. In the case of timetabling one must strictly differentiate between the so called "hard" constraints and the soft ones. The hard constraints consider physical limitations while the soft ones focus on wishes of the affected persons. The main "hard " constraints are those requesting that one object cannot be at two places at the same time. As one easily can see, a timetabling problem with only this kind of constraints is extremely symmetric. Even with the additional and unrealistic conjecture that there is only one correct partition of the lessons in slots there will be (#timeslots)! solutions which result from simply permutating the timeslots. Applying the soft constraints to a given partition will require the solution of a problem which is in many aspects similar to the general traveling salesman problem. A good coding of the above problem concentrating on the -more important- hard constraints will therefore try to operate on equivalence classes in respect to the operation of timeslot permutation. Ignoring this effect leads to bad

performance and results as shown by [11]. Especially genetic algorithms tend to fail on this kind of problems because several species are likely to belong to the same class. On the other hand one has to ignore this effect in order to pay attention to the soft constraints which often show no symmetry at all. We therefore chose tabu-search as metaheuristic because it uses two kinds of functions, the ordinary cost-function and a tabu-function. The tabu-function can be used to force the process to leave the current equivalence class by considering partitions instead of timeslots. The tabu-list contains pairs of lesson-types. It prevents types of lessons and thereby lessons which have been separated form being regrouped. Considering types instead of concrete lessons removes another symmetry created by several lessons of the same type. When moving a lesson we chose another lesson from the "source" slot at random an add the unordered pair to the tabu list. Considering ordered pairs would prevent lessons from beeing repositioned together, which seemed an unnessesary or perhaps even wrong attemt. The size of the admissible neighborhood shrinks -depending on the number of lessons with the same type - very fast when the length of the tabu-list ist growing.

3.2 A User Interface for Cooper & Kingston's Model combined with ASAPs' attribute system

In order to provide the user with maximal flexibility, we chose a subset of Cooper & Kingstons'[8] model (although it was unknown to us at this time), especially we removed the possibility to request "object consistency", e.g. to ask for a constant but unspecified room. As Carter [5] has pointed out, this would have made the room allocation subproblem $\mathcal{NP}$-hard. As in [8] the basic founding principle of the system is that of a set. A set may include objects, timeslots or other sets either as a subset or as an element. The latter

differentiation is used for output purposes only. The possible interpretation, that all members of a set as an element have to be used together if that set is used at all, would change the resource (room) -allocation problem into an instance of set packing, and make it $\mathcal{NP}$-hard again.

FIG II

This concept of a system of sets ist now used as a semantic abstraction layer in the overall construction. Operations "above" handle sets and don't see how they are build. Operations "below" build sets, but do not care what happens to them. This concept has proved advantageous for a quick and stable construction but seemed a bit clumsy considering direct relations from objects perceived by the user to the underlying constructs. In this paper this layer allows to discuss the parts "below" and "above" separately.

The set building part is very similar to that of Burke et. al. [3]. Objects can have different attributes (text, numbers, bools) and sets are formed by evaluating characteristic formulas using these attributes. Additionally sets can be formed manually.

FIG III

In order to facilitate preferences a value describing in five grades, whether the objects of these sets like each other or not, can be assigned to each pair of sets. Requesting, that the set-subset relation is acyclic, which is true for any sane construct anyway, values are propagated using the following formula:

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

($\subset$ means direct relations and not the transitive hull). Values equaling zero represent "ignore" and are therefore not accounted. In order to make the evaluation process cycle-free the sets are sorted topologically according to the set-subset relation. Then sets are handled in this order, and for each set

the relations to the already finished sets are evaluated, looking at these in topological order, too. Values given by the operator override the calculated values at every stage of the propagation.
"Unary" requests like gap reduction are propagated according to the topological structure averaging over the predecessors.
FIG IV
For instance and following the example of [3] this allows to define an attribute "wheelchair access", to use this to create a set of special rooms automatically and to specify that these rooms are preferred by the set of disabled people created in the same way. Then one could pick a single room which is especially disliked by one of the staff and reset their relation to "dislike"overriding the above.
The presentation part consists of user modificatable tables based on the set-subset relation. Any two triples of sets can be chosen to form the index of the table, allowing to create almost any view on the data. Copy-operations are adjusted according to the context of each cell, allowing the complete creation of the plan by just clicking. This is best understood by looking at the following screenshot: We have chosen a tuple for the top categories and a single set for the left ones. The clipboard shows lessons striped from timeslots and rooms, which will be added again by reinsertion into the table.
FIG V

3.3 An online Improving Assistant

The so called "new wave" in timetabling [2] supports the use of interactive features in timetabling programs. The provable intractability of the problem and the excellence of human timetablers support this idea. But with the exception of the simplest checking and moving operations partial problems

turn out to be equivalently hard, thus disabling the possibility to support the user with solutions to single aspects.

Seeing the relationship between user and program more as a partnership than as a slavery in any direction, one easily arrives at models of two persons trying to solve a problem by cooperation. They will usually state their problem, start to think, and whoever reaches a conclusion first will suggest it to the other, which may accept or not. In this context the handling of human suggestions by the program, especially the detection of clashes, is one of the most common features of timetabling programs. In the other direction either a complete plan is specified, which development has to be awaited by the user or moves which do not cause an immediate clash are presented. In both situations the time spent for thinking by the user is a complete idle time for the program and most probably for the whole system, because mostly single-user PC's are used for practical timetabling.

To exploit this time leads into the field of so called "anytime" algorithms, which can be interrupted at any time giving a better result the more time is spent waiting. This kind of algorithm is usually found in engineering environments, where calculation is interrupted, when the minimal specifications are met.

For example any local search algorithm as described by [15] could be transformed to an anytime algorithm by just outputting the best solution found so far.

It didn't seem reasonable to provide the user with a suggestion providing a sequence of steps which does not lead to a clash in a bounded number of steps. Furthermore the resulting moves often cannot be understood by the user.

We therefore chose to look for provable negative instances, that is moves

which cannot lead to a solution, in order to warn the user not to perform them. Therefore a exhaustive search algorithm was forced with some overhead to be "anytime" by performing a complete search for sequences of a successively growing depth. From the field of online-algorithms is known, that doubling the length provides optimal results, when the costs are proportional to the depth[1]. In this case it is exponential, so we chose a simple one step increment. Changes which cannot result in a solution without removing an already set lesson are marked but not prohibited, because the user might know, what he is doing.

3.4 Annotations to the Actual Implementation

We didn't implement the algorithm of Galvin because ist is still to specialized to be integrated in a general environment. The "filter"-algorithm deduced from the theorem of Hall was first implemented standalone: Performing quite poor on the data of a local high school in the beginning it's performance could be vastly improved by resorting the timeslots according to the number of clashes and restarting the algorithm using the previous output as an input. The number of clashes cold be reduced to 2% quite fast but in a very oscillating manner and without recognizable respect of the wishes.
The perhaps most important but in principle trivial insight gained by this was that it is necessary to spread the information given in the wishes over the whole set of timeslots. Suppose there are two teachers A and B, one class having one lesson with each teacher, two Timeslots TS_1 and TS_2 and all are indifferent to each other except for teacher A specially liking TS_1. Then it is not sufficient to give all relations except (A, TS_1) the same value. Any technique modifying TS_2 first will have no clue that A in a way dislikes it in respect to TS_1. Placing A here will force B to be placed in TS_2 resulting

in the need for many exchanges to get A moved. We therefore suggest to divide any value given by the user by the average of all values belonging to the corresponding category (Timeslots, Rooms, etc.).
Implementing the algorithm as a filter for tabu search in the memetic sense especially accelerated the reduction of clashes in the first 500 iterations but showed no effect later on. Tuning the parameters resulted in changes between 50% and 5% of violated constraints, but a theoretic foundation of this behaviour could not be identified.
The support of general sets and the "shealding layer" was found to be a suitable method. By defining the set-creation layer and then ignoring it the system can appear as specialized timetabling tool for any application. Labels and semantics can be chosen arbitrarily.
On the other hand it increased the complexity of every single operation by at least a factor proportional to the input size. Only a specialized treatment of special cases (sets with one ore two elements) allowed an acceptable number of several thousand iterations per hour. Furthermore knowledge given for specialized problems could not be applied, but the possibility to integrate several solution methods has allowed partially to overcome this problem.

References

[1] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993. Preliminary version in Proc. 1st Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science 318, Springer-Verlag, Berlin, 1988, 176–189. Also Tech. Report CS-87-68, University of Waterloo, Department of Computer Science, October, 1987.

[2] Victor A. Bardadym. Computer-aided school and university timetabling: the new wave. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 253–268, 1995.

[3] Edmund Burke, Kirk Jackson, Jeff Kingston, and Rupert Weare. Automated timetabling: The state of the art. http://www.asap.cs.nott.ac.uk, 1996.

[4] Edmund K. Burke, James P. Newell, and Rupert F. Weare. A memetic algorithm for university exam timetabling. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 496–503, 1995.

[5] M. W. Carter and C. A. Tovey. When is the classroom assignment problem hard? Working Paper 89-03, Department of Industrial Engineering, University of Toronto, 1989.

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

[7] Tim B. Cooper and Jeffrey H. Kingston. The complexity of timetable construction problems. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, pages 511–522, 1995.

[8] 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.

[9] D. de Werra. Restricted coloring models for timetabling. *Discrete Math.*, 165/166:161–170, 1997.

[10] Reinhardt Diestel. *Graphentheorie.* Springer, Berlin,Heidelberg,New York et al., 1996.

[11] 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.

[12] Dorit Hochbaum, editor. *Approximation algorithms for NP-hard problems.* PWS Publishing Company, Boston, 1997.

[13] Dieter Jungnickel. *Graphen, Netzwerke und Algorithmen.* BI-Wissenschaftsverlag, Mannheim, Wien, Zrich, 2 edition, 1990.

[14] Martin Löhnertz. Theorie und Praxis der automatischen Stundenplanerstellung, February 1999. Diplomarbeit Institut für Informatik III Rheinische Friedrich-Wilhelms-Universität Bonn.

[15] A. Schaerf. Tabu search techniques for large high-school timetabling problems. CWI-Report CS-R9611, Centrum voor Wiskunde en Informatica, Amsterdam, 1996.

[16] Tomaz Slivink. Short proof of galvins theorem on the list-chromatic index of a bipartite multigraph. *Combinatorics, Probability and Computing*, 5:91–94, 1996.

[17] Zsolt Tuza. Graph colorings with local constraints a survey. *Discussiones Mathematicae Graph Theory*, 17(2):161–228, 1997.

FIG I

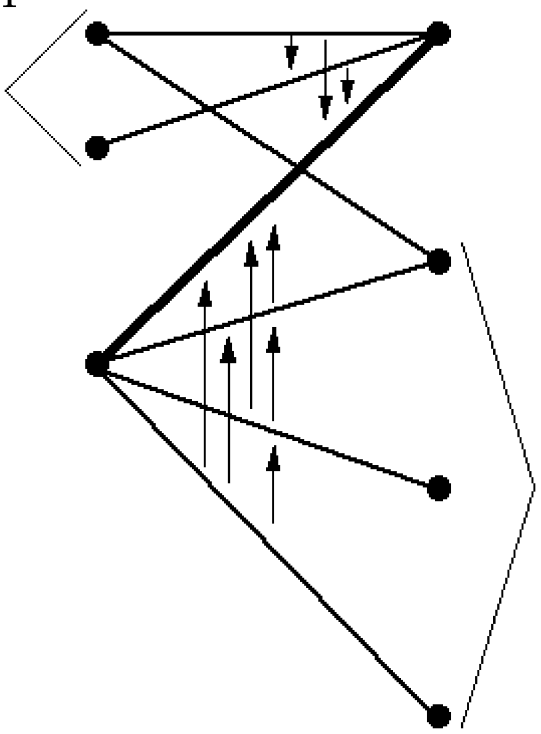


FIG II

FIG III

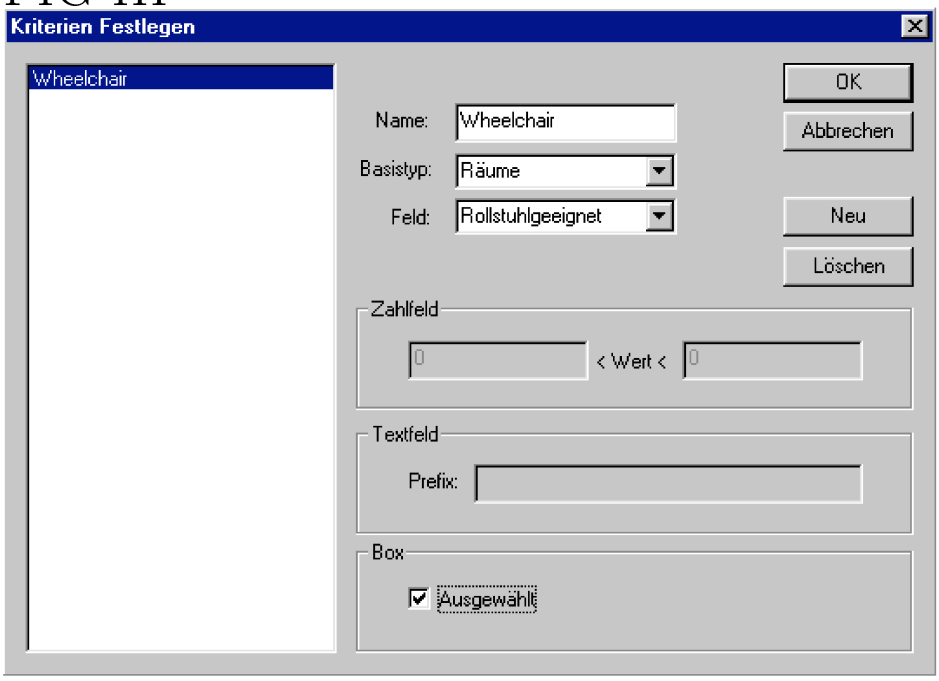

Kriterien Festlegen
Wheelchair
Name:
Wheelchair
Basistyp:
Räume
Feld:
Rollstuhlgeeignet
OK
Abbrechen
Neu
Löschen
Zahlfeld
0
< Wert <
0
Textfeld
Prefix:
Box
Ausgewähl

FIG IV

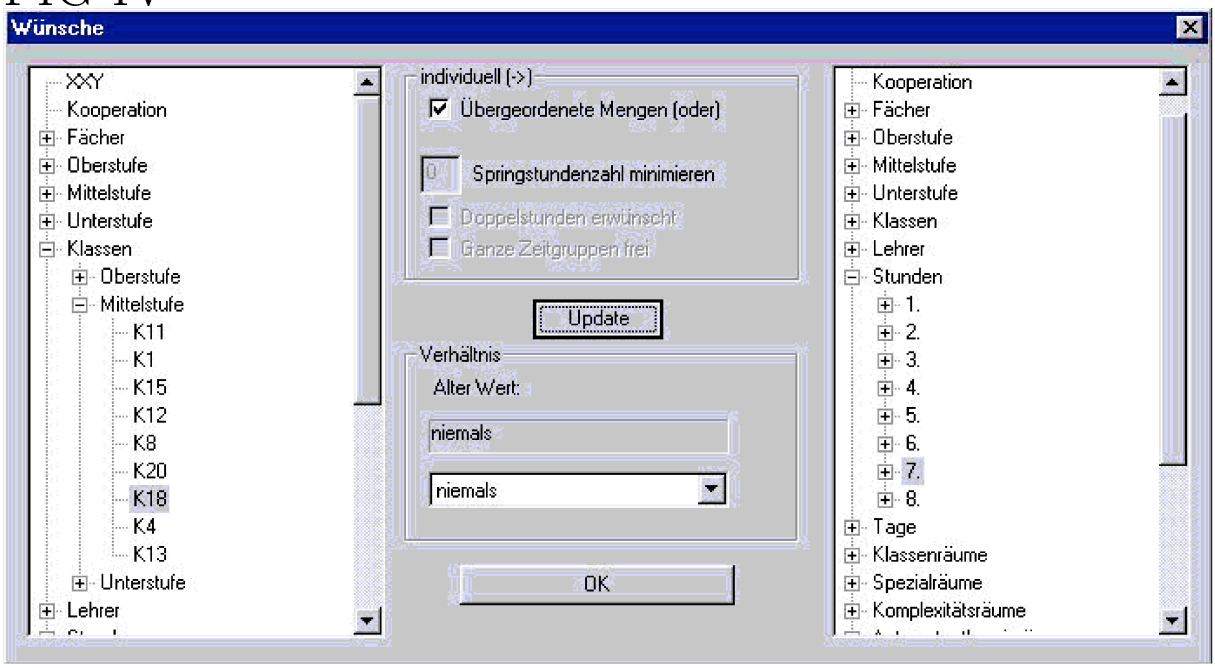
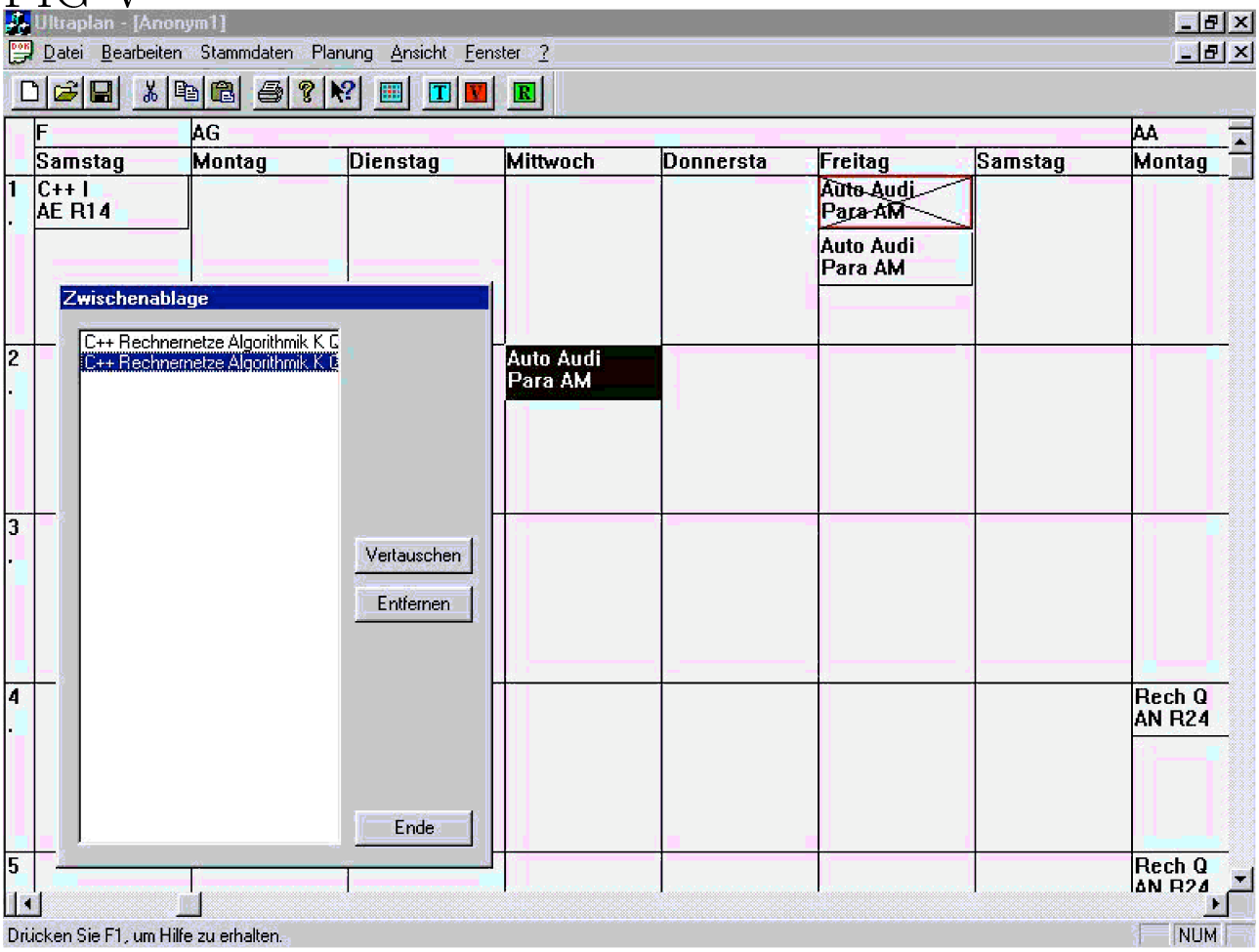

Wünsche
XXY
Kooperation
Fächer
Oberstufe
Mittelstufe
Unterstufe
Klassen
Oberstufe
Mittelstufe
K11
K1
K15
K12
K8
K20
K18
K4
K13
Unterstufe
Lehrer
individuell (->)
Übergeordenete Mengen (oder)
0
Springstundenzahl minimieren
Doppelstunden erwünscht
Ganze Zeitgruppen frei
Update
Verhältnis
Alter Wert:
niemals
niemals
OK
Kooperation
Fächer
Oberstufe
Mittelstufe
Unterstufe
Klassen
Lehrer
Stunden
1.
2.
3.
4.
5.
6.
7.
8.
Tage
Klassenräume
Spezialräume
Komplexitätsräume

FIG V


Ultraplan - [Anonym1]
Datei Bearbeiten Stammdaten Planung Ansicht Fenster ?
F
AG
AA
Samstag
Montag
Dienstag
Mittwoch
Donnersta
Freitag
Samstag
Montag
1 .
C++ I
AE R14
Auto Audi
Para AM
Auto Audi
Para AM
Zwischenablage
C++ Rechnernetze Algorithmik K G
C++ Rechnernetze Algorithmik K G
2 .
Auto Audi
Para AM
3 .
Vertauschen
Entfernen
4 .
Rech Q
AN R24
Ende
5
Rech Q
Drücken Sie F1, um Hilfe zu erhalten.
NUM