INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen

Oettingenstrasse 67 D–80538 München

Constraint-Based Grammar School Timetabling

A Case Study

Michael Marte

Diplomarbeit

Beginn der Arbeit:	14. Januar 1998
Abgabe der Arbeit:	20. Juli 1998
Aufgabensteller:	Prof. Dr. François Bry
Betreuer:	Dr. Slim Abdennadher

Ich versichere hiermit, daß ich die vorliegende Arbeit selbstständig verfaßt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 20. Juli 1998

Zusammenfassung

Das Ziel der vorliegenden Arbeit war es, unter Anwendung von Methoden der Constraintprogrammierung einen Stundenplan für das Maria-Theresia-Gymnasium, Augsburg, zu erzeugen. Gegeben waren die Klassen, die Kurse der Kollegstufe, die Kopplungen, die Einteilung der Lehrer und deren Verfügbarkeit, eine Liste der Räume mit besonderer Ausstattung und die Stundenpläne der Kollegstufe und des Sportunterrichts. Der Stundenplan sollte nicht nur physikalisch durchführbar sein, sondern darüber hinaus pädagogischen Prinzipien gehorchen und die Wünsche der Lehrer berücksichtigen. Wegen der Vielzahl zu erfüllender Bedingungen ist Schulstundenplanung eine schwierige und langwierige Expertenaufgabe. Am Maria-Theresia-Gymnasium nimmt die Stundenplanung jedes Jahr aufs Neue zwei Lehrer zwei Wochen lang in Anspruch.

Das Problem wurde zunächst in Zusammenarbeit mit dem Maria-Theresia-Gymnasium zusammengestellt und dann als ein endliches Constrainterfüllungsproblem modelliert, wobei nicht unbedingt zu erfüllende Bedingungen (sogenannte weiche Constraints) nicht berücksichtigt wurden. Gelöst wurde das resultierende Problem mit einem Suchverfahren, daß chronologisches Backtracking mit Constraintpropagierung und Heuristiken zur Variablen- und Wertauswahl kombiniert. Dieses Verfahren basiert auf dem Constraintlöser von Sicstus Prolog. Die Variablenauswahl hat das erst kürzlich vorgeschlagene Prinzip $e(n)$ zur Grundlage, das dynamisch in eines derjenigen Teilprobleme verzweigt, für die eine maximale Anzahl von Lösungen erwartet wird. Die Wertauswahl basiert auf der nur selten verwendeten Heuristik prom, die dynamisch in eines derjenigen Teilprobleme verzweigt, die die Anzahl der möglichen Wertekombinationen maximieren. Auf einem Personalcomputer benötigt das Suchverfahren etwa fünfzehn Minuten zur Lösung des Problems, wobei kein Backtracking auftritt. Der berechnete Stundenplan ist physikalisch durchführbar und zufriedenstellend in Hinblick auf die Qualität des Unterrichts, aber die Stundenpläne der Lehrer sind noch nicht akzeptabel weil deren Wünsche und weiche Qualitätsconstraints nicht berücksichtigt werden.

Contents

1 Introduction

In Germany, the task of grammar school timetabling consists in designing classes and courses, assigning teachers, developing the timetables and assigning rooms. A lot of requirements are to be met which makes timetabling a difficult and time-consuming expert task.

The Maria-Theresia-Gymnasium (MTG) at Augsburg, Germany is a grammar school with about 800 pupils, who are educated by about 60 teachers in three directions of study. Timetabling is carried out in three main steps. First, courses and classes are created. Second, teachers are assigned. Third, timetables and room assignments are developed. Development of timetables is task of two dedicated teachers; the work is performed manually and takes about two weeks. Due to changes in structure, the problem of timetabling is to be solved from scratch every year.

This work deals with the 1997 instance of the MTG timetabling problem. The task consisted in creating timetables on base of a specification including classes, courses, assignment of teachers, course phase timetables, the timetables of sports education and the requirements to be met by timetables. In total, there were 967 lessons to place. The specification was compiled in cooperation with MTG. Since there is no lack of space at MTG, assignment of rooms played a minor role.

The problem was tackled and solved in part by means of constraint programming (CP) [JM94, Wal96, JFM96, HS97, FA97]. For the timetabler CP offers a variety of methods, mainly from Artificial Intelligence, developed to solve combinatorial problems efficiently [Nad88, Pro93, Tsa93]. These methods require to declare the problem as a constraint satisfaction problem (CSP), i.e. a set of variables (representing lessons, for example), each associated with a domain of values it can take on, and a set of constraints among the variables. Constraints are relations which specify the space of solutions by forbidding combinations of values. Methods include search, heuristics and constraint propagation and there are CP tools providing some of the methods.

Systematic search typically instantiates variables sequentially following some search order. If the procedure fails to extend a partial solution, decisions are undone and alternatives explored. Systematic search often relies on heuristics which define the order in which variables and values are chosen. Orders have been found to have a substantial effect on the efficiency of search [DM94].

Constraint propagation is complementary; it simplifies a problem by pruning values which cannot participate in a solution. This way the search space is decreased and search becomes easier. Usually, constraints are propagated after each instantiation.

What does this work contribute?

- It presents a large timetabling problem from practice and the techniques used by human timetablers to solve the problem.
- It shows how to model a major part of the problem as a CSP. For this purpose, a new constraint is introduced: If X is a set of variables, $\mathsf{nval}(X,n)$ renders instantiations illegal, which use more than n values for the variables from X. Due to its generality, this abstraction might be useful in contexts other than timetabling,

too. An algorithm propagating nval constraints is proposed.

- It shows how to solve the resulting CSP efficiently by means of a search procedure integrating the CP environment of Sicstus Prolog with the heuristics $e(n)$ and prom. $e(n)$, a principle for variable selection proposed recently, branches into one of the subproblems which are expected to have a maximum number of solutions [GMP$^+$96]. prom, a general value selection heuristic, branches into one of the biggest subproblems [Gee92]. This result is compared against the performance of other, widely-used search procedures. Experience with respect to implementation and experimental evaluation is reported, conclusions are drawn.

The treatment is organized as follows. Section 2 explains the german grammar school timetabling problem. Section 3 details on timetabling at MTG and introduces the 1997 instance of the problem. Section 4 introduces the approach of constraint programming. Section 5 shows how to apply this approach to grammar school timetabling, section 6 talks about related work and section 7 summarizes and looks out for future work. Appendix A introduces notations. Appendices B, C and D detail on modelling and computing search control information. Appendix E gives some parts of the data set which timetabling was based on.

Acknowledgements

Thanks go out to Mats Carlsson and Christian Holzbaur, who contributed with their fast and accurate technical support; to Patrick Prosser for the discussions on $e(n)$ and to Claus Schmalhofer, teacher and timetabler at MTG, who initialized the project, explained the problem and assembled all the data.

2 The German Grammar School Timetabling Problem

2.1 Structure and Characteristics of Education at German Grammar Schools

In Germany the aim of visiting grammar school is to aquire college graduation. The german grammar school comprises nine grades, which are numbered from five to thirteen. (Grades one to four are the grades of elementary school.) Important features of this system are:

- Sexes are segregated in sports education.
- There is religious education for catholics and protestants. As an alternative, there is ethical education. One of the subjects has to be chosen.
- English is the first foreign language teached.
- For the grades seven up to and including eleven a second foreign language has to be chosen, either latin or french.
- For the grades eight up to and including eleven a *direction of study* (a *branch*) has to be chosen. There are three directions: *mathematics and natural sciences*, *social sciences* and *modern languages*. Latin as second foreign language is condition for modern languages. The number of lessons per week depends on the direction of study chosen.
- Throughout the grades five up to and including eleven pupils are grouped to form classes. For the last two grades pupils must choose a set of courses resulting in a more college-like, course-structured education, which will be referred to as the *course phase*. Base courses are distinguished from major courses. Two major courses must be selected.
- The number of lessons per week ranges between 28 and 34. Throughout the grades five up to and including eleven education mainly takes place in the morning. In general, six lessons are given a day. In the grades twelve and thirteen education may also take place in the afternoon. A lesson lasts 45 minutes. *Morning periods* are numbered from one to six. *Afternoon periods* are numbered from seven to twelve.

See the curriculum in appendix E for detailed information on subjects and number of lessons.

An *offer* comprises all lessons of a course or all lessons of some subject of some class.

A *free period* is a period a teacher has not been assigned some class or course and which he has to spend at school because there are periods before and afterwards he has been assigned. This notion also applies to pupils.

2.2 Tasks of Timetabling

Solving the german grammar school timetabling problem involves several tasks.

- Classes must be formed for grades five up to and including eleven.
- Courses must be formed for grades twelve and thirteen such that each pupil can visit the courses he or she selected.
- An *assignment matrix* must be created. This matrix assigns a teacher to each course and it assigns a teacher to each subject of each class. Labour law must be taken into account.
- A *timetable* must be created, which assigns an appropriate period to each lesson. Several constraints are to be considered (cf. 2.4). For example, teachers must be available and some subjects must be teached in dedicated facilities.

2.3 Techniques of Modelling

2.3.1 Couplings

The process of forming courses and classes also yields a set of couplings. A *coupling* is a modelling technique describing a set of offers which must take place the same time. Only offers comprising the same number of lessons may be coupled. There are three reasons for introducing couplings.

- If there are boys and girls in a class, the class must be split up for sports education. It is a common wish for these lessons to take place the same time and a coupling can be used to express this wish. If a class has pupils from different confessions or branches or some pupils have latin as second foreign language while others have french, splitting is usually required, too (cf. 2.3.2).
- Often groups emerged from splitting are small. It is a common wish to merge groups from different classes to form a new class for a certain subject. To enable merging for a certain subject, the lessons of this subject must run in parallel in all classes concerned, which can be expressed by a coupling. As a result of merging, sports education may run in parallel in all classes of a grade.
- Pupils enjoy some freedom in composing a set of subjects for the course phase. This implies that some pupils might have to wait while others attend their courses. In order to keep the number of free periods small, it is common to offer courses in parallel. This simultaneity can be modelled by means of couplings.

The placement of a lesson which is member of a coupling may be quite difficult because a place must be found which satisfies the constraints of all lessons of the coupling and of all teachers involved. Couplings therefore impose strong restrictions on timetables and it is usually a good idea to minimize the number of couplings in order to make timetabling easier. This goal can be reached by making classes as homogenous as possible. Unfortunately, extensive savings in public expenditure have lead to an increasement in class sizes, which makes more couplings necessary, especially in higher grades.

2.3.2 Mixed Classes & Virtual Classes

Before the ninth grade pupils have to decide for a direction of study for the years following. Since the number of pupils in different branches often varies considerably, *mixed classes* with pupils from two or three branches often cannot be avoided. Since branches differ partly in composition of subjects and material, mixed classes must be split up for branch specific education. Often merging of pupils of the same grade and branch is required, which in turn requires simultaneity of branch specific education of different classes of a grade.

To faciliate the expression of the simultaneity required, mixed classes are represented by sets of virtual classes. A *virtual class* is a set comprising all pupils of a class that are members of the same branch. With virtual classes simultaneity of branch specific education can be modelled by coupling branch specific subjects of virtual classes.

Virtual classes introduce a new problem: simultaneity of education of *common subjects* of a class, i.e. subjects that are not specific to any branch of the class, must be ensured. This can be done by coupling common subjects among virtual classes.

2.4 General Requirements of Timetables

Resource constraints make sure that rooms, equipment and teachers are available. In other words, resource constraints ensure physical feasibility. Reasons for unavailability of resources are mainly due to the competition of lessons for resources. Furthermore resources may be shared among different schools. In this case the times of unavailability are likely to be known in advance.

Quality constraints make sure that the timetable is good or acceptable. Since preferences and priorities depend highly on school and grade, a precise definition of a good or acceptable timetable cannot be given in this broad context. In general a timetable will be considered good if it is in line with pedagogical principles, which improve the efficiency of education, and if it supports the working climate.

Most resource constraints are *hard constraints*, i.e. constraints that must be satisfied. In contrast, most quality constraints are *soft constraints*, i.e. constraints that may be violated but should be fulfilled.

3 Timetabling at Maria-Theresia-Gymnasium

The *Maria-Theresia-Gymnasium* (MTG) at Augsburg, Germany is a grammar school with about 800 pupils, who are educated by about 60 teachers in all directions of study.

3.1 The Process of Timetabling

After courses, classes, couplings and the assignment matrix have been designed, the timetables are created. The timetables of sports education are created first, then the course timetables are created. Based on these timetables the class timetables are developed.

The new timetable cannot be developed on base of the previous year's timetable because too much changes from year to year: the number of classes of the fifth grade varies; the number of teachers decreases due to savings in public expenditure; composition of classes changes at the transition from the eighth to the ninth grade due to branch selection; pupils join and leave.

The divide & conquer strategy used for timetabling is vital for solving the problem; without division the problem could not be handled by human timetablers. Since many design decision in early stages of timetabling are quite arbitrary, the divide & conquer strategy restricts the space of solutions.

Timetables are developed by two dedicated teachers. Since quite a number of requirements have to be met (see below), timetabling is a difficult and time-consuming expert task. At MTG it takes about four man weeks.

3.2 Requirements of Timetables

The following list details the requirements timetables at MTG have to comply with.

- *Non-overlap constraints*:
  - Except for couplings, lessons of a teacher must not overlap.
  - Except for couplings, lessons of a class must not overlap.
- *In-parallel constraints*: Coupled lessons must run in parallel.
- *Facility constraints*:
  - Chemistry, physics, music, arts and sports must be teached in dedicated facilities. Biology should be teached in rooms with appropriate equipment.
  - For each lesson a room of sufficient size must be available.
- *Availability constraints*: Sports facilities and teachers are shared among several schools. Some teachers have a long way to work and are not available for the first period. Times of availability must be considered.
- *Teacher constraints* support the quality of teacher timetables.

  - *One-day-off rule*: Teachers with less than seventeen lessons to teach a week must have one day off a week.
  - *Five-lesson rule*: Teachers must not be forced to give more than five lessons a day.
  - *Free-periods rule*: A teacher's timetable should not contain more than five free periods. It should not contain less than three free periods. It should not contain a day with more than two free periods.

    An even distribution of free periods is quite important. Teachers having a free period may be instructed to replace a missing colleague. This additional work is not paid. Since teachers with many free periods are more likely to be instructed to replace a missing colleague than teachers with few free periods, an uneven distribution of free periods will cause iniquities and thus deteriorate the working climate.
  - To support the working climate teachers' wishes should be considered.

- *Education constraints* support the quality and efficiency of education.
  - For sports education double lessons are required.
  - If arts is a two- or three-lesson subject, a double lesson is appreciated.
  - *AM constraints*: The lessons of classes must not take place in the afternoon. Exceptions from this rule may be specified by the given timetables.
  - *Monday-friday rule*: If one lesson of a two-lesson offer takes place on Monday, the other lesson should not take place on Friday.
  - *Period constraints*:
    * The lesson of a one-lesson offer should not take place in the sixth period.
    * Not both lessons of a two-lesson offer should take place after the fourth period.
    * If one lesson of a three-lesson offer takes place in a sixth period, the other lessons should take place before the sixth period.
    * If one lesson of a four-lesson offer takes place in the sixth period, the other lessons should take place before the sixth period. If one lesson takes place in the sixth period and another lesson takes place in the fifth period, the other lessons should take place before the fifth period.
  - *Distinct-days constraints*:
    * The lessons of a two-lesson offer must not take place the same day, except for sports, arts and handwork and for reasons of unavailability of teachers.
    * The lessons of a three-lesson offer must not take place the same day, except for reasons of unavailability of teachers.
    * The lessons of a four-lesson offer should not take place the same day.
    * For grades five, six and seven the lessons of mathematics must take place on different days.
  - *Interleaving constraints*: The lessons of a two- or three-lesson offer should not take place on successive days.
  - *Anti-teasing constraints* avoid the accumulation of brain-teasing lessons.

        * For each class of the grades five, six and seven, at most two lessons of arts, music, sports and handwork may take place the same day.
        * Except for reasons of unavailability of teachers, at most two lessons of an offer may take place the same day. If two lessons of an offer take place the same day, these lessons should not take place one after another.

- *Examination constraints* support the realisation of examinations.
    - Often a teacher teaches the same subject in two classes of a grade. In this case two of the lessons, one from each class, should take place one after another. This allows for carrying out the same examination in both classes thus enabling juster marking and avoiding the work of creating a second set of examination problems.
    - For five- and six-lesson offers of the grades ten and eleven a double lesson for examinations would be appreciated.

3.3 The Constraint Hierarchy

Soft constraints are not of equal importance. Teachers' wishes are least important and if it helps to satisfy some other soft constraint, teachers' wishes can be neglected altogether. With respect to the remaining soft constraints, the quality of education is most important; obeying the free-periods rule is more important than the support of examinations; soft facility constraints are least important. However, if obeying some soft education constraint forces to violate the free-periods rule in several cases, it is permissible to neglect the offending soft education constraint. In this sense, each soft constraint, except for teachers' wishes, can be thought of as associated with the cost of its violation. These facts lead to the following constraint hierarchy. The first level contains the hard constraints; the second level contains the soft constraints except for teachers' wishes, which constitute the third level.

1. Non-overlap constraints, availability constraints, hard facility constraints, in-parallel constraints, hard teacher constraints, hard education constraints
2. Education constraints, free-periods rule, examination constraints, soft facility constraints
3. Teacher's wishes

3.4 Techniques of Timetabling

The method of timetabling employed at MTG can be described as a loop, where each iteration selects and places a lesson. Placing a lesson involves finding a day and a period such that no hard and not too many soft constraints will be violated by placing the lesson. In case no good place is available for the lesson under consideration, a conflict resolution method is used to free a good place by moving and interchanging lessons already placed. Both lesson selection and conflict resolution are supported by problem specific guidelines. Finally, if there is time left, the timetable is optimized by means of conflict resolution.

The aim of *conflict resolution* is to yield a good place for the lesson which cannot be placed without violating constraints. A conflict is resolved by local repair. The principle of *local repair* consists in moving and interchanging lessons until a good place is available. Consider the following problem: One lesson of English is left to place, but the teacher is not available for any of the free lessons. Local repair looks for a good place and dispels the lesson placed there. If there is a good unused place for the lesson dispelled, then the lessons are placed and local repair finishes. If there is no good unused place for the lesson dispelled, local repair iterates in order to place the lesson dispelled. As experience shows, human timetablers can cope with up to ten iterations.

Not only the timetable is subject to change in the course of local repair; the assignment matrix may be changed, too. But since consequences of changing the assignment matrix may be difficult to track and to cope with, such changes come into question only in early stages of timetabling. Furthermore, the headmaster, who is the person in charge, must be consulted before changing the assignment matrix.

Neither the next lesson to place, nor the next lesson to dispel is chosen arbitrarily because lessons are not equal with respect to number and rigour of constraints. Since the satisfaction of constraints gets harder and harder with the number of free places decreasing, it is clever to place more constrained lessons first. This strategy diminishes the number of conflicts to resolve in the further course of timetabling. In local repair, if the lesson to dispel can be chosen from a set of candidates, it has proved advantageous to dispel the least constrained lesson. In general, this lesson can be placed easier than the more constrained lessons. This strategy diminishes the average number of iterations necessary to resolve a conflict.

Selecting the next lesson to place is supported by a generic lesson selection scheme, which has been established over the years. The next lesson is selected according to a hierarchy. Each level of the hierarchy defines a set of lessons which are considered equivalent in terms of constrainedness. The levels are ordered: the higher the level the more constrained the lessons. According to the strategy introduced above, timetabling starts with placing lessons from the highest level. After placing the most constrained lessons, timetabling continues with the level one lower, and so on. The following list gives the hierarchy's levels in decreasing order. Coupled lessons are given priority because they are harder to place (cf. 2.3.1).

1. Lessons of the course phase.

   Up to four courses are coupled.

2. Sports lessons.

   Couplings among all classes of a grade occur frequently. Often three teachers are involved. Furthermore double lessons are required and sports facilities are available only for a given, very limited number of periods.

3. Lessons of religious and ethical education which are members of couplings.

   Again, couplings among all classes of a grade occur frequently. Usually three or four teachers are involved. Since there are no facility constraints, lessons may be placed anytime. Therefore lessons of religious and ethical education are easier to place than sports lessons.

4. Latin and french lessons which are members of couplings.

   Couplings among all classes of a grade occur frequently. Often three teachers are involved.

5. Lessons of branch specifc subjects of the grades nine, ten and eleven which are members of couplings.

   Usually two classes are involved.

6. Handwork lessons of the fifth grade.

   Double lessons are required. Since handwork is an optional subject, lessons must take place in the first and second or in the fifth and sixth period. Only one teacher is available. This teacher has one day off. All this makes handwork lessons a bit difficult to place.

7. Physics, chemistry, biology, arts and music lessons.

   Rigid facility constraints are to be considered. For arts double lessons are required.

8. Lessons of languages and mathematics.

   In higher grades double lessons for examinations are required.

9. The remaining lessons.

In general, when placing a level's lessons, the lessons of classes, which only have a few lessons left to place, and the lessons of teachers, who only have a few lessons left to place, are preferred. Again, this strategy avoids expensive conflict resolutions.

3.5 The Timetabling Problem of 1997

This work deals with the MTG timetabling problem of 1997. Appendix E gives the assignment matrix. There are 26 classes, 72 courses and 57 teachers. For the grades five up to and including eleven there are 967 lessons to place; 47 couplings comprise 414 lessons from 190 offers. There are two rooms with chemistry, physics, biology, arts and music equipment, respectively. Each of the grades nine, ten and eleven has a mixed class, i.e. there are two virtual classes in each of these grades.

In 1997 room capacities do not need consideration, since there are enough rooms of sufficient size. However, for chemistry, physics and biology the availability of an appropriate room must/ should be ensured. This is not necessary for arts: since the number of teachers allowed to teach arts equals the number of appropriate rooms, it is not possible to place more arts lessons at a time than rooms available. The same is true for music.

For sports teachers the five-lesson rule and the free-periods rule do not apply; sports teachers like to keep their timetables compact and therefore do not mind to teach six or more lessons in succession.

4 Modelling and Solving Combinatorial Problems

4.1 Constraint Programming

A variety of methods have been used for solving timetabling problems [Sch95]. This work follows the approach of constraint programming (CP) [JM94, Wal96, JFM96, HS97, FA97].

Tackling some combinatorial problem by means of CP involves to model the problem as a *constraint satisfaction problem* (CSP). A CSP is a set of variables, each associated with a domain of values it can take on, and a set of constraints among variables. Constraints are relations which specify the space of solutions by forbidding combinations of values. Often a CSP is viewed as a graph (with variables as nodes and constraints as arcs) and graph-related concepts are referred to.

The CSP paradigm is of particular interest, because many problems from Artificial Intelligence and other fields of computer science can be viewed as CSP including problems from design (of machines, circuits and experiments), resource allocation in manufacturing, transportation and education (scheduling of workers, machines and teachers; assignment of rooms), molecular biology, decision support and temporal reasoning [JM94, Wal96].

In order to account for the demands of applications, the CSP paradigm has been generalized and specialized in many ways. For example, related frameworks allow for preferences among constraints (cf. 4.6) [Jam96] or restrict the domain of discourse [JM94]. A framework particularly suitable for timetabling is the finite CSP framework [Mac77].

A *finite CSP* is a triple (X,D,C) where X is a finite set of variables, C is a finite set of constraints among variables from X and D associates each variable from X with a finite and discrete domain. In this context, an *instantiation* is a function mapping variables from X to values from their respective domains as given by D. An instantiation is said to be *legal* or *locally consistent* iff it does not violate any of the constraints among the variables instantiated. It is called a *k-instantiation* iff its domain contains k variables. It is called a *partial solution* iff it is a legal k-instantiation with $k < |X|$ and a *solution* iff it is a legal $|X|$-instantiation. (X,D,C) is called *satisfiable* iff there is a legal $|X|$-instantiation.

To see how the finite CSP framework applies to timetabling, consider the following problem. Three lessons have to be placed within the periods 1, 2 and 3 such that the lessons do not overlap. Associating each lesson with a variable leads to the finite CSP $(\{x,y,z\},D,\{x \neq y, x \neq z, y \neq z\})$ with $D_x = D_y = D_z = \{1,2,3\}$. In structure, this problem is identical with the map-colouring problem which consists in assigning different colours to adjacent regions of a map. Since colouring a map is known to be NP-complete, an algorithm able to decide any finite CSP within polynomial time is not likely to exist. Research has focused on identifying restricted subclasses which can be solved efficiently by means of dedicated algorithms [Nad88, Kum92, Pro93, Tsa93]. Methods include systematic search and constraint propagation (see below), which are complementary.

4.2 Generate & Test

The brute force approach to a finite CSP $(\{x_1,\ldots,x_n\},D,C)$ consists in enumerating and testing the elements of $D_{x_1} \times \ldots \times D_{x_n}$ in order to find a solution. The size of the search space expanded by this *generate & test* method is $|D_{x_1}| \cdot \ldots \cdot |D_{x_n}|$ in the worst case, i.e. exponential in n.

4.3 Chronological Backtracking

A more efficient method is *chronological backtracking* [BV92]. Chronological Backtracking is a depth-first tree-search algorithm which instantiates variables sequentially following some search order. If a partial solution cannot be extended, chronological backtracking undos the last decision and explores an alternative. If there is no more alternative, another decision is undone, and so on. In effect, it returns to the most recent node of the search tree which has a subtree left to expand. This way chronological backtracking identifies and eliminates a subspace of the search space which does not contain a solution. This explains the efficiency gain of backtracking over generate & test.

Nevertheless, chronological backtracking can perform quite poor because it does not learn from failures. This point is illustrated by the following example from [BV92]. Consider the finite CSP $(\{x_1,x_2,\ldots,x_k,\ldots,x_n\},D,\{x_1 \neq x_2, x_1 \neq x_k, x_2 \neq x_k\})$ with $D_{x_1} = D_{x_2} = \{1,2,3\}$ and $D_{x_k} = \{1,2\}$. Suppose variables are instantiated in the order $x_1,x_2,\ldots,x_k,\ldots x_n$ and cronological backtracking starts with $x_1 = 1$ and $x_2 = 2$. After extending this partial instantiation for $x_3,\ldots,x_{k-1}$, the algorithm fails to find a value for x_k which does not violate the constraints $x_1 \neq x_k$ and $x_2 \neq x_k$. Obviously, the partial instantiation for $x_1,\ldots,x_{k-1}$ cannot be extended further because of the values chosen for x_1 and x_2, but chronological backtracking has to search the subproblem $x_3,\ldots,x_{k-1}$ completely before the value 3 for x_2 can be given a try. This behaviour is known as *thrashing*. [Mac92] describes thrashing as 'the repeated exploration of subtrees of the backtrack search tree that differ only in inessential features, such as the assignment to variables irrelevant to the failure of the subtrees'.

Thrashing has been addressed from two directions. Intelligent backtracking tries to jump back as far as possible or to learn from failures. The other approach embeds constraint propagation into search.

4.4 Constraint Propagation

The efficiency of search can be improved by means of constraint propagation. *Constraint propagation* simplifies a problem by pruning values which cannot participate in a solution. In any case, search becomes easier since pruning of values decreases the search space. The terms *establishing consistency*, *maintaining consistency* and *enforcing consistency* are often used in place of constraint propagation. The level of consistency a search procedure enforces is referred to as the *pruning* or *propagation performance* of the procedure.

A finite CSP (X,D,C) is called 1-consistent iff each value in each domain is legal. It is called *k-consistent* $(1 < k < |X|)$ iff, given any legal $(k-1)$-instantiation, this instantiation can be extended to include any kth variable such that a legal k-instantiation results. It is called *strongly k-consistent* iff it is i-consistent for all $1 \leq i \leq k$. It is called *globally consistent* iff it is strongly $|X|$-consistent. *Node*, *arc* and *path consistency* refer to 1-,2- and 3-consistency, respectively.

For example, consider the finite CSP $(\{x\}, D, \{x \neq 1\})$ with $D_x = \{1,2\})$. This CSP is not node-consistent because the value 1 from D_x conflicts with the constraint $x \neq 1$. For another example, consider the finite CSP $(\{x,y,z\}, D, \{x \neq y, x \neq z, y \neq z\})$ with $D_x = \{1,2\}$ and $D_y = D_z = \{2,3\}$ [Kum92]. Now choose x and some value a from D_x. In any case, there are values $b \in D_y$ and $c \in D_z$ such that $a \neq b$ and $a \neq c$. Repeating this check for y and z leads to the conclusion that this CSP is arc-consistent. Since it is node-consistent, too, it is strongly arc-consistent.

To see how chronological backtracking can benefit from constraint propagation, remember the thrashing example from above. The cause of the thrashing observed is a lack of arc consistency; there is no legal value for x_k if $x_1 = 1$ and $x_2 = 2$. Enforcing arc consistency after instantiating x_1 with 1 would prune 1 from the domains of x_2 and x_3 and, in consequence, the value 2 from the domain of x_2. There would be no need for further search.

Clearly, if a finite CSP is globally consistent, it is satisfiable and a solution can be assembled in a backtrack-free manner. In constrast, lower levels of consistency do not, in general, entail satisfiability and thus search is necessary to decide the problem. For example, the finite CSP $(\{x,y,z\}, D, \{x \neq y, x \neq z, y \neq z\})$ with $D_x = D_y = D_z = \{1,2\}$ [Kum92] is arc-consistent but there is no legal 3-instantiation because three variables cannot be given different values if only two values are available.

If a finite CSP has n variables, the complexity of enforcing k-consistency is $O(n^k)$ [Sei83]. Thus, enforcing global consistency has complexity $O(n^n)$. Since this approach is usually more expensive than chronological backtracking [Kum92], hybrid procedures enforcing a low level of consistency after each instantiation [Nad88, Pro93, Tsa93] are preferred in practice. Note that hybrid procedures may suffer from thrashing, too.

Which level of consistency to enforce is subject to an ongoing discussion. For the moment, it seems that enforcing arc consistency along with chronological backtracking is the most efficient procedure for solving the harder problems while the easier problems are best tackled by forward checking in combination with intelligent backtracking [SF94, BR96]. In contrast to enforcing arc consistency, *forward checking* [Pro93] does not revise the complete CSP but limits its actions to the direct neighbourhood of the variable instantiated. As soon as a variable becomes instantiated, the domains of the neighbourhood's variables are pruned but results of pruning are not propagated to the neighbourhood's neighbourhood. Clearly, the level of consistency enforced by forward checking is lower than arc consistency.

4.5 Heuristics

Tree search relies on orders in choosing variables and values. Orders have been found to have a substantial effect on the efficiency of search [DM94]. A variety of heuristics defining orders have been proposed and examined in literature. Value selection has not attracted as much attention as variable selection [BR96]. Heuristics for variable selection often recommend to prefer variables which are most constrained [GMPW96]; this way search branches into one of the least constrained subproblems. Heuristics differ in how constrainedness is measured.

There are static and dynamic orders. *Static orders* are computed before the search by analyzing the problem. *Dynamic orders* are computed, by analyzing the current state of computation, before choosing the next variable or value. Of course, the merits of an order depends on the problem it is applied to and on the search procedure it is embedded in. However, in general, dynamic orders have been found superior to static orders [DM94]. With a dynamic order the instantiation order can vary from branch to branch of the search tree.

The *minimum remaining values heuristic* (mrv) dynamically orders variables by increasing cardinality of domains, i.e. mrv proposes to select one of the variables with the smallest domains with respect to the current state of computation [HE80]. In effect, mrv branches into one of the biggest subproblems [GMP+96]. (The *size of a problem* (X,D,C) is the cardinality of the domains' scalar product.)

Another approach to constrainedness proposed recently is based on the notion of solution density. The *solution density* of a CSP with n variables is the fraction of n-instantiations which are expected to be solutions. The heuristic ρ branches into one of the subproblems which are expected to have a maximum solution density. Based on ρ, the heuristic $e(n)$ branches into one of the subproblems which are expected to have a maximum number of solutions [GMP+96].

Measuring solution density is based on the notion of looseness. The *looseness* $L(c)$ of a constraint c is the fraction of instantiations which satisfy c. If $L(c) = 0$, c is not satisfiable; if $L(c) = 1$, c cannot be violated. With looseness at hand, [GMP+96] approximates the solution density of some CSP (X,D,C) by

$$\prod_{c \in C} L(C).$$

Let $C_x \subset C$ be the set of constraints x occurs in. ρ selects the variable $x \in X$ minimizing

$$\prod_{c \in C_x} L(C).$$

Thus, the solution density

$$\prod_{c \in C - C_x} L(c)$$

of the remaining subproblem is maximized.

The expected number of solution is approximated by

$$\prod_{x \in X} |D_x| \cdot \prod_{c \in C} L(c).$$

$e(n)$ selects the variable $x \in X$ minimizing

$$|D_x| \cdot \prod_{c \in C_x} L(c).$$

Thus, the expected number of solutions

$$\prod_{\substack{x' \in X \\ x' \neq x}} |D_x| \cdot \prod_{C - C_x} L(c)$$

of the remaining subproblem is maximized.

The promise heuristic (prom) dynamically orders values [Gee92]. Suppose (X, D, C) is the current state of computation and the variable $x \in X$ has been selected. Then prom selects the value $a \in D_x$ maximizing

$$\prod_{x \in X} |D'_x|$$

where D' results from constraint propagation after x has been instantiated to a. Thus, similiar to mrv, prom branches into one of the biggest subproblems.

Finally, enum statically orders values increasingly.

4.6 Soft Constraints

Hard constraints must not be violated, while soft constraints should hold but may be violated. With soft constraints not some solution but a good or the best solution is the aim of search. In other words, adding soft constraints turns a decision problem into an optimization problem.

A popular approach consists in using a branch & bound search instead of chronological backtracking [HW95, FHS95]. Branch &bound starts out from a solution and requires the next solution to be better. Quality is measured by a suitable cost function. With this approach, however, soft constraints are only part of the cost function but play no role in selecting variables and values. Another approach is to adopt techniques developed to propagate hard constraints; soft constraint propagation is intended to associate values with an estimate of how selecting a value will influence solution quality, i.e. which value is known (or expected) to violate soft constraints, or the other way round, which value is known (or expected to) satisfy soft constraints [Mey97, AM98b]. By considering estimates in value selection, one hopes that the first solution will satisfy a lot of soft constraints.

Several frameworks have been introduced which faciliate the formal treatment of soft constraints. For example, HCLP allows for constraint hierarchies (a constraint on some level is more important than an any set of constraints from lower levels but constraints of the same level are equally important) [BFBW92] while PCSP associates each constraints with the cost of its violation [FW92]; HCSP subsumes both HCLP and PCSP (and other frameworks) and thus allows for constraint hierarchies where preferences among constraints of a hierarchy level can be defined [Mey96]. Further subsuming frameworks are described in [BMR95] and [SFV95].

4.7 Constraint Programming Tools

If a problem is to be solved, which can be cast as a finite CSP, two ways to profit from the concepts and methods of CP are prevailing.

1. Some search procedure embedding some kind of consistency enforcement is adopted from literature.
2. A CP system is used which embeds suitable techniques like search and constraint propagation into some kind of programming environment.

Since solving problems, which can be cast as a finite CSP, is an important application of CP systems, CP systems usually support the idea of finite CSPs by featuring a so-called *finite domain (FD) constraint system*. In a FD constraint system, FD variables are associated with finite and discrete domains and FD constraints may be used to constrain FD variables. Usually, the set of admissible FD constraints includes linear equations, inequations and disequations and specialized constraints tailored to the needs of timetabling and scheduling applications. The following list includes the FD constraints this work refers to.

- true is always satisfied.
- false cannot be satisfied.
- $\mathsf{in}(x,L)$ specifies that x must take its value from the finite set L of natural numbers.
- $\mathsf{neq}(x,y)$ specifies that x and y must take different values.
- $\mathsf{lt}(x,y)$ specifies that the value chosen for x must be lower than the value chosen for y.
- The $\mathsf{cumulative}$ constraint is applicable whenever tasks compete for a limited resource [AB92]. Possible resources include machince capacity, room capacity, energy, water, man power, and so on. Fields of application include scheduling [AB92] and timetabling [BDP96]. $\mathsf{cumulative}(X,d,r,n)$ specifies that at any instant i, the resource allocation by the tasks overlapping i must not exceed the limit n. $d : X \to \mathcal{N}$ and $r : X \to \mathcal{N}$ specify the duration and resource consumption of each task, respectively, and n is the amount of resource available. A $\mathsf{cumulative}$ constraint can only monitor a single resource, the resource is expected to be available entirely all the time and tasks are expected to have a constant resource consumption as specified by r.[1]
- $\mathsf{allDistinct}$ is a special case of $\mathsf{cumulative}$. $\mathsf{allDistinct}(X)$ translates to $\mathsf{cumulative}(X,d,r,1)$ with $d(x) = r(x) = 1$ for all $x \in X$. In words, the tasks, all of which are assumed to last for one unit of time and to consume one unit of the resource to control, must not overlap. More succintly, no two variables from X may get assigned the same value.

[1] This description diverts from literature. For example, [AB92] associate each task with three variables. The first one is for the task's begin, the seond one for its duration and the third one for its resource consumption. This faciliates the modelling of problems where only task surfaces, i.e. the product of duration by resource consumption, are known. However, this feature is not necessary for modelling the problem under study.

- nval is a new abstraction motivated by the needs of the problem under study (cf. 5.2.4 and B.11). $\mathsf{nval}(X,n)$ renders instantiations illegal, which use more than n values for the variables from X.

- $\mathsf{atmost}(X,i,n)$ specifies that at most n variables from X may take the value i.

- $\bigwedge C$, where C is a set of FD constraints, specifies that all constraints from C have to be satisfied simultaneously. $\bigwedge C$ is called a *conjunctive constraint*.

Constraints like cumulative, allDistinct, nval atmost and conjunctive constraints are sometimes called *global constraints*.

Constraints are propagated by a so-called constraint solver. Typically, a FD constraint solver is composed of several dedicated constraint solvers which access a common *constraint store*, communicate through variables and are scheduled by the core of the FD constraint system. This way semantics of global constraints can be exploited in algorithm design, possibly resulting in effective and efficient methods (cf. [Rég94], [CL96], [COC97] and appendix C). Furthermore, the modular architecture allows for embedding application-specific constraint solvers.

The list of well known CP systems featuring a FD constraint system includes (among others) ECL^{i}PS^e [ECR95], Sicstus Prolog [Int98], Oz [Smo95] and ILOG Solver [ILO98]. ECL^{i}PS^e and Sicstus Prolog both enhance Prolog [SS94] with constraint solving. Oz is a concurrent multiparadigm language providing for functional, object-oriented and constraint programming. ILOG Solver is a commercial C++ library providing CP techniques.

This work reported here has been performed using Sicstus Prolog. Combining Prolog's backtracking with a FD constaint solver yields a search procedure integrating chronological backtracking with constraint propagation. As with the other systems mentioned, soft constraints are not supported.

5 Creating a Timetable

5.1 Course of Work

The MTG timetabling problem can be cast as a HCSP [Mey97]. On the one hand, HCSP supports a hierarchical organiziaton of constraints, whereas a constraint on some level is more important than any set of constraints from lower levels. On the other hand, HCSP allows for preferences among the constraints of a level. Unfortunately, available constraint solvers do not support HCSP or some other kind of framework for dealing with soft constraints. Thus, for solving the resulting HCSP, it was projected to develop and implement specialized consistency techniques capable of processing hard and soft constraints. Soft constraint propagation was intended to compute recommendations controlling value selection in search. A preceding, encouraging project engaged in university timetabling [AM98b, AM98a] showed that this approach may result in a first solution satisfying a lot of soft constraints (cf. 6.1).

In fact, by following the method proposed in [AM98b], implementing a simple HCSP solver, tailored to the problem's demands, was a matter of a few days. This solver maintained arc consistency for binary constraints and propagated global constraints in the manner of forward checking. However, a procedure embedding the solver along with chronological backtracking and different variable selection heuristics did not come up with a solution even after days of search. Instead thrashing occured. Studying work on the merits of maintaining arc consistency [SF94, BR96, SF97] led to the conclusion that search is misguided because forward checking does not eliminate enough values which are inconsistent with the partial solution computed so far. Further, literature on solving global constraints made clear that it is not easy to do better [Rég94, CL96]. Therefore, the approach was given up, the more powerful FD constraint system of Sicstus Prolog was resorted to and subsequent work concentrated on finding a solution satisfying the hard constraints. The main result of this effort is: the problem restricted to hard constraints can be solved without backtracking by applying a modified form of the variable selection heuristics $e(n)$ along with the value selection heuristic prom.

5.2 Modelling the Problem as a Finite CSP

5.2.1 Representative Lessons and Offers

Two ways of modelling the grammar school timetabling problem were examined. Both approaches associate lessons with variables, whereas variables are intended to hold the periods lessons must take place. However, the approaches differ in the way couplings are handled. The first approach associates each lesson with a problem variable. Coupled lessons are enforced to take place the same time by means of equality constraints. The second approach associates each offer with a representative offer. An offer is its own representative if it is not member of a coupling. In contrast, for each coupling one of its members is designated to represent all the other offers of the same coupling. Variables are introduced only for the lessons of representative offers. The placement of the lessons of some representative offer has to consider the constraints imposed on the offers represented. As will be justified in section B.14, the second approach is superior to the first approach from an operational point of view, i.e. it allows for more effective constraint propagation. Since propagation performance turned out to be vital for solving the problem, the second approach was chosen.

5.2.2 Levels of Representation

There is a couple of requirements which are formulated in terms of the days some lessons take place (cf. 3.2). In order to faciliate the expression of these requirements in terms of FD constraints, each representative lesson is associated with two variables: a *period-level variable* and a *day-level variable*. A lesson's day-level variable is intended to reflect the days the lesson can take place.

5.2.3 Symmetry Exclusion

By imposing an order on the lessons of an offer, solutions are excluded which differ only in the order of these lessons but are equivalent, and therefore redundant, from a practical point of view. This way of avoiding redundant solutions is usually called symmetry exclusion. Actually, *symmetry exclusion* only undos the effects of a rather technical issue introduced by the CSP framework: lessons of an offer must be distinguished, which is something a human timetabler would not consider as necessary. Furthermore, the operational merits of symmetry exclusion are vital for solving the problem (cf. B.14).

5.2.4 Mapping Timetabling Constraints to FD Constraints

Except for the one-day-off rule, all hard application specific requirements can be expressed by means of four well-established finite domain constraints: in, lt, allDistinct and cumulative. in constraints are used to initialize the domains of variables (cf. B.2), to establish the timetable given and to inform search about the availability of teachers. lt constraints enforce symmetry exclusion and distinct-days constraints. Facility and quality constraints (five-lesson rule and anti-teasing constraints) are expressed by means of cumulative. Non-overlap constraints map to allDistinct constraints. The one-day-off rule maps to nval constraints. Refer to appendix B for the precise mapping and a discussion of operational concerns.

Two models (finite CSPs) M_0 and M_1 will be refered to. The models share the same set of variables, but differ in the sets of constraints. M_1 contains all the hard constraints while M_0 lacks the hard quality constraints, five-lesson rule and one-day-off rule, for example.

5.3 Properties of Constraint Solvers

As reported in 5.1, two approaches to constraint propagation were taken. This section characterizes the constraint solvers in terms of functionality and propagation performance. The first solver, S_0, maintains arc consistency for lt constraints and propagates allDistinct and cumulative constraints in the manner of forward checking. S_0 is not able to process nval constraints. The second solver, S_1, is based on the FD constraint solver of Sicstus Prolog [COC97]. in, lt, allDistinct and cumulative constraints are propagated by Sicstus Prolog. nval is propagated by the method proposed in appendix C. This solver is called after the domain of some variable constrained has been reduced. Sicstus Prolog maintains arc consistency for lt constraints. It maintains global consistency for allDistinct constraints by applying the method proposed by [Rég94]. The level of consistency maintained for cumulative constraints is not known. However, experiments show that the solver's propagation performance is far beyond forward checking. Obviously, the nval solver proposed performs better than forward checking, too, but which level of consistency is achieved has not been examined, either.

5.4 Variable Selection

Problem variables are organized in two levels; there are period-level and day-level variables (cf. 5.2.2). In principle, variable selection might consider the set of all period- and day-level variables. In practice, this approach turns out as a dead-end because heuristics prefer or tend to prefer variables with smaller domains. Because day-level variables have small domains (at most five elements), search will be misguided. Anyway, the day-level variable associated with a period-level variable is not intended to be a problem variable in its own right. Rather day-level variables constitute a level of representation which faciliates the convienent expression of application specific constraints. Furthermore, period-level variables are connected to day-level variables (cf. B.3), which ensures a steady communication between the levels, i.e. as soon as the domain of a day-level variable gets reduced, the domain of the associated period-level variable will be reduced, too. Thus it will reflect the state of the day-level variable. Therefore it makes sense to exclude day-level variables from variable selection. The problems introduced by organizing variables in two levels will require further attention in the computation of looseness (cf. D).

At MTG the next lesson to place is selected following a heuristic specified by a hierarchy of sets of lessons. A hierachy's level contains the lessons which human timetablers assess equally difficult to place. This heuristic translates to a static partial order. This order will be called mtgph. Since the lessons of a level are not ordered, mtgph has to be combined with some other variable selection heuristic.

Consider some finite CSP (X,D,C). By selecting the variable $x \in X$ minimizing

$$|D_x| \cdot \prod_{c \in C_x} L(c)$$

$e(n)$ branches into one of the subproblems maximizing the expected number of solutions. $e(n)$ has been adopted for the problem under study resulting in the heuristic $e'(n)$ which selects the variable $x \in X$ minimizing

$$\frac{1}{e^{2|D_x|}} \cdot \prod_{c \in C_x} L(c).$$

In comparison to $e(n)$, $e'(n)$ pays more attention to variables with small domains. For example, suppose $|D_x| = 3$ and $|D_y| = 4$. $e'(n)$ will decide for x as long as $\dfrac{\prod_{c \in C_y} L(c)}{\prod_{c \in C_x} L(c)} <$

$\frac{e^8}{e^6} \approx 7.39$. In contrast, $e(n)$ will decide for y as soon as $\frac{\prod_{c \in C_y} L(c)}{\prod_{c \in C_x} L(c)} > \frac{4}{3}$.

The problem of computing looseness is dealt with in appendix D.

5.5 Results, Experience and Conclusions

Altogether 18 search procedures have been given a try. All procedures have chronological backtracking, as provided by Prolog, in common. The procedures differ in the solvers and heuristics used.

Solver S_0 has been tested along with $\mathsf{mrv} - \mathsf{enum}$[2] and $(\mathsf{mtgph} + \mathsf{mrv}) - \mathsf{enum}$[3]. These procedures have been applied to the problem M_0, which includes the hard resource constraints but does not include quality constraints.

Solver S_1 has been tested along with $\mathsf{mrv} - \mathsf{enum}$, $\rho - \mathsf{enum}$, $e(n) - \mathsf{enum}$, $e'(n) - \mathsf{enum}$, $(\mathsf{mtgph} + \mathsf{mrv}) - \mathsf{enum}$, $(\mathsf{mtgph} + \rho) - \mathsf{enum}$, $(\mathsf{mtgph} + e(n)) - \mathsf{enum}$, $(\mathsf{mtgph} + e'(n)) - \mathsf{enum}$, $\mathsf{mrv} - \mathsf{prom}$, $\rho - \mathsf{prom}$, $e(n) - \mathsf{prom}$, $e'(n) - \mathsf{prom}$, $(\mathsf{mtgph} + \mathsf{mrv}) - \mathsf{prom}$, $(\mathsf{mtgph} + \rho) - \mathsf{prom}$, $(\mathsf{mtgph} + e(n)) - \mathsf{prom}$ and $(\mathsf{mtgph} + e'(n)) - \mathsf{prom}$. These procedures have been applied to the problem M_1, which extends M_0 by the hard quality constraints.

There is only one procedure which comes up with a solution: $S_1 - e'(n) - \mathsf{prom}$. It solves problem M_1 within 15 minutes on a personal computer equipped with a Pentium 166 microprocessor and the Linux operating system. *Backtracking does not occur.* All other procedures thrash. All tests went on for at least twelve hours, some tests went on for days. The general experience with backtracking is: as soon as backtracking occurs, the search procedure starts thrashing.

The result of $S_1 - e'(n) - \mathsf{prom}$ is satisfying with respect to the timetables of classes, i.e. the quality of education is ensured. However, since M_1 does not include the free-periods rule, the timetables of teachers are characterized by too many free periods. Nine free periods a week is not unusual. Therefore, the timetables are not yet acceptable.

The results lead to the following conclusions.

- It is not sufficient to combine stronger constraint solvers with a well-established and widely used heuristic like mrv. Not only variable selection needs more attention; value selection must be guided, too, as indicated by the success of $S_1 - e'(n) - \mathsf{prom}$ over $S_1 - e'(n) - \mathsf{enum}$. Furthermore, the implementation of $e(n)$ (cf. D) does not attach enough importance to the sizes of domains, as suggested by the success of $S_1 - e'(n) - \mathsf{prom}$ over $S_1 - e(n) - \mathsf{prom}$.
- mtgph is not suitable for a search procedure, which only has chronological backtracking for undoing wrong decisions.

[2] $\mathsf{mrv} - \mathsf{enum}$ means variable selection with mrv and value selection with enum.

[3] mtgph is not a total order (cf. 5.4). Therefore, it has to be combined with some other order. $\mathsf{mtgph} + \mathsf{mrv}$ is such a combination.

- It pays to invest into ordering variables and values, as demonstrated by the success of $S_1 - e'(n) - \mathsf{prom}$ over $S_1 - \mathsf{mrv} - \mathsf{enum}$, for example.

Since the combination $S_0 - e'(n) - \mathsf{prom}$ has not been tested, no conclusions can be drawn with respect to the importance of pruning performance. However, literature on the merits of maintaining arc consistency [SF94, BR96, SF97] suggests that more effective pruning is the key to solving hard and large CSPs effectively. Furthermore, the influence of pruning on the computation of variable and value orders is evident; all heuristics experimented with directly rely on domains. In the computation of orders, values which are not consistent with the current partial solution but have not been pruned are considered consistent with the current partial solution. Therefore, it is reasonable to assume that bad pruning might result in orders that do not reflect the current situation properly and thus misguide search while better pruning allows for making much more informed choices.

Finally, there is experience raising doubts about the robustness and flexibility of $S_1 - e'(n) - \mathsf{prom}$.

- Adding a new requirement often has entailed the need to enhance the search procedure by improving the computation of looseness.
- Abondoning the five-lesson rule causes $S_1 - e'(n) - \mathsf{prom}$ to thrash.

This topic will receive discussion in 7.3.

5.6 Implementation

5.6.1 Components

The implementation consists of three components.

- The constraint generator maps the problem's specification to finite domain constraints. The generator has been implemented with Haskell [Tho96], a modern functional language. The specification (classes, courses, coupling, assignment matrix, and so on) is provided as a set of Haskell data structures.
- The timetabling component, implemented with Sicstus Prolog, provides the search procedure including constraint solvers and heuristics.
- The display component visualizes advances and set-backs of search. It has been implemented with TkHaskell [CVM97], which provides Haskell with a typed, abstract interface to the Tcl/Tk user interface toolkit [Ous94].

Components communicate via files. The constraint generator produces a Prolog database, which is read by the timetabling component. The display component gets connected via a pipe.

5.6.2 The Timetabling Component

The timetabling component provides two constraint solvers (cf. 5.1). S_0 has been implemented with Constraint Handling Rules [Frü95] (embedded into Sicstus Prolog), a high-level, rule-based language especially designed for writing constraint solvers, following the approach proposed by [AM98b]. S_1 is based on the FD constraint solver provided by Sicstus Prolog. The nval solver has been implemented with Prolog and added by means of the global constraint programming interface, a feature of the FD constraint solver faciliating user-defined global constraints [COC97]. The nval solver is consulted whenever the domain of a variable constrained gets reduced.

Timetabling starts with reading the constraints and adding them to the constraint store. Whenever a new constraints arrives, the constraint solver prunes values which are inconsistent with the current state of computation. When all constraints have been added and no inconsistency has been detected, search starts.

Variable selection based on ρ, $e(n)$ and $e'(n)$ requires the looseness of each constraint. In M_1, there are 593 lt and 211 cumulative, allDistinct and nval constraints. Therefore, recomputing the looseness of each constraint before variable selection would be quite expensive. In order to gain efficiency, each constraint gets attached a daemon which is called whenever the domain of a variable constrained gets reduced which means a change in looseness. The daemon records the change in a global, mutable list. Thus, if a variable has to be selected, the constraints are known, the looseness of which has to be updated. Daemons are attached by means of the global constraint programming interface.

Value selection by means of prom requires the size of each subproblem, which remains if a value has been propagated. The size of a value's subproblem is computed as follows. First, the value is propagated. Then the size of the remaining subproblem is measured and the result is returned by raising an exception. Exception handling undos the effects of pruning. As a side effect, values which are inconsistent with the current state of computation but have not been pruned are recognized and excluded from selection.

5.6.3 Experience

Implementing the constraint generator was straight-forward. The same is true for the display component. No knowledge of Tcl/Tk was necessary.

A lot of time was spent debugging the specification, i.e. identifying and removing contradictions. Experience leads to the following distinction of contradictions.

- Good-natured contradictions are those which are recognized by the constraint solver before search starts. With a suitable display, the offending constraint can be identified and some reasoning, more or less, reveals the contradiction. For example, suppose the given timetable places two lessons of a teacher the same period. This error will be recognized immediately after adding the teacher's allDistinct constraint.

- Vicious contradictions are those which are not recognized before search starts and usually lead to thrashing which continues for days. In this case, the human timetabler does not know whether the problem has contradictions or the search procedure is not suitable to solve the problem (cf. 7.3).

Contradictions were mainly due to errors in transcription. On the one hand, the specification received from MTG was not original but had been assembled manually for reasons of data protection, which introduced some errors and ambiguities. On the other hand, early versions of the constraint generator sometimes suffered from bugs. Altogether, it took about two month to obtain a consistent specification.

Another source of thrashing were type errors. Prolog does not have a type system and programs with type errors are executable. Four ways of how type errors affect the execution have been experienced.

- Prolog detects the type error and raises an exception. For example, this happens if a mistyped arithmetic expression is to be evaluated.
- Prolog does not detect the type error and dumps core. This behaviour often occured in mistyped calls to predicates of the FD constraint system.
- Some part of the program does not terminate. For example, this may happen if a recursive list predicate is called with the result variable in place of the input variable.
- Unification fails silently and backtracking occurs. This is likely to cause thrashing. This happens if arguments are mixed up, some argument is missing or mistyped, for example if there is a triple instead of a tuple.

Failing unifications were the most frequent effects of type errors. These type errors were difficult to track down because search may thrash for other reasons, too.

Another problem with Prolog is that nested algebraic expression have to be flattened. With complex expressions this introduces a lot of new variables and makes programs difficult to write, understand and maintain. The declarative producer-consumer view is replaced by a very operational, machine-language-like way of thinking. This contradicts the goals of declarative programming [Kow79, Llo94].

Looking back, a lot of time was wasted because Prolog lacks rudimentary support of software engineering. Programming was inefficient and no fun at all. Concluding, CP environments based on Prolog are not suitable for tackling complex combinatorial problems. This conclusion contradicts earlier, positive experience [BDP96].

6 Related Work

6.1 University Timetabling

[AM98b] reports work on university timetabling. The goal was to compute a weekly timetable placing 89 courses of varying length. There were 42 possible starting points. The assignment of teachers to courses was fixed. Constraints included resource constraints (except for facility constraints), non-overlap constraints among subsets of courses and some soft quality constraints.

The problem was modelled as a PCSP and solved by means of a custom-made FD constraint solver. This solver propagated soft constraints and thus ended up with an estimate of how selecting a value would influence solution quality (cf. 5.1). `mrv` was used for variable selection. Value selection was controlled by the results of soft constraint propagation. Global non-overlap constraints (hard and soft) were propagated in the manner of forward checking. Due to soft constraint propagation the first solution was satisfying and, therefore, optimization was not necessary. The constraint solver was implemented with Constraint Handling Rules [Frü95].

6.2 College Timetabling

[HW95] reports work on college timetabling which aimed at computing a weekly timetable placing 91 courses of varying length. There were 34 teachers and the assignment of teachers to courses was fixed. Courses were to be held in seven rooms of varying size between 8:15 am and 5 pm with a possible starting point every quarter of an hour. Constraints included resource constraints, couplings, non-overlap constraints among subsets of courses, the demand for two days off for each teacher and some soft quality constraints.

The problem was modelled as a PCSP and solved by means of the FD constraint system of the CP tool Oz [Smo95]. `mrv` was used for variable selection. For value selection a custom-made static order was applied. Global non-overlap constraints were translated to binary non-overlap constraints; the two-days-off rule was implemented by means of reified constraints. *Reification* associates a constraint with a variable reflecting its state with respect to entailment and violation. This mechanism is two-way, i.e. the variable may be used to enforce the satisfaction or violation of the constraint. In this case, application of reification resulted in a procedure performing forward checking.

Since the value order's design took soft constraints into account, they played an active role in search to some extent. However, soft constraints were not propagated (in the sense of [AM98b]). Branch & bound was used for further optimization.

7 Discussion and Future Work

7.1 Summary

This work aimed at generating a timetable for a german grammar school by means of CP. The givens were sets of classes, courses and couplings, a matrix assigning teachers to offers, availabilities of teachers, a list associating rooms with equipment and a timetable placing lessons of sports education and of the course phase. The timetable was to obey a set of hard and soft constraints demanding for physical feasibility, conformance to pedagogical principles and teachers' wishes. The problem was compiled in cooperation with MTG, modelled as a finite CSP and solved in part by a search procedure embedding constraint solving and heuristics for variable and value selection into chronological backtracking.

The constraint solver applied is based on Sicstus Prolog and cannot deal with soft constraints. Variable selection is based on $e(n)$, a heuristic proposed recently, which dyamically branches into one of the subproblems which are expected to have a maximum number of solutions. Value selection uses prom, proposed a few years ago, which dynamically branches into one of the biggest subproblems. The resulting search procedure solved the problem restricted to the hard constraints within 15 minutes on a personal computer, which is a very reasonable period of time. No backtracking occured. The plan is physically feasible and satisfying with respect to the quality of education, but teachers' timetables are not acceptable because teachers' wishes and soft quality constraints are not considered.

7.2 Contributions

This work contributes in several ways.

- It presents a large timetabling problem from practice and the techniques used by human timetablers to solve the problem.
- It shows how to model the problem restricted to the hard constraints by means of five finite domain constraints, four of which are well-established while nval is a new and general abstraction, which might be useful in contexts other than timetabling, too. Operational concerns are discussed.
- It shows how to propagate nval constraints and how to put the philosophy underlying $e(n)$ into practice. The problem of computing looseness, a measure $e(n)$ makes use of, receives an extensive formal treatment based on probability theory. Formal treatment is motivated by the success of $e(n)$.
- Several search procedures differing in pruning performance and heuristics are introduced. Results and experiences from experimental evaluation are summarized. It is concluded that, if a difficult problem is to be solved, it pays to invest into look-ahead, i.e. into propagation and the ordering of variables and values. This conclusion is in line with recent reports on the merits of look-ahead [SF94, BR96, SF97].

- Difficulties experienced when putting the CP approach into practice are reported and discussed. It is concluded that, if a Prolog-based CP system is used, the CP approach to timetabling mainly suffers from Prolog's insufficient support of software engineering .

7.3 Reuseability

This work concentrated on solving one instance of one certain school's timetabling problem. Both the model and the search procedure have been tuned to solve this instance. This raises concern about robustness and reuseability.

With the success reported, it seems that only a treatment for the free-periods rule has to be added in order to obtain the core of a timetabling system suitable for the MTG. However, as reported (cf. 5.5), the procedure is very sensitive to major changes in the problem structure. In particular, removing constraints leads to thrashing. In principle, this is not a surprise, since, given a set of NP-hard problems, a procedure given may be able to decide some problems efficiently while deciding the other problems requires exponential time. This raises the question whether the search procedure presented is able to cope with the changes in problem structure entailed by the changes from year to year, like changes in the number of pupils and teachers (cf. 3.1). More confidence may be gained by examining past and future instances.

Of course, the same considerations apply if some timetabling problem of some other german grammar school is to be solved. Furthermore, the model, tailored to the needs of the MTG, is reuseable only in part, because the quality standards of other schools may differ considerably. For example, a school in rural area may attach much importance to the support of car pools while the quality of education receives fewer attention than at MTG. Thus, only the model restricted to resource constraints is likely to be reuseable.

7.4 Future Work

- One important step towards the core of a timetabling system suitable for the MTG consists in modelling and processing soft constraints. If soft constraints are to play an active role in the search process, effective soft constraint solvers have to be provided and results from soft constraint propagation must find their way into heuristics.
- Since the search procedure's flexibility is in question, past and future instances of the MTG timetabling problem have to examined.
- In order the increase the seach procedure's flexibility, several approaches may be explored. Note that the approaches are complementary.
  - The procedure applied by human timetablers can be seen as a combination of look-ahead, heuristics and local repair (cf. 3.4). Since this method has been applied successfully over the years, local repair deserves attention.
  - If the system recognizes that there has been a wrong decision, the user may be consulted in the process of recovering from the wrong decision. This

requires the development of a suitable interaction model, which attaches particular importance to cognitive aspects, if the user is to be faced which a large and complex data set.

- As it turned out, effective pruning plays a central role if there is only chronolgical backtracking to undo wrong decisions. In particular, the merits of heuristics depend on the results of pruning (cf. 5.5). Therefore, it is reasonable to look for better constraint solvers. cumulative and nval are candidates.
- Considering the central part of $e(n)$ in the success reported, it is worth to think about improving the computation (or approximation) of looseness. In the long term, if $e(n)$ proves to be useful, it would be justified to require constraint solvers to provide the service of computing (or approximating) looseness. It might be possible to compute looseness efficiently as a by-product of constraint solving. The other way round, computing looseness can be used as a decision procedure.

A Notations

If p is a predicate, $\{x : p(x)\}$ is the set of x such that $p(x)$ is true. $\{x \in X : p(x)\}$ means $\{x : x \in X \wedge p(x)\}$, $\{X \subset Y : p(X)\}$ means $\{X : X \subset Y \wedge p(X)\}$, and so on.

If X is a set, $|X|$ is the cardinality of X. $\mathcal{P}(X) = \{Y : Y \subset X\}$ is the power set of X.

If R is a relation, $\mathrm{dom}(R) = \{a : (a,b) \in R\}$ and $\mathrm{ran}(R) = \{b : (a,b) \in R\}$.

$\mathcal{N}$ is the set of natural numbers. $\mathcal{R}$ is the set of real number. $\mathcal{B} = \{\top, \bot\}$ is the set of boolean values. $\top$ means *true*, $\bot$ means *false*. If $a, b \in \mathcal{N}$, $[a,b] = \{i : a \leq i \leq b\}$.

If T is a set, $x : T$ means $x \in T$. If $x : T$, x is said to have *type* T. In this context, $A \to B$ is the set of functions which map elements from A to B. $A \to B \to C$ means $A \to (B \to C)$. (A,B) means $A \times B$. $\{A\}$ means $\mathcal{P}(A)$.

Suppose $\sim : A \times A$ is an equivalence relation. Then $\bar{a}$ means $\{b \in A : b \sim a\}$ and $A/\sim$ means $\{\bar{a} : a \in A\}$.

The minus is overloaded; it is used to subtract numbers and sets.

B Modelling the Problem as a Finite CSP

This section's purpose is to show precisely how timetabling constraints have been mapped to FD constraints.

B.1 Preliminaries

The following definitions introduce some basic concepts related to offers, lessons and variables.

Definition B.1. Offer denotes the set of offers. The operator $\tilde{}: \mathsf{Offer} \to \mathsf{Offer}$ associates an offer with its representative. However, if some offer o is not member of a coupling, $\tilde{o} = o$. An offer o is called *representative* if $\tilde{o} = o$. Otherwise it is called *represented*. The operator $\bar{}: \mathsf{Offer} \to \{\mathsf{Offer}\}$ associates a representative offer with the offers it represents. If o is not member of a coupling, $\bar{o} = \{o\}$.

For example, let $c = \{o_1, o_2, o_3\}$ be a coupling specifying that the offers o_1, o_2 and o_3 must take place the same time. Suppose o_1 is the designated representative of c. Then $\tilde{o}_1 = \tilde{o}_2 = \tilde{o}_3 = o_1$ and $\bar{o}_1 = c$, but $\bar{o}_2$ and $\bar{o}_3$ are not defined.

Definition B.2. Lesson denotes the set of lessons. The function $\mathsf{lessons} : \mathsf{Offer} \to \{\mathsf{Lesson}\}$ associates an offer with its lessons. A lesson l of some offer o is called *representative* if o is representative. Otherwise it is called *represented*.

Definition B.3. pl denotes *period level*, dl denotes *day level*. Level denotes $\{\mathsf{pl}, \mathsf{dl}\}$. $\mathcal{V}^l$ denotes the set of l-level variables. $\mathcal{V}$ denotes $\mathcal{V}^{\mathsf{pl}} \cup \mathcal{V}^{\mathsf{dl}}$. The operator $\uparrow: \mathcal{V}^{\mathsf{pl}} \to \mathcal{V}^{\mathsf{dl}}$ associates a period-level variable with its day-level counterpart. The operator $\downarrow: \mathcal{V}^{\mathsf{dl}} \to \mathcal{V}^{\mathsf{pl}}$ works the other way round; it associates a day-level variable with its period-level counterpart. The operator $\hat{}: \mathsf{Lesson} \to \mathcal{V}^{\mathsf{pl}}$ associates a representative lesson with its period-level variable; it is not defined for represented lessons. The relation $\leq_V$: $\mathcal{V}^{\mathsf{pl}} \times \mathcal{V}^{\mathsf{pl}}$ is a partial order on the set $\mathcal{V}^{\mathsf{pl}}$ of period-level variables. $x <_V$ means that $x \leq_V y$ and $x \neq y$.

For example, if x is a period-level variable and u is the associated day-level variable, $x\uparrow = u$ and $u\downarrow = x$.

The following definitions make classes, equipment, subjects and teachers available to a formal method of modelling the timetabling problem.

Definition B.4. Class denotes the set of classes. The set $\mathsf{Equipment}$ includes those items of equipment the availability of which has to be guaranteed. (For the problem instance under study this set includes biology, chemistry and physics.) $\mathsf{Subject}$ denotes the set of subject. $\mathsf{Teacher}$ denotes the set of teachers.

Definition B.5. $\mathsf{offers} : (\mathsf{Teacher} \cup \mathsf{Class}) \to \mathsf{Offer}$ associates a teacher or a class with his, her or its offers. $\mathsf{subject} : \mathsf{Offer} \to \mathsf{Subject}$ associates an offer with its subject. $\mathsf{teachers} : \mathsf{Offer} \to \{\mathsf{Teacher}\}$ associates an offer with its teachers. $\mathsf{available} : \mathsf{Teacher} \to \mathcal{N}$ associates a teacher with the periods he or she is available. $\mathsf{grade} : \mathsf{Class} \to \mathcal{N}$ associates a class with its grade. $\mathsf{supply} : \mathsf{Equipment} \to \mathcal{N}$ associates an item of equipment with the quantity of its supply. $\mathsf{timetable} : \mathsf{Lesson} \to \mathcal{N}$ associates a representative lesson with its given period. $AMSH = \{\mathrm{arts}, \mathrm{music}, \mathrm{sports}, \mathrm{handwork}\}$.

$$\left\{ \mathsf{in}(\hat{l},[1,60]) : \begin{array}{rl} & o \in \mathsf{Offer}^{\sim} \\ \wedge & l \in \mathrm{lessons}(o) \end{array} \right\}$$

$$\left\{ \mathsf{in}(\hat{l}\uparrow,[1,5]) : \begin{array}{rl} & o \in \mathsf{Offer}^{\sim} \\ \wedge & l \in \mathrm{lessons}(o) \end{array} \right\}$$

$$\left\{ \mathsf{levelLink}(\hat{l},\hat{l}\uparrow) : \begin{array}{rl} & o \in \mathsf{Offer}^{\sim} \\ \wedge & l \in \mathrm{lessons}(o) \end{array} \right\}$$

$$\left\{ \mathsf{lt}(x,y) : \begin{array}{rl} & o \in \mathsf{Offer}^{\sim} \\ \wedge & \forall l \in \mathrm{lessons}(o).\, l \notin \mathrm{dom}(\mathrm{timetable}) \\ \wedge & x,y \in \mathrm{lessons}(o)^{\wedge} \\ \wedge & x <_V y \end{array} \right\}$$

$$\left\{ \mathsf{in}(\hat{l},\{i\}) : (l,i) \in \mathrm{timetable} \right\}$$

$$\left\{ \mathsf{in}(\hat{l},\mathrm{available}(t)) : \begin{array}{rl} & t \in \mathsf{Teacher} \\ \wedge & o \in \mathrm{offers}(t)^{\sim} \\ \wedge & l \in \mathrm{lessons}(o) \end{array} \right\}$$

Figure 1: Initializing domains, linking levels, excluding symmetries, establishing the timetable given and ensuring teachers' availability

B.2 Initializing Domains

Before starting off the search, the domains of the variables must be initialized. For this purpose, *initialization constraints* are introduced for the period-level and day-level variables of each representative lesson. Period-level variables are allowed to range from 1 to 60, i.e. there are twelve periods a day, where 1 denotes the first period on Monday, 13 denotes the first period on Tuesday, and so on. Day-level variables are allowed to range from 1 to 5, where 1 denotes Monday, 2 denotes Tuesday, and so on. Initial domains are defined by means of in constraints. Suppose o is an offer from the set $\mathsf{Offer}^{\sim}$ of representative offers and l is a lesson of o, i.e. $l \in \mathrm{lessons}(o)$. The first in constraint generated for l constrains the period-level variable $\hat{l}$ of l to take a value from the set $[1,60]$. The second in constraint constrains the day-level variable $\hat{l}\uparrow$ of l to take a value from the set $[1,5]$.

B.3 Linking Levels

In order to keep the search procedure well informed, a period-level variable x has to communicate reductions of its domain to its associated day-level variable $x\uparrow$, and vice versa. This is the task of level-linking constraints. There is one level-linking constraint for each representative lesson.

$$\left\{ \mathsf{in}(\hat{l}, I) : \begin{array}{ll} & c \in \mathsf{Class} \\ \wedge & o \in \text{offers}(c)^{\sim} \\ \wedge & l \in \text{lessons}(o) \\ \wedge & l \notin \text{dom}(\text{timetable}) \end{array} \right\}$$

where

$$I = \bigcup \{[12(j-1)+1, 12(j-1)+6] : 1 \leq j \leq 5\}$$

Figure 2: Enforcing AM constraints

B.4 Symmetry Exclusion

Symmetry exclusion is enforced by means of period-level lt constraints. The order of lessons is defined by $<_V$. No symmetry exclusion is performed for offers with one or more lessons already placed by the timetable given. On the one hand, this inhibits symmetry exclusion for course phase offers and sports education. On the other hand, this avoids contradictions of $<_V$ with the order of lessons implied by the timetable.

B.5 The Timetable Given

The timetable given is enforced by means of period-level in constraints.

B.6 Availability Constraints

Availability constraints restrict the times teachers and equipment are available. However, the availability of equipment does not need consideration, because the timetable of sports education is given and all the other facilities are available all the time. Availability is enforced by means of period-level in constraints.

B.7 AM Constraints

AM constraints demand that lessons of class offers have to take place in the morning. They are enforced by means of period-level in constraints. By refering to the members of Class, attention is restricted to offers for classes. Conflicts with the timetable given, which schedules some lessons of the eleventh grade for afternoon periods, are avoided by considering the domain of timetable as defining exeptions from the general AM requirement.

$$\left\{ \mathsf{allDistinct}(\hat{L}) : \begin{array}{ll} & x \in \mathsf{Class} \cup \mathsf{Teacher} \\ \wedge & L = \bigcup_{o \in \mathrm{offers}(x)^\sim} \mathrm{lessons}(o) \\ \wedge & |L| \geq 2 \end{array} \right\}$$

Figure 3: Enforcing non-overlap constraints

B.8 Non-overlap Constraints

Non-overlap constraints are required for classes and teachers. Clashes are avoided by means of period-level allDistinct constraints. There is an allDistinct constraint for each class and each teacher. For example, suppose $c \in \mathsf{Class}$. The allDistinct constraint for c constrains the set $\hat{L} = \left\{\hat{l} : l \in L\right\}$ of period-level variables where $L = \bigcup_{o \in \mathrm{offers}(c)^\sim} \mathrm{lessons}(o)$ is the set of representative lessons of c.

B.9 Facility Constraints

Facility constraints make sure that equipment is available in case the supply of some item of equipment is limited. Limits are enforced by means of period-level cumulative constraints. There is a cumulative constraint for each item of equipment. Assuming that each offer may require at most one item of some equipment e, the number of items required by some representative offer $o \in \mathsf{Offer}^\sim$ is the number of offers from the set $\bar{o}$ of offers represented by o which require e, whereas offers sharing teachers must be identified, for a teacher cannot teach in more than one room the same time. Offers sharing teachers are identified with the aid of the equivalence relation $\bowtie$. $n'(e)$ associates each $o \in \mathsf{Offer}^\sim$ with the accumulated need of $\bar{o}$ for e. $n(e)$, based on $n'(e)$, associates each representative lesson with its need for e. Note that $\mathrm{ran}(n(e)) = \mathrm{ran}(n'(e)) = \{1\}$ for the problem instance under study here: Biology, chemistry and physics lessons always share teachers if coupled. Other subjects requiring appropriate facilities (arts, music, sports) do not have to be considered here (cf. 3.5). This observation will be exploited in the computation of looseness (cf. D).

B.10 The Five-Lesson Rule

The five-lesson rule demands that teachers must not be compelled to teach more than five lessons a day. In other words, if t is a teacher, a value may be used at most five times for the day-level variables of the lessons of t. This requirement is enforced by means of a day-level cumulative constraint. Of course, if t has less than six lessons to teach a week, the five-lesson rule cannot get violated. Contradictions with the timetable given are avoided by excluding teachers which have been assigned more than five lessons a day. $n(t,d)$ is the set of representative lessons teacher t has to teach the day d.

$$\left\{ \mathsf{cumulative}(\mathrm{dom}(n(e)), d, n(e), \mathrm{supply}(e)) : \begin{array}{ll} & e \in \mathsf{Equipment} \\ \wedge & n(e) \neq \emptyset \\ \wedge & \displaystyle\sum_{x \in \mathrm{dom}(n(e))} n(e)(x) > \mathrm{supply}(e) \end{array} \right\}$$

where

$$\forall x.\, d(x) = 1$$

and

$$\begin{array}{rcl} f & : & \mathsf{Equipment} \to \mathcal{V} \to \mathcal{N} \\ n(e) & = & \left\{ (\hat{l}, m) : \begin{array}{ll} & (o,m) \in n'(e) \\ \wedge & l \in \mathrm{lessons}(o) \end{array} \right\} \\ \\ n' & : & \mathsf{Equipment} \to \mathsf{Offer} \to \mathcal{N} \\ n'(e) & = & \left\{ (o,m) : \begin{array}{ll} & o \in \mathsf{Offer}^{\sim} \\ \wedge & m = |\{o' \in \bar{o} : o' \text{ requires } e\} \,/ \bowtie| \\ \wedge & m > 0 \end{array} \right\} \\ \\ \bowtie & : & (\mathsf{Offer}, \mathsf{Offer}) \\ \bowtie & = & \left\{ (o, o') \in \mathsf{Offer} \times \mathsf{Offer} : \mathrm{teachers}(o) = \mathrm{teachers}(o') \right\} \end{array}$$

Figure 4: Enforcing facility constraints

$$\left\{ \mathsf{cumulative}(\hat{L}\uparrow, d, r, 5) : \begin{array}{ll} & t \in \mathsf{Teacher} \\ \wedge & L = \bigcup_{o \in \mathrm{offers}(t)} \mathrm{lessons}(\tilde{o}) \\ \wedge & |L| > 5 \\ \wedge & \forall d \in [1, \ldots, 5].\, n(t,d) \leq 5 \end{array} \right\}$$

where

$$\forall x.\, d(x) = r(x) = 1$$

and

$$n(t,d) = \left\{ l : \begin{array}{ll} & o \in \mathrm{offers}(t)^{\sim} \\ \wedge & l \in \mathrm{lessons}(o) \\ \wedge & (l,a) \in \mathrm{timetable} \\ \wedge & \mathrm{day}(a) = d \end{array} \right\}$$

Figure 5: Enforcing the five-lesson rule

$$\left\{ \mathsf{nval}(\hat{L}\uparrow, 4) : \begin{array}{ll} & t \in \mathsf{Teacher} \\ \wedge & L = \bigcup_{o \in \mathrm{offers}(t)^{\sim}} \mathrm{lessons}(o) \\ \wedge & 4 < |L| < 17 \end{array} \right\}$$

Figure 6: Enforcing the one-day-off rule

B.11 The One-Day-Off Rule

If a teacher has less than seventeen lessons to teach a week, he has the right to one day off a week. In other words, if t is a teacher and satisfies the premise of the one-day-off rule, at most four values may be used for the day-level variables of the lessons of t. However, the set of admissible values is not known in advance; any day or days may be missing. This requirement is enforced by means of a day-level nval constraint. Of course, if t has less than five lessons to teach a week, the one-day-off rule cannot get violated.

B.12 Anti-Teasing Constraints

For pedagogical reasons, at most two lessons of arts, music, sports and handwork may take place the same day in classes of grades five, six and seven. This requirement is

$$
\left\{ \mathsf{cumulative}(\hat{L}\uparrow, d, r, 2) : \begin{array}{ll} & c \in \mathsf{Class} \\ \wedge & \mathrm{grade}(c) \in \{5,6,7\} \\ \wedge & L = \bigcup_{\substack{o \in \mathrm{offers}(c) \\ \mathrm{subject}(o) \in AMSH}} \mathrm{lessons}(\tilde{o}) \\ \wedge & |L| > 2 \end{array} \right\}
$$

$$
\cup \left\{ \mathsf{cumulative}(\hat{L}\uparrow, d, r, 2) : \begin{array}{ll} & c \in \mathsf{Class} \\ \wedge & o \in \mathrm{offers}(c) \\ \wedge & \neg \left(\begin{array}{ll} & \mathrm{grade}(c) \in \{5,6,7\} \\ \wedge & \mathrm{subject}(o) = \mathrm{maths} \end{array} \right) \\ \wedge & L = \mathrm{lessons}(\tilde{o}) \\ \wedge & |L| > 2 \end{array} \right\}
$$

where

$$
\forall x.\, d(x) = r(x) = 1
$$

Figure 7: Enforcing anti-teasing constraints

enforced by means of $\mathsf{cumulative}$ constraints. There is one $\mathsf{cumulative}$ constraint for each class. Furthermore, at most two lessons of an offer may take place the same day. Of course, if o is an offer and o has less than three lessons, this anti-teasing constraint cannot get violated.

B.13 Distinct-Days Constraints

Hard distinct-days constraints demand that an offer's lesson must not take place the same day. This requirement may be expressed by day-level $\mathsf{allDistinct}$ constraints. However, symmetry exclusion on period level allows for expressing a distinct-days constraint by enforcing an order over the offer's day-level variables. Ordering instead of inequality is more efficient and effective. Refering to $<_V$ guarantees that the order does not contradict the order imposed by symmetry exclusion. For an offer $o \in \mathsf{Offer}^{\sim}$ the distinct-days requirement will only be enforced if there is at least one $o' \in \bar{o}$ with $p(o)$, i.e. which requires distinct days for its lessons, and all teachers involved (T_o) have some days of presence D in common with $|D| \geq |\mathrm{lessons}(o)|$. This avoids contradictions of the distinct-days requirement with the availability of part-time teachers.

$$
\left\{ \mathsf{lt}(x\uparrow, y\uparrow) : \begin{array}{ll} & o \in \mathsf{Offer}^{\sim} \\ \wedge & \text{grade}(o) < 12 \\ \wedge & \exists o' \in \bar{o}.\ p(o') \\ \wedge & T_o = \bigcup_{o' \in \bar{o}} \text{teachers}(o') \\ \wedge & D = \bigcap_{t \in T_o} \{\text{day}(i) : i \in \text{available}(t)\} \\ \wedge & |D| \geq |\text{lessons}(o)| \\ \wedge & x, y \in \text{lessons}(\tilde{o})^{\wedge} \\ \wedge & x <_V y \end{array} \right\}
$$

where

$$
p(o) \leftrightarrow \left(\begin{array}{ll} & \left(\begin{array}{ll} & \text{grade}(c) \in \{5,6,7\} \\ \wedge & \text{subject}(o) = \text{maths} \end{array} \right) \\ \vee & \left(\begin{array}{ll} & |\text{lessons}(o)| = 2 \\ \wedge & \text{subject}(o) \notin AMSH \end{array} \right) \\ \vee & \quad |\text{lessons}(o)| = 3 \end{array} \right)
$$

Figure 8: Enforcing distinct-days constraints

B.14 Operational Concerns

Modelling a combinatorial problem as a CSP means to give a set of variables and a set of constraints. In general, the design of the CSP is neither obvious nor arbitrary. Several models, equivalent in their logical meaning, may be possible. In this case, the question may arise, which model will exhibit the best operational behaviour in the search procedure targeted, i.e. which models will lead to a solution within a reasonable period of time. Therefore, the properties of the constraint solvers available or targeted and difficulties in computation of search control information may influence modelling heavily.

Two ways of modelling the grammar school timetabling problem have been examined (cf. 5.2.1). The first approach has variables for each lesson. Coupled lessons are enforced to take place the same time by means of equality constraints. The second approach designates representatives and there are no variables for represented lessons. The second approach is motivated by the insight that for each member of a coupling the constraints imposed on each of the other members must be taken into account, too. Thus the second approach seems to be more natural. In contrast to the first approach, it does not lead to the need for a lot of equality constraints. Most important, it allows for expressing a non-overlap constraint (for a teacher or for a class) by a single allDistinct constraint while the first approach would require a set of allDistinct and neq constraints. Modelling by a single allDistinct constraint is advantageous, because, in the CP environment of Sicstus Prolog, a single allDistinct constraint propagates much better than the logically equivalent network resulting from the first approach [Rég94, COC97]. Since propagation performance turned out to be vital for solving the problem, the second approach was chosen.

Enforcing symmetry exclusion does not really specify a new problem but restricts the space of solutions by excluding redundant solutions. Since redundant solutions do no harm in general, symmetry exclusion constraints are often classified as *redundant constraints*. However, symmetry exclusion may entail more pruning and thus improve the overall quality of search control information, which in turn may speed up the search for a solution. For solving the problem instance dealed with here, symmetry exclusion even turned out to be mission critical, i.e. no solution was found without excluding symmetries. However, there is drawback. With symmetry exclusion each lesson of an offer should get imposed the same constraints. For example, with the first approach the demand for a double lesson for an offer can be expressed by a linear equation ($x = y + 1$). Which lessons of the offer get constrained does not matter as long as other constraints do not interfere. In contrast, with symmetry exclusion determining two designated lessons to take place one after another would mean to determine the relative position of the double lesson due to the order of lessons. This might exclude solutions which are not redundant. Thus the search for a solution might become more difficult. Therefore, enforcing double lessons under the conditions of symmetry exclusion demands for a specialized global constraint. This example demonstrates, that some requirements cannot be stated convienently under the conditions of symmetry exclusion.

C Solving nval Constraints

Tasks of a constraint solver include propagation and the decision of consistency with respect to the current state of computation. Another service a constraint solver may be demanded for is the decision of *entailment*. A constraint is said to be entailed by the current state of computation, if any extension of the current partial instantiation will satisfy the constraint. For example, if $x = 3$ and $|D_y| = \{5,6\}$, the constraint $x < y$ is entailed because no instantiation of x and y is possible which violates $x < y$. If a constraint is entailed, further calls to its decision procedure and to its propagation procedure are superfluous because the constraint is satisfied anyway and propagation will not contribute to pruning any more. The nval solver proposed below performs propagation and decides consistency as well as entailment. Decision of entailment is motivated by operational concerns.

For $\mathsf{nval}(X,n)$, the solver distinguishes three cases by considering the set

$$U = \bigcup_{\substack{x \in X \\ |D_x|=1}} D_x$$

of values already in use.

1. If $|U| > n$, too many values are already in use. In this case, $\mathsf{nval}(X,n)$ is violated.
2. If $|U| = n$, the maximum number of values is reached. In this case, propagation restricting the domains of all free variables of X to values from U implies entailment.
3. If $|U| < n$, examining the set

   $$D = \{D_x : x \in X \wedge D_x \cap U = \emptyset\}$$

   of domains having no value in common with U reveals further cases of interest.

   (a) $|\bigcup D| \leq k$, where $k = n - |U|$ is the number of new values which may be used without violating $\mathsf{nval}(X,n)$. In this case, violation is not possible at all. This implies entailment.

   (b) If $|\bigcup D| > k$, violation is possible. The question arises whether there is an instantiation which uses no more than k new values. This is the case if the set

   $$S = \left\{A \subset \bigcup D : |A| = k \wedge \forall B \in D.\, A \cap B \neq \emptyset\right\}$$

   is not empty. Otherwise the constraint is violated. Note that those $A \subset \bigcup D$ with $|A| < k$ do not have to be considered because $\forall A' \subset \bigcup D.\, (|A'| < k \rightarrow \exists A \subset \bigcup D.\, |A| = k \wedge A' \subset A)$. If $S \neq \emptyset$, propagation constraining the free variables of X to take values from U may eliminate values the usage of which would imply violation. For example, consider a situation with two variables u and v with $D_u = \{1,2\}$ and $D_v = \{2,3\}$ and only one more new value admissible. In this case, $S = \{\{2\}\}$ because neither the value 1 for u nor the value 3 for v can be part of a solution. Therefore, 1 and 3 can be eliminated. In this case, elimination implies entailment. In general, elimination implies entailment only if $|\bigcup S| = k$.

$$
\begin{aligned}
\text{solve} &: \mathcal{C} \to (\{\mathcal{C}\}, \mathcal{B}) \\
\text{solve}(\mathsf{nval}(X,n)) &= \begin{cases} (\emptyset, \bot), & \text{if } |U| > n \\ (C_1, \top), & \text{if } |U| = n \\ R, & \text{if } |U| < n \end{cases} \\
k &= n - |U| \\
U &= \bigcup_{\substack{x \in X \\ |D_x| = 1}} D_x \\
D &= \{D_x : x \in X \wedge D_x \cap U = \emptyset\} \\
S &= \{A \subset \bigcup D : |A| = k \wedge \forall B \in D.\, A \cap B \neq \emptyset\} \\
R &= \begin{cases} (\emptyset, \top), & \text{if } |\bigcup D| \leq k \\ (\emptyset, \bot), & \text{if } |\bigcup D| > k \wedge S = \emptyset \\ (C_2, e), & \text{if } |\bigcup D| > k \wedge S \neq \emptyset \end{cases} \\
C_1 &= \{\mathsf{in}(x, U) : x \in X \wedge |D_x| > 1\} \\
C_2 &= \left\{\mathsf{in}(x, U \cup \bigcup S) : x \in X \wedge |D_x| > 1\right\} \\
e &\leftrightarrow \left|\bigcup S\right| = k
\end{aligned}
$$

Figure 9: Specification of the nval solver

The nval solver's specification, as given in figure 9, assumes that constraints are solved by calls to the function solve. solve takes a constraint and returns a set of constraints and a boolean. The constraints returned are intended to communicate the results of look-ahead, i.e. the reduction of domains can be given rise by means of in constraints. The boolean is intended to signal entailment ($\top$) or inconsistency ($\bot$). [4]

Experiments have shown that the third case is necessary for solving M_1 efficiently. If no propagation is performed in case $|U| < n$, thrashing occurs. The same is true if propagation is performed only if $|U| = n - 1$.

[4] solve is a simplification of Sicstus Prolog's FD global constraint programming interface [COC97].

D Computing Looseness

Computing ρ, $e(n)$ and $e'(n)$ involves the computation of looseness (cf. 4.5). The looseness of a constraint is the probability of its satisfaction. Computing looseness has turned out to be quite involved. However, since $e'(n)$ is the only heuristics which lead to a solution within a reasonable period of time, the problem of computing looseness deserves an extensive and precise treatment. Therefore, this section will give a rather formal account of the topic. Treatment is based on probability theory and includes definitions of appropriate probability spaces, a discussion of aspects of computability, an algorithm for computing the looseness of nval constraints, which is efficient enough for the problem under study, along with a proof of its correctness, a method for approximating the looseness of global constraints and a short discussion of independence assumptions underlying ρ. Definitions from B.1 apply.

D.1 Instantiations and Probability

The formal treatment affords the introduction of some basic notions.

Definition D.1. $\mathcal{C}^l$ denotes the set of l-level constraints. $\mathcal{C}$ denotes $\mathcal{C}^{\mathsf{pl}} \cup \mathcal{C}^{\mathsf{dl}}$.

Definition D.2. A function $\sigma : \mathcal{V}^l \to \mathcal{N}$ is called an *l-level instantiation* if $\mathrm{dom}(\sigma) = \mathcal{V}^l$. Σ^l denotes the set of l-level instantiations and Σ denotes $\Sigma^{\mathsf{pl}} \cup \Sigma^{\mathsf{dl}}$.

The function S (see figure 10) associates a constraint with its meaning in terms of period-level instantiations. S is based on s and π. s associates an l-level constraints with its meaning in terms of l-level instantiations. π projects day-level instantiations to period-level.

Definition D.3. If $c \in \mathcal{C}$ and $\sigma \in s(c) \cup S(c)$, σ is called *c-satisfying*. A c-satisfying instantiation is also said to *satisfy c*.

Definition D.4. $\varphi : \{\Sigma\} \to \{\Sigma\}$, defined by

$$\varphi(A) = \{\sigma \in A : \forall x \in \mathrm{dom}(\sigma).\ \sigma(x) \in D_x\},$$

is the set of instantiations from A, which take the current state of computation into account, i.e. which do not assign values which have already been eliminated by the search procedure. If $\sigma \in \varphi(\Sigma)$, σ is said to be *admissible*.

In this section, computing the looseness of a constraint c is understood as determining the probability of the truth of the proposition

$$\boxed{A(c) = \exists\sigma.\ \sigma \text{ is an admissible period-level instantiation satisfying } c.}$$

Computing probability requires a problem specific probability space.

Definition D.5. A *probability space* is a tuple $(\Omega, \mathcal{F}, P)$. Ω is the *space of events* under consideration. $\mathcal{F}$ is a subset of the power set of Ω. Each element of $\mathcal{F}$ can be thought of as describing a proposition A in terms of events which imply A. P is a *probability distribution*, i.e. a function associating each proposition from $\mathcal{F}$ with the probability of its truth.

$$
\begin{array}{rcl}
S & : & C \to \left\{\Sigma^{\mathsf{pl}}\right\} \\
S(\mathsf{true}) & = & \Sigma^{\mathsf{pl}} \\
S(\mathsf{false}) & = & \emptyset \\
S\left(\bigwedge C\right) & = & \bigcap_{c \in C} S(c) \\
S(c) & = & \bigcup_{\sigma \in s(c)} \pi(\sigma), \text{ if } c \neq \mathsf{true} \text{ and } c \neq \mathsf{false} \\
 & & \\
\pi & : & \Sigma \to \left\{\Sigma^{\mathsf{pl}}\right\} \\
\pi(\sigma) & = & \begin{cases} \{\sigma\}, \text{ if } \sigma \in \Sigma^{\mathsf{pl}} \\ \left\{\tau \in \Sigma^{\mathsf{pl}} : \forall x \in \mathrm{dom}(\tau).\ \mathrm{day}(\tau(x)) = \sigma(x\uparrow)\right\} \\ \text{otherwise} \end{cases} \\
 & & \\
s & : & C^l \to \left\{\Sigma^l\right\} \\
s(\mathsf{neq}(x,y)) & = & \left\{\sigma \in \Sigma^l : \sigma(x) \neq \sigma(y)\right\} \\
s(\mathsf{lt}(x,y)) & = & \left\{\sigma \in \Sigma^l : \sigma(x) < \sigma(y)\right\} \\
s(\mathsf{allDistinct}(X)) & = & \left\{\sigma \in \Sigma^l : \forall x, y \in X.\ x \neq y \to \sigma(x) \neq \sigma(y)\right\} \\
s(\mathsf{cumulative}(X,d,r,n)) & = & \left\{\sigma \in \Sigma^l : \forall i \in \mathrm{ran}(\sigma).\ \sum_{x \in X} A(i,x) \leq n\right\} \\
 & & \text{where} \\
 & & A(i,x) = \begin{cases} r(x), & \text{if } \sigma(x) \leq i < \sigma(x) + d(x) \\ 0 & \text{otherwise} \end{cases} \\
s(\mathsf{nval}(X,n)) & = & \left\{\sigma \in \Sigma^l : |\mathrm{ran}(\sigma|_X)| \leq n\right\} \\
s(\mathsf{atmost}(X,i,n)) & = & \left\{\sigma \in \Sigma^l : |\{x \in X : \sigma(x) = i\}| \leq n\right\}
\end{array}
$$

Figure 10: Semantics of FD Constraints

$$
\begin{aligned}
PS_E &: \{\alpha\} \to (\{\alpha\}, \{\{\alpha\}\}, \{\alpha\} \to \mathcal{R}) \\
PS_E(\Omega) &= (\Omega, \mathcal{P}(\Omega), P_E(\Omega)) \\
\\
P_E &: \{\alpha\} \to \{\alpha\} \to \mathcal{R} \\
P_E(\Omega)(\{\omega\}) &= \frac{1}{|\Omega|}
\end{aligned}
$$

Figure 11: Definition of PS_E

$$
\begin{aligned}
PS_\Sigma &: (\{\Sigma\}, \{\{\Sigma\}\}, \{\Sigma\} \to \mathcal{R}) \\
PS_\Sigma^l &= PS_E(\Omega_\Sigma) \\
\\
\Omega_\Sigma &: \{\Sigma\} \\
\Omega_\Sigma &= \varphi(\Sigma^{\mathsf{pl}})
\end{aligned}
$$

Figure 12: Definition of PS_Σ

D.2 Probability Spaces for Computing Looseness

Computing looseness is based on several probability spaces. The function PS_E (see figure 11) generates probability spaces with an even probability distribution. PS_Σ (see figure 12), which is based on PS_E, is a probability space which allows for reasoning about the probability of period-level instantiations. The probability space PS_C (see figure 13), which in turn is based on PS_Σ, provides a probability distribution which associates constraints with their looseness.

PS_E takes an arbitrary set of objects Ω and returns a probability space with Ω as its event space. $P_E(\Omega)$, the probability distribution of $PS_E(\Omega)$, is even, i.e.

$$\forall \omega, \omega' \in \Omega.\ P_E(\Omega)(\{\omega\}) = P_E(\Omega)(\{\omega'\}).$$

If Ω is finite, $PS_E(\Omega)$ is a discrete probability space, since the set of propositions of $PS_E(\Omega)$ is the power set of Ω. Therefore

$$\forall A \in \mathcal{P}(\Omega).\ P_E(A) = \sum_{\omega \in A} P_E(\{\omega\}) = \sum_{\omega \in A} \frac{1}{|\Omega|} = \frac{|A|}{|\Omega|}$$

and thus the definition of $P_E(\Omega)$ given naturally extends to arbitrary propositions from $\mathcal{P}(\Omega)$.

$$
\begin{array}{rcl}
PS_C & : & (\{\mathcal{C}\}, \{\{\mathcal{C}\}\}, \{\mathcal{C}\} \rightarrow \mathcal{R}) \\
PS_C & = & (\Omega_C, \mathcal{P}(\Omega_C), P_C) \\
 & & \\
\Omega_C & : & \{\mathcal{C}\} \\
\Omega_C & = & \{\mathsf{true}\} \cup C \\
 & & \text{where } C \text{ is the set of constraints from the} \\
 & & \text{current state of computation except for} \\
 & & \mathsf{in} \text{ and } \mathsf{levelLink} \text{ constraints} \\
 & & \\
P_C & : & \{\mathcal{C}\} \rightarrow \mathcal{R} \\
P_C(C) & = & P_\Sigma\left(\bigcup_{c \in C} \varphi(S(c))\right)
\end{array}
$$

Figure 13: Definition of PS_C

Ω_Σ, the event space of PS_Σ, is the set of instantiations mapping period-level variables onto the values allowed by the current state of computation. This definition of Ω_Σ is motivated by the insight that the proposition $A(c)$ (see above) can be modelled by the set

$$\boxed{\{\sigma : \sigma \text{ is an admissible period-level instantiation satisfying } c\}.}$$

Therefore, in order to faciliate the computation of looseness, Ω_Σ must contain admissible all period-level instantiations Since, by definition, $PS_\Sigma = PS_E(\Omega_\Sigma)$, the distribution of PS_Σ is even. This property is inherited from $PS_E(\Omega_\Sigma)$. Since Ω_Σ is finite, PS_Σ is discrete. P_Σ will be used as a shortcut for $P_E(\Omega_\Sigma)$.

PS_C provides the means for computing looseness. Since we are interested in the events of satisfying constraints, the event space of PS_C contains relevant constraints from the current state of computation, whereas each constraint symbolizes the event of its own satisfaction. P_C is the probability distribution of PS_C, whereas $P_C(c)$ is the probability of c's satisfaction, i.e. the looseness of c. [5] $P_C(c)$ is computed using the La Placian notion of probability, i.e.

$$\boxed{P_C(c) = \frac{|\text{set of admissible period-level instantiations satisfying } c|}{|\text{set of admissible period-level instantiations}|}}$$

by refering to P_Σ, i.e. by refering to PS_Σ. Note that this way of computing looseness

[5] Note that P_C takes a set of constraints. This is necessary because a probability distribution is intended to work on propositions, i.e. on sets of events. However, in this application we are interested in singleton propositions only.

assumes that instantiations are distributed evenly. The set c-satisfying period-level instantiations is obtained by refering to S.

Proposition D.6. *PS_C is a probability space.*

Proof. Since P_Σ is a probability distribution, the following propositions hold.

1. $\forall A \in \mathcal{P}(\Omega_C).\ P_C(A) \geq 0$
2. $\forall A, B \in \mathcal{P}(\Omega_C).\ P_C(A) + P_C(B) = P_\Sigma\left(\bigcup_{c \in A} \varphi(S(c))\right) + P_\Sigma\left(\bigcup_{c \in B} \varphi(S(c))\right)$ $= P_\Sigma\left(\bigcup_{c \in A \cup B} \varphi(S(c))\right) = P_C(A \cup B)$
3. $P_C(\Omega_C) = P_C(\{\mathsf{true}\} \cup C) = P_\Sigma\left(\varphi(S(\mathsf{true})) \cup \bigcup_{c \in C} \varphi(S(c))\right) = P_\Sigma(\Omega_\Sigma) = 1$

Therefore, P_C is a probability distribution, too. ☐

The following lemma gives some relations which are useful for computing $P_C(c)$.

Lemma D.7. *Suppose $l \in \mathsf{Level}$, c is an l-level constraint but not a conjunction. Then the following propositions hold.*

1. $\forall \sigma, \sigma' \in \Sigma^l.\ \sigma \neq \sigma' \rightarrow \pi(\sigma) \cap \pi(\sigma') = \emptyset$
2. $\forall A \subset \Sigma^l.\ \left|\bigcup_{\sigma \in A} \pi(\sigma)\right| = \sum_{\sigma \in A} |\pi(\sigma)|$
3. $|\Omega_\Sigma| = \prod_{x \in \mathcal{V}^{\mathsf{pl}}} |D_x|$
4. $\forall \sigma \in \varphi(\Sigma^{\mathsf{dl}}).\ |\pi(\sigma)| = \prod_{(u,i) \in \sigma} |D_{u\downarrow} \cap \mathrm{periods}(i)|$
5. $P_C(\{c\}) = \dfrac{\sum_{\sigma \in \varphi(s(c))} |\pi(\sigma)|}{|\Omega_\Sigma|}$

Proof.

1. Suppose $\sigma, \sigma' \in \Sigma^{\mathsf{pl}}$ and $\sigma \neq \sigma'$. Then, by the definition of π, $\pi(\sigma) \cap \pi(\sigma') = \{\sigma\} \cap \{\sigma'\} = \emptyset$. Now suppose $\sigma, \sigma' \in \Sigma^{\mathsf{dl}}$, $\sigma \neq \sigma'$, $\pi(\sigma)$ and $\pi(\sigma')$ are not disjoint and $\tau \in \pi(\sigma) \cap \pi(\sigma')$. Then, by the definition of π, $\forall x \in \mathrm{dom}(\tau).\ \mathrm{day}(\tau(x)) = \sigma(x\uparrow) = \sigma'(x\uparrow)$. Since $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma') = \{x\uparrow : x \in \mathrm{dom}(\tau)\}$, $\sigma = \sigma'$, which contradicts the assumption.
2. Follows immediately by application of the first proposition.

3. If $x \in \mathcal{V}^{\mathrm{pl}}$, i.e. if x is a period-level variable, there are $|D_x|$ possibilities to place the lesson associated with x.

4. If $u \in \mathcal{V}^{\mathrm{dl}}$, i.e. if u is a day-level variable, there are $|D_{u\downarrow} \cap \mathrm{periods}(i)|$ possibilities to place the lesson associated with u the day i.

5. $P_C(\{c\}) = P_\Sigma(\varphi(S(c))) = \dfrac{|\varphi(S(c))|}{|\Omega_\Sigma|} = \dfrac{\left|\varphi\left(\bigcup\limits_{\sigma \in s(c)} \pi(\sigma)\right)\right|}{|\Omega_\Sigma|} = \dfrac{\left|\bigcup\limits_{\sigma \in \varphi(s(c))} \pi(\sigma)\right|}{|\Omega_\Sigma|} = \dfrac{\sum\limits_{\sigma \in \varphi(s(c))} |\pi(\sigma)|}{|\Omega_\Sigma|}$ by the definitions of S, PS_C, PS_Σ and PS_E and by the second proposition of this lemma.

□

D.3 Aspects of Computability

Enumerating all elements of $S(c)$ is equivalent to enumerating all solutions to the CSP consisting of c. Since this problem is even harder than the CSP decision problem, which in turn is known to be NP-complete (cf. 4.1), a general algorithm capable of computing $S(c)$ within polynomial time for all c is not likely to exist. This implies that special methods have to be provided, which try to gain efficiency by exploiting the semantics of constraints. These methods may rely on domain specific knowledge and on conditions maintained by the propagation, which is being performed by the toplevel search process. Of course, computing $|S(c)|$ is sufficient for computing $P_C(\{c\})$, but computing $|S(c)|$ still means to solve the CSP decision problem.

Obviously, if c is a binary constraint, computing $S(c)$ is no problem since enumeration is tractable. If c is a day-level constraint, the relation given for π given by lemma D.7 can be exploited. If c is a global constraint, computing $S(c)$ is more involved. Clearly, checking each member of Ω_Σ is not tractable in general. Moreover, it is not obvious at all how to compute $|S(c)|$. Seemingly, the literature does not offer direct answers to this question. Consistency techniques, which gain efficiency by exploiting semantics, do not help either. Even establishing strong n-consistency (if c contrains n variables) does not make instantiations impossible, which do not satisfy the constraint. Therefore, multiplying the cardinalities of the pruned domains does not yield the number of solutions. A tree search would be necessary after establishing consistency. In general, this search would not be tractable. For example, consider the constraint $\mathsf{allDistinct}(X)$ with $D_x = \{1, \dots, |X|\}$ for all $x \in X$. Obviously, the underlying CSP is strongly $|X|$-consistent. Nevertheless, $|X|^{|X|}$ instantiations have to be considered.

D.4 Computing the Looseness of nval Constraints

By exploiting semantics, a method for computing the looseness of nval constraints has been devised. Consider the l-level constraint $\mathsf{nval}(X, n)$. We have to compute

$|S(\mathsf{nval}(X,n))|$, i.e. the number of admissible period-level instantiations satisfying $\mathsf{nval}(X,n)$. By definition, an l-level instantiation σ satisfies $\mathsf{nval}(X,n)$ if $|\mathrm{ran}(\sigma|_X)| \leq n$. Of course, in general, a brute force generate & test approach is destined to require exponential time. Consequently, the approach taken does not enumerate admissible period-level instantiations. Instead it identifies admissible l-level instantiations which have the same range when restricted to X. Identification yields a partition of the set $\varphi(\Sigma^l)$ of admissible l-level instantiations. Obviously, for each class C of this partition there is a unique set R such that $\forall \sigma \in C.\ \mathrm{ran}(\sigma|_X) = R$. R will be called the *range* of C and $|C|$ will be called the *frequency* of R. Furthermore, if $|R| \leq n$, all instantiations of C satisfy $\mathsf{nval}(X,n)$. In this case, R will be said to *satisfy* $\mathsf{nval}(X,n)$. Clearly, for computing $|S(\mathsf{nval}(X,n))|$, it is sufficient to sum up the frequencies of the ranges which occur and satisfy $\mathsf{nval}(X,n)$.

Example D.8. Consider the constraint $\mathsf{nval}(\{x,y,z\},2)$. Suppose $D_x = \{1,2,4\}$, $D_y = \{1,3,4\}$ and $D_z = \{1,4\}$, as depicted by the following table.

	1	2	3	4
x	·	·		·
y	·		·	·
z	·			·

Obviously, the ranges $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{1,2\}$, $\{1,3\}$, $\{1,4\}$, $\{2,3\}$, $\{2,4\}$ and $\{3,4\}$ satisfy $\mathsf{nval}(\{x,y,z\},2)$. However, the ranges $\{2\}$, $\{3\}$ and $\{2,3\}$ do not occur.

Computation of frequencies is a little bit involved. Consider the range $\{1,2\}$. The method described below starts at the number of possibilities to choose values for x, y and z from $\{1,2\}$. For x there are two possibilities to take a value within $\{1,2\}$. For y and z there is only one possibility, respectively. Thus, it seems that the frequency of $\{1,2\}$ is 2. But this count includes the instantiation σ with $\sigma(x) = \sigma(y) = \sigma(z) = 1$, i.e. $\mathrm{ran}(\sigma) = \{1\}$. Therefore, the frequency of $\{1,2\}$ is only 1. This shows that frequencies of sub-ordinate ranges have to be considered. The following table associates occuring ranges and frequencies.

range	frequency	range	frequency
$\{1\}$	1	$\{1,4\}$	6
$\{4\}$	1	$\{2,4\}$	1
$\{1,2\}$	1	$\{3,4\}$	1
$\{1,3\}$	1		

In total, there are 12 instantiations satisfying $\mathsf{nval}(\{x,y,z\},2)$. Since there are 18 instantiations allowed by the current state of computation, the looseness of $\mathsf{nval}(\{x,y,z\},2)$ is $\frac{12}{18} = \frac{2}{3}$. □

The following definition introduces the notions this section is based on.

Definition D.9. The definition applies if $l \in \mathsf{Level}$, $X \subset \mathcal{N}$ and $R \subset \mathcal{N}$.

$\bowtie$, defined by

$$\sigma \bowtie^l_X \sigma' \leftrightarrow \sigma, \sigma' \in \varphi(\Sigma^l) \wedge \mathrm{ran}(\sigma|_X) = \mathrm{ran}(\sigma'|_X),$$

is an equivalence relation which identifies instantiations which have the same range when restricted to X.

$\bar{R}^l_X$, defined by

$$\bar{R}^l_X = \left\{ \sigma \in \varphi(\Sigma^l) : \mathrm{ran}(\sigma|_X) = R \right\},$$

is the set of l-level instantiations with range R when restricted to X. Clearly, if $\bar{R}^l_X \neq \emptyset$, $\bar{R}^l_X \in \varphi(\Sigma^l)/\bowtie^l_X$.

$\hat{R}^l_X$, defined by

$$\hat{R}^l_X = \bigcup_{\sigma \in \bar{R}^l_X} \pi(\sigma),$$

is the period-level equivalent of $\bar{R}^l_X$. $|\hat{R}^l_X|$ will be called the *frequency* of R. □

The following lemmas prepare the grounds for computing $|S(\mathsf{nval}(X,n))|$. Lemma D.10 shows how to compute $|S(\mathsf{nval}(X,n))|$ by summing up the frequencies of occuring ranges, the identification of which is supported by lemma D.11. Lemma D.12 faciliates the computation of frequencies. Following this last lemma an outline of the algorithm for computing the looseness of nval constraints is given. The proofs follow at the end of the section.

Lemma D.10. *Suppose $l \in \mathsf{Level}$, $X \subset \mathcal{V}^l$, $n \in \mathcal{N}$ and $n < |X|$. Then*

$$|S(\mathsf{nval}(X,n))| = \sum_{\substack{\bar{R}^l_X \in \varphi(\Sigma^l)/\bowtie^l_X \\ |R| \leq n}} |\hat{R}^l_X|.$$

Lemma D.11. *Suppose $l \in \mathsf{Level}$ and $X \subset \mathcal{V}^l$. Then*

$$\left\{ R : \bar{R}^l_X \in \varphi(\Sigma^l)/\bowtie^l_X \right\} = \left\{ R \subset \bigcup_{x \in X} D_x : \begin{array}{ll} & \forall x \in X.\, D_x \cap R \neq \emptyset \\ \wedge & \exists \sigma \in \varphi(\Sigma^l).\, \mathrm{ran}(\sigma|_X) = R \end{array} \right\}.$$

Lemma D.12. *Suppose $l \in \mathsf{Level}$ and $X \subset \mathcal{V}^l$.*

Let $A = \bigcup_{\substack{\sigma \in \varphi(\Sigma^l) \\ \mathrm{ran}(\sigma|_X) \subset R}} \pi(\sigma)$ *and let* $B = \bigcup_{\substack{\sigma \in \varphi(\Sigma^l) \\ \mathrm{ran}(\sigma|_X) \subsetneq R}} \pi(\sigma)$.

Then

1. $|\hat{R}^l_X| = |A| - |B|$
2. $|A| = \begin{cases} \prod_{x \in V-X} |D_x| \prod_{x \in X} |D_x \cap R|, & \text{if } l = \mathsf{pl} \\ \prod_{x \in V-X} |D_{x\downarrow}| \prod_{x \in X} \left| D_{x\downarrow} \cap \bigcup_{i \in R} \mathrm{periods}(i) \right|, & \text{if } l = \mathsf{dl} \end{cases}$
3. $|B| = \sum_{\substack{\bar{S}^l_X \in \varphi(\Sigma^l)/\bowtie^l_X \\ S \subsetneq R}} |\hat{S}^l_X|$

Outline of the Algorithm for Computing the Looseness of nval *constraints.* Suppose $\mathsf{nval}(X,n)$ is an l-level constraint. For computing $|\mathsf{nval}(X,n)|$, lemma D.10 demands for the set of ranges R with $\bar{R}^l_X \in \varphi(\Sigma^l)/\bowtie^l_X$ and $|R| \leq n$. Lemma D.11 gives necessary and sufficient conditions for identifying the members of $\varphi(\Sigma^l)/\bowtie^l_X$. However, considerable overhead may be introduced by the search demanded by the condition $\exists\sigma \in \varphi(\Sigma^l).\ \mathrm{ran}(\sigma|_X) = R$. The number of searches necessary can be minimized by testing this condition only after the other conditions have been shown satisfyed. For computing the looseness of the day-level nval constraints appearing in the problem under study here (cf. B.11), searching for identifying relevant ranges has not been a problem. Lemma D.10 further demands for the frequencies of the ranges which have been rated relevant in the sense of lemma D.11. A method for computing the frequency of some range R is given by lemma D.12. This method involves recursion since it refers to the frequencies of relevant ranges S with $S \subsetneqq R$. Therefore, an efficient method for computing $|\mathsf{nval}(X,n)|$ has to address the danger of multiple computations. Fortunately, avoiding multiple computations is not difficult: First, all relevant ranges have to be determined. Second, the frequencies of relevant, singleton ranges have to be computed and recorded. Then, when computing the frequencies of relevant ranges with two elements, recursion can be replaced by a look-up in the record of frequencies, and so on. Thus, each frequency is computed only once.

Proof of Lemma D.10. By the definition of S, $S(\mathsf{nval}(X,n)) = \bigcup_{\sigma \in s(\mathsf{nval}(X,n),\varphi(\Sigma^l))} \pi(\sigma) = \bigcup_{\substack{\sigma\in\varphi(\Sigma^l)\\ |\mathrm{ran}(\sigma|_X)|\leq n}} \pi(\sigma)$.

Obviously, $\left\{\sigma \in \varphi(\Sigma^l) : |\mathrm{ran}(\sigma|_X)| \leq n\right\} = \bigcup_{\substack{\bar{R}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ |R|\leq n}} \bar{R}^l_X$ by the definition of $\bar{R}^l_X$.

Hence $S(\mathsf{nval}(X,n)) = \bigcup_{\substack{\bar{R}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ |R|\leq n\\ \sigma\in\bar{R}^l_X}} \pi(\sigma) = \bigcup_{\substack{\bar{R}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ |R|\leq n}} \bigcup_{\sigma\in\bar{R}^l_X} \pi(\sigma) = \bigcup_{\bar{R}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X} \hat{R}^l_X$ by the associativity and commutativity of $\cup$ and by the definition of $\hat{R}^l_X$.

By lemma D.7, $|S(\mathsf{nval}(X,n))| = \sum_{\substack{\bar{R}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ |R|\leq n}} |\hat{R}^l_X|$. □

Proof of Lemma D.11.

($\supset$) Suppose $\sigma \in \varphi(\Sigma^l)$ and $\mathrm{ran}(\sigma|_X) = R$. Then, by the definition of $\bar{R}^l_X$, $\sigma \in \bar{R}^l_X$. Thus, $\bar{R}^l_X \neq \emptyset$ and therefore $\bar{R}^l_X \in \varphi(\Sigma^l)/\bowtie^l_X$.

($\subset$) Suppose $\bar{R}^l_X \in \varphi(\Sigma^l)/\bowtie^l_X$. Then $\bar{R}^l_X \subset \varphi(\Sigma^l)$ and hence, by the definition of φ, $\forall\sigma \in \bar{R}^l_X.\ \forall x \in X.\ \sigma(x) \in D_x$. Suppose $\sigma \in \bar{R}^l_X$. Then $\mathrm{ran}(\sigma) \subset \bigcup_{x\in X} D_x$. Since $R = \mathrm{ran}(\sigma|_X) \subset \mathrm{ran}(\sigma)$, $R \subset \bigcup_{x\in X} D_x$. Now suppose $x \in X$. Since $\sigma(x) \in \mathrm{ran}(\sigma|_X) = R$ and $\sigma(x) \in D_x$, $R \cap D_x \neq \emptyset$. Furthermore, $\bar{R}^l_X \neq \emptyset$ because $\forall A \in \varphi(\Sigma^l)/\bowtie^l_X.\ A \neq \emptyset$ and thus, by the definition of $\bar{R}^l_X$, $\exists\sigma \in \varphi(\Sigma^l).\ \mathrm{ran}(\sigma|_X) = R$. □

Proof of Lemma D.12.

1. By the definition of $\hat{R}^l_X$, $\hat{R}^l_X = \bigcup_{\substack{\sigma\in\varphi(\Sigma^l)\\ \mathrm{ran}(\sigma|_X)=R}} \pi(\sigma)$ and hence $|\hat{R}^l_X| = |A-B|$.

   Furthermore, $|A-B| = |A|-|B|$ because $B \subset A$.

2. If $x \in V-X$, there is no restriction on the placement of the lesson associated with x. Hence there are $|D_x|$ possibilities to place the lesson, if x is a period-level variables, and $|D_{x\downarrow}|$ possibilities, if x is a day-level variable.

   If $x \in X$, the lesson associated with x is constrained to take place some period or day $i \in R$. Thus there are only $|D_x \cap R|$ possibilities to place the lesson, if x is a period-level variable, and $\left|D_{x\downarrow} \cap \bigcup_{i\in R} \mathrm{periods}(i)\right|$ possibilities, if x is a day-level variable.

3. Obviously, $\left\{\sigma \in \varphi(\Sigma^l) : \mathrm{ran}(\sigma|_X) \subsetneqq R\right\} = \bigcup_{\substack{\bar{S}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ S\subsetneqq R}} \bar{S}^l_X$ by the definition of $\bar{S}^l_X$.

   Hence $B = \bigcup_{\substack{\bar{S}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ S\subsetneqq R\\ \sigma\in\bar{S}^l_X}} \pi(\sigma) = \bigcup_{\substack{\bar{S}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ S\subsetneqq R}} \bigcup_{\sigma\in\bar{S}^l_X} \pi(\sigma) = \bigcup_{\substack{\bar{S}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ S\subsetneqq R}} \hat{S}^l_X$ by the definition of B, by the associativity and commutativity of $\cup$ and by the definition of $\hat{S}^l_X$.

   By lemma D.7, $|B| = \sum_{\substack{\bar{S}^l_X\in\varphi(\Sigma^l)/\bowtie^l_X\\ S\subsetneqq R}} |\hat{S}^l_X|$.

□

D.5 Approximating the Looseness of Global Constraints

Summarizing, it is not clear how to determine the exact value of looseness for global constraints. However, the approximation approach specified by the set of equations in figure 14 is tractable.

The looseness of $\mathsf{allDistinct}(X)$ is approximated by the looseness of a fully connected network of neq constraints. The number of neq constraints is $\sum_{k=1}^{n-1} = \frac{n^2-n}{2}$. Similiar, the looseness of $\mathsf{cumulative}(X,d,r,n)$ is approximated by the looseness of a conjunction of atmost constraints. This translation is possible because $d(x) = r(x) = 1$ for all $x \in X$ (cf. B.9).

$L(\mathsf{atmost}(X,i,n))$ is defined in terms of P_E. For $\mathsf{atmost}(X,i,n)$, the proposition of interest reads

> None of the variables gets assigned the value i or ... or m variables get assigned the value i, where m is the highest number such that the limit is not exceeded.

$$
\begin{array}{rcl}
L & : & \mathcal{C} \to \mathcal{R} \\
L(c) & = & P_C(c), \text{ if } c \text{ is a binary or an } \mathsf{nval} \text{ constraint} \\
L\left(\bigwedge C\right) & = & \prod_{c \in C} L(c) \\
L(\mathsf{allDistinct}(X)) & = & L\left(\bigwedge \{\mathsf{neq}(x,y) : x, y \in X \wedge x <_V y\}\right) \\
L(\mathsf{cumulative}(X,d,r,n)) & = & L\left(\bigwedge \left\{\mathsf{atmost}(X,i,n) : i \in \bigcup_{x \in X} D_x\right\}\right) \\
L(\mathsf{atmost}(X,i,n)) & = & P_E(\{k : 0 \le k \le l\})(\{k : 0 \le k \le m\}) \\
 & & \text{where} \\
 & & l = |\{x \in X : |D_x| > 1 \wedge i \in D_x\}| \\
 & & m = \min(l, n - |\{x \in X : D_x = \{i\}\}|)
\end{array}
$$

Figure 14: Definition of L

This proposition is formalized as a set of counts (cf. the second argument to P_E in the definition of $L(\mathsf{atmost}(X,i,n))$). Therefore, the event space must contain all counts which are possible under consideration of the current state of computation (cf. the first argument to P_E).

D.6 Independence Assumptions Underlying ρ

Proposition D.13. *The events of satisfying constraints are not independent.*

Proof.

1. $L(\bigwedge C) \neq L_C(\bigwedge C)$: Let $C = \{\mathsf{neq}(x,y), \mathsf{neq}(y,z)\}$ and let $D_x = D_y = D_z = \{1,2,3\}$. Then $L(C) = L_C(\mathsf{neq}(x,y))L_C(\mathsf{neq}(y,z)) = \left(\frac{18}{27}\right)^2 = \frac{4}{9} \neq \frac{6}{27} = L_C(\bigwedge C)$.
2. If C is a set of binary constraints,

$$L\left(\bigwedge C\right) = \prod_{c \in C} L(c) = \prod_{c \in C} P_\Sigma(\varphi(S(c)))$$

and

$$P_C\left(\bigwedge C\right) = P_\Sigma\left(\varphi\left(S\left(\bigwedge C\right)\right)\right) = P_\Sigma\left(\varphi\left(\bigcap_{c \in C} S(c)\right)\right) = P_\Sigma\left(\bigcap_{c \in C} \varphi(S(c))\right).$$

Since $L(\bigwedge C) \neq L_C(\bigwedge C)$, the events of satisfying constraints are not independent. □

E Curriculum and Assignment Matrix

Grade	Branch	K	Ev	Eth	D	E	F	L	M	Ph	C	B	G	Wr	Ek	Sk	Ku	Mu	Sm	Sw	Swp	Hw	Ha
5		2	2	2	5	6			4			2			2		2	3	2	2			2
6		2	2	2	5	6			4			2	2		2		3	2	2	2			
7		2	2	2	4	4	5	5	4			2	2		1		2	2	2	2			
8		2	2	2	4	4	4	4	4	2		1	2	1	2		1	1	2	2			
9	mn	2	2	2	3	3	3	3	4	2	4	2	2	1	1		1	1	2	2			
9	ml	2	2	2	3	3	5	3	3	1		2	2	1	1		1	1	2	2			
9	soc	2	2	2	3	3	3	3	3	1	2	2	2	1	1	2	1	1	2	2		1	
10	mn	2	2	2	3	3	3	3	4	3	3	2	2	1		1	1	1	2	2			
10	ml	2	2	2	3	3	5	3	3	2		2	2	1		1	1	1	2	2			
10	soc	2	2	2	3	3	3	3	3	2	2	2	2	1		2	1	1	2	2		4	
11	mn	2	2	2	4	4	4	4	5	3	3		2		2		2	1	2	2			
11	ml	2	2	2	4	4	5	4	3	2	2		2		2		1	1	2	2			
11	soc	2	2	2	4	4	4	4	3	2	2		2		2	3	1	1	2	2	2		
12	GK	2	2	2	4	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2
12	LK	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
13	GK	2	2	2	4	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2
13	LK	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Figure 15: The curriculum

Grade	Branch	K	Ev	Eth	D	E	F	L	M	Ph	C	B	G	Wr	Ek	Sk	Ku	Mu	Sm	Sw	Soz	Hw	Ha
5a		s7			g2	r5			r9			c4			t3		y4	m1	r9	s2			y6
5b		s7	s5		w9	g3			t4			c4			t3		y4	m1	s1	s2			y6
5c		s7		r0	r4	g5			t2			c2			r7		y4	m2	b3	s2			y6
5d		s6	s5	r0	w1	g1			r1			c1			t5		y4	m2	b3	s1			y6
6a		w3			r2	r0			t2			c5	r2		t5		y5	m1	b3	s2			
6b		s7		r0	s3	g1			t3			c5	r4		b3		y5	m1	c3	s1			
6c		s6	s5		y3	w2			t2			c3	w5		t5		y5	m2	c3	b3			
6d		s7	s5		w2	g2			r1			c2	w1		t5		y5	m2	b3	s2			
6e		s7	s5		w8	r7			t3			c2	r2		b3		y4	m1	b3	s2			
7a		s7			g4	r6	g5	y3	t1			c4	r0		t3		y4	m1	r9	s2			
7b		s6	s5		w2	g3	r6	y2	t1			c1	r0		t5		y4	m2	b3	s2			
7c		s6	s5	w8	w0	g2	r6	y3	r1			c1	w1		b3		y5	m2	b3	s1			
8a		w3	s5	y3	s6	r3		r8	t7	t2		c4	w7	b1	b1		y4	m1	r9	s1			
8b		s7			r2	w2	w6		t3	t7		c1	w5	b1	b1		y4	m2	r9	s1			
8c		w3	s5	y3	r8	g1	y2		t7	t9		c5	w7	b1	b1		y5	m2	r9	s1			
9a	mn	s6	s5	w8	w0	g3	w6	y2	t8	t6	c1	c3	r4	b1	r7		y5	m1	b3	s2			
9bn	ml	s6	s5	w8	r4	w2	r0	r8	t8	t6		c3	w7	b2	r7		y5	m1	r9	s1			
9bs	soc	s6	s5	w8	r4	w2		y2	t8	t6	c4	c3	w7	b2	r7	r4	y5	m1	r9	s1		y6	
9c	soc	w3	s5	w8	w4	w3	r6	y2	t9	t5	c4	c1	w5	b2	b3	r4	y4	m1	r9	s2		y6	
10a	mn	w3	s5	y3	w5	w3	g3	y2	t7	t6	c1	c1	r2	b2		r2	y4	m2	b3	s1			
10b	soc	s6	s5	y3	w5	w3	r3	y2	r9	r1	c5	c3	w5	b2		w5	y4	m2	b3	s1		y6	
10cn	ml	w3	s5	y3	g4	r3	r6	y3	r1	r1		c4	w7	b2		w7	y5	m2	b3	s2			
10cs	soc	w3	s5	y3	g4	r3	r3	y2	r1	r1	c5	c4	w5	b2		w5	y5	m2	b3	s2		y6	
11a	mn	s6	s5		r4	r7	g3	y2	t8	t8	c1		w5		b1		y4	m1	r9	s2			
11b	ml	s7	s5	w8	w7	r3	w6	r8	t9	t1	c4		w7		b1		y4	m1	r9	s2			
11c	soc	s6	s5	w8	g4	w3	g3	y2	t1	t5	c2		r2		b1	w1	y5	m1	r9	s2	w1		

Figure 16: The assignment matrix

References

[AB92] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. In *Actes des Journées Francophones de Programmation en Logique*, 1992.

[AM98a] Slim Abdennadher and Michael Marte. Constraintbasierte Stundenplanung für Universitäten. In *12. Workshop Planen und Konfigurieren*, 1998.

[AM98b] Slim Abdennadher and Michael Marte. University timetabling using constraint handling rules. In *Actes des Journées Francophones de Programmation en Logique et Programmation par Contraintes*, 1998.

[BDP96] Patrice Boizumault, Yan Delon, and Laurent Peridy. Constraint logic programming for examination timetabling. *Journal of Logic Programming*, 26(2):217–233, 1996.

[BFBW92] Alan Borning, Bjorn N. Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992.

[BMR95] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Constraint solving over semirings. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 624–630. Morgan Kaufmann, 1995.

[BR96] Christian Bessiere and Jean-Charles Régin. Mac and combined heuristics: Two reasons to forsake fc (and cbj?) on hard problems. In *Second International Conference on Principles and Practice of Constraint Programming*, LNCS 1118, pages 61–75. Springer, 1996.

[BV92] Maurice Bruynooghe and Raf Venken. Backtracking. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 84–88. Wiley, 1992. Volume 1, second edition.

[CL96] Yves Caseau and François Laburthe. Cumulative scheduling with task intervals. In Michael Maher, editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 363–377. MIT Press, 1996.

[COC97] Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In H. Glaser, P. Hartel, and H. Kucken, editors, *PLILP'97, International Symposium on Programming Languages: Implementations, Logics, and Programming*, number 1292 in LNCS, Southampton, September 1997. Springer.

[CVM97] Koen Claessen, Ton Vullinghs, and Erik Meijer. Structuring graphical paradigms in TkGofer. In *Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming*, pages 251–262, 1997.

[DM94] Rina Dechter and Itay Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.

[ECR95] ECRC Munich Germany. *ECL^iPS^e 3.5 User Manual*, 1995.

[FA97] Thom Frühwirth and Slim Abdennadher. *Constraint-Programmierung: Grundlagen und Anwendungen*. Springer, 1997.

[FHS95] Harikleia Frangouli, Vassilis Harmandas, and Panagiotis Stamatopoulos. UTSE: Construction of optimum timetables for university courses — A CLP based approach. In *Proceedings of the Third International Conference on the Practical Applications of Prolog*, pages 225–243, 1995.

[Frü95] Thom Frühwirth. Constraint handling rules. In Andreas Podelski, editor, *Constraint Programming: Basics and Trends*, LNCS 910. Springer, March 1995.

[FW92] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3):21–70, 1992.

[Gee92] Pieter A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 31–35. John Wiley & Sons, 1992.

[GMP+96] Ian P. Gent, Ewan MacIntyre, Patrick Prosser, Barabara M. Smith, and Toby Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In Eugene C. Freuder, editor, *Second International Conference on Principles and Practice of Constraint Programming*, pages 179–193. Springer, 1996.

[GMPW96] Ian P. Gent, Ewan MacIntyre, Patrick Prosser, and Toby Walsh. The constrainedness of search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 246–252. AAAI Press / MIT Press, 1996.

[HE80] Robert M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[HS97] Pascal Van Hentenryck and Vijay A. Saraswat. Constraint programming: Strategic directions. *Constraints Journal*, 2:7–33, 1997.

[HW95] Martin Henz and Jörg Würtz. Using Oz for college time tabling. In *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling*, pages 283–296, 1995.

[ILO98] ILOG SA. *ILOG Optimization Suite White Paper*, 1998.

[Int98] Intelligent Systems Laboratory, Swedish Institute of Computer Science. *SICStus Prolog User's Manual, release 3.7*, 1998.

[Jam96] Michael Jampel. A brief overview of over-constrained systems. In *Over-constrained systems*, volume 1106 of *LNCS*, pages 1–22. 1996.

[JFM96] Michael Jampel, Eugene C. Freuder, and Michael Maher, editors. *Over-constrained systems*, volume 1106 of *LNCS*. Springer, 1996.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *The Journal of Logic Programming*, 19 & 20:503–582, 1994.

[Kow79] Robert A. Kowalski. Algorithm = logic + control. *Communications of the ACM*, 22(7):424–436, July 1979.

[Kum92] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1), 1992.

[Llo94] John W. Lloyd. Practical advantages of declarative programming. In *Joint Conference on Declarative Programming*, 1994.

[Mac77] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mac92] Alan K. Mackworth. Constraint satisfaction. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 285–293. Wiley, 1992. Volume 1, second edition.

[Mey96] Harald Meyer auf'm Hofe. Representations of requirements through preference orderings of soft constraints. In *Notes on the DFKI '96 Workshop on Constraint-Based Problem Solving*, 1996.

[Mey97] Harald Meyer auf'm Hofe. ConPlan/SIEDAplan: Personnel assignment as a problem of hierarchical constraint satisfaction. In *Proceedings of the 3rd International Conference on the Practical Application of Constraint Technology*, pages 257–272, 1997.

[Nad88] Bernard A. Nadel. Tree search and arc consistency in constraint satisfaction algorithms. In Laveen N. Kanal and Vipin Kumar, editors, *Search in Artificial Intelligence*, pages 287–342. Springer, 1988.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[Pro93] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[Rég94] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 1, pages 362–367. AAAI Press / MIT Press, 1994.

[Sch95] Andrea Schaerf. A survey of automated timetabling. Technical Report CS-R9567, CWI - Centrum voor Wiskunde en Informatica, 1995.

[Sei83] Raimund Seidel. On the complexity of achieving K-consistency. Technical Report TR-83-04, Department of Computer Science, University of British Columbia, 1983.

[SF94] Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A. G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence*. John Wiley and Sons, 1994.

[SF97] Daniel Sabin and Eugene C. Freuder. Understanding and improving the mac algorithm. In *Third International Conference on Principles and Practice of Constraint Programming*, LNCS 1330, pages 167–181. Springer, 1997.

[SFV95] Thomas Schiex, Héléne Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–639. Morgan Kaufmann, 1995.

[Smo95] Gert Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer, 1995.

[SS94] Leon Sterling and Ehud Shapiro. *The Art of Prolog*. The MIT Press, Cambridge, Mass., second edition, 1994.

[Tho96] Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley, 1996.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[Wal96] Mark Wallace. Practical applications of constraint programming. *Constraints Journal*, 1:139–168, 1996.