Models and Algorithms for School Timetabling – A Constraint-Programming Approach

Dissertation

an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

von

Michael Marte

vorgelegt am 5. Juli 2002

Berichterstatter waren Prof. Dr. François Bry und Prof. Dr. Klaus U. Schulz.
Das Rigorosum fand am 15. Oktober 2002 statt.

Abstract

In ***constraint programming*** [JM94, Wal96, FA97, MS98], combinatorial problems are specified declaratively in terms of constraints. ***Constraints*** are relations over problem variables that define the space of solutions by specifying restrictions on the values that the variables may take simultaneously. To solve problems stated in terms of constraints, the constraint programmer typically combines chronological backtracking with ***constraint propagation*** that identifies infeasible value combinations and prunes the search space accordingly.

In recent years, constraint programming has emerged as a key technology for combinatorial optimization in industrial applications. In this success, global constraints have been playing a vital role. ***Global constraints*** [AB93] are carefully designed abstractions that, in a concise and natural way, allow to model problems that arise in different fields of application. For example, the `alldiff` constraint [Rég94] allows to state that variables must take pairwise distinct values; it has numerous applications in timetabling and scheduling.

In school timetabling, we are required to schedule a given set of meetings between students and teachers s.t. the resulting timetables are feasible and acceptable to all people involved. Since schools differ in their educational policies, the school-timetabling problem occurs in several variations. Nevertheless, a set of entities and constraints among them exist that are common to these variations. This common core still gives rise to NP-complete combinatorial problems.

In the first place, this thesis proposes to model the common core of school-timetabling problems by means of global constraints. The presentation continues with a series of operational enhancements to the resulting problem solver which are grounded on the ***track parallelization problem*** (TPP). A TPP is specified by a set of task sets which are called ***tracks***. The problem of solving a TPP consists in scheduling the tasks s.t. the tracks are processed in parallel. We show how to infer TPPs in school timetabling and we investigate two ways of TPP propagation: On the one hand, we utilize TPPs to down-size our models. On the other hand, we propagate TPPs to prune the search space. To this end, we introduce the `tpp` constraint along with a suitable constraint solver for modeling and solving TPPs in a finite-domain constraint programming framework.

To investigate our problem solvers' behavior, we performed a large-scale computational study. When designing the experiment, the top priority was to obtain results that are both reliable from a statistical point of view and practically relevant. To this end, the sample sizes have been chosen accordingly – for each school, our problem set contains 1000 problems – and the problems have been generated from detailed models of ten representative schools. Our timetabling engine essentially embeds network-flow techniques [Rég94, Rég96] and value sweep pruning [BC01] into chronological backtracking.

Acknowledgments

François Bry introduced me to logic programming and artificial intelligence and supervised this thesis.

Thom Frühwirth and Slim Abdennadher introduced me to constraint-logic programming and encouraged me to get involved in automated timetabling.

Klaus Schulz promptly agreed to review my work.

From 1999 to 2002, I participated in and was financed by the SIL graduate college (Graduiertenkolleg „Sprache, Information und Logik“) which in turn was financed by the DFG (Deutsche Forschungsgemeinschaft, German Research Council).

I am grateful to Mehmet Dincbas for his invitation to COSYTEC in Paris. The stay at Paris was supported by the BFHZ (Bayerisch-Französisches Hochschulzentrum – Centre de Coopération Universitaire Franco-Bavarois).

There were fruitful discussions on constraint programming and automated school timetabling with Abderrahmane Aggoun, Nicolas Beldiceanu, Helmut Simonis, Hana Rudova, and Elias Oliveira.

Norbert Eisinger and Ralph Matthes did some proof reading and made useful suggestions.

Mats Carlsson, the maintainer of SICStus Prolog, contributed with reliable technical support.

Tim Geisler and Sven Panne answered a lot of Haskell-related questions.

Martin Josko administered the Condor installation.

Ellen Lilge always was a helping hand in administrative issues.

Last but not least, this project is essentially based on the cooperation with the timetable makers Claus Schmalhofer, Erwin Appenzeller, Bärbel Kurtz, and Karl-Heinz Auktor.

I wish to express my gratitude to all the people and organizations I mentioned.

Contents

Chapter 1

Introduction

In school timetabling, we are required to schedule a given set of meetings between students and teachers s.t. the resulting timetables are feasible and acceptable to all people involved. Since schools differ in their educational policies, the school-timetabling problem occurs in several variations. Nevertheless, a set of entities and constraints among them exist that are common to these variations. This thesis presents, evaluates, and compares constraint-based solvers for this common core which still gives rise to NP-complete combinatorial problems [EIS76][1].

1.1 School Timetabling

Timetables are ubiquitous in daily life. For example, timetables substantially control the operation of educational institutions, health-care institutions, and public transport systems. Collins Concise Dictionary (4th edition) defines a ***timetable*** as a "table of events arranged according to the time when they take place". Timetables always have to meet domain-specific requirements.

Willemen [Wil02] defines ***educational timetabling*** as the "sub-class of timetabling for which the events take place at educational institutions". Events include lessons and lectures which occur in class-teacher timetabling and course timetabling, respectively. We refer to Schaerf [Sch95, Sch99] and Bardadym [Bar96] for well-known reviews of research in automated educational timetabling.

Educational systems differ from country to country and, within a given educational system, education differs from one level of the system to the other. In consequence, different views evolved on how to characterize the categories of educational timetabling. We adopt the view of Carter & Laporte [CL98] who consider class-teacher and course timetabling to occur at prototypical high schools

[1]See also [GJ79], problem SS19.

and universities, respectively, which they regard as opposite extreme points on the continuum of real-world educational institutions.

At the prototypical high school, timetabling is based on classes which result from grouping students with a common program. As only a few programs are offered, the number of classes is low. Timetables must not contain conflicts, need to be compact, and have to satisfy various other constraints on the spread and sequencing of lessons. Since each class has its own classroom, time-slot assignment and room allocation do not interact except for lessons that require science labs, craft rooms, or sports facilities. All rooms and facilities are situated in the same location and hence traveling times are negligible.

In contrast, the prototypical university allows for assembling individual programs. This freedom of choice may render individual timetables without conflicts impossible even though compactness is not an issue. To meet the students' choices as far as possible, timetable makers try to minimize the number of conflicts. Timetabling is complicated by room allocation and section assignment. Time-slot assignment interacts with room allocation because room sizes and requirements of equipment have to be taken into account. Moreover, since rooms and facilities are situated in different locations, traveling times need consideration. The problem of ***section assignment*** arises whenever a course has to be divided for practical reasons. As lectures of different sections may be scheduled independently, the way of sectioning affects timetabling.

In this thesis, solvers are studied on problems from German secondary schools of the *Gymnasium* type. This type of school offers nine grades of academically orientated education. Each school offers a specific set of programs and the higher the grade, the higher the number of programs. In the lower grades, a Gymnasium more or less resembles the prototypical high school. There are classes with fixed classrooms though, frequently, classes with students of different programs cannot be avoided. Subjects that are common to all programs occurring in a class are taught to the class as a whole. For education in program-specific subjects, classes are split and students from different classes are united. Student timetables must not contain conflicts and idle time. In the higher grades, a Gymnasium more or less resembles the prototypical university. Students with identical programs are rare, there are no classes, and section-assignment problems occur. However, student timetables must not contain conflicts and have to satisfy moderate compactness constraints. Whether time-slot assignment and room allocation for a specific school interact very much depends on its facilities. In the worst case, public facilities have to be shared with other schools and course lectures have to take place in classrooms when classes stay in science labs, craft rooms, or sports facilities. Last but not least, teacher timetables have to satisfy moderate compactness constraints. Drexl & Salewski [DS97] concluded that "the situation we are confronted with in secondary schools in Germany (...) seems to be the most general one which has ever been addressed in the open literature."

1.2 Constraint Programming

In ***constraint programming*** (CP) [JM94, Wal96, FA97, MS98], combinatorial problems are specified declaratively in terms of constraints. ***Constraints*** are relations over variables that define the space of solutions by specifying restrictions on the values that the variables may take simultaneously. To give an example relevant to timetabling, the so-called `alldiff` constraint [Rég94, vH01] takes a set of variables and requires the variables to take pairwise distinct values. As the process of expressing a given problem in terms of variables and constraints over them is widely perceived as a process of modeling, its outcome is called a ***constraint model*** in this thesis. Note that a constraint model does not contain any information on how to solve the problem it encodes. The term of ***declarative modeling*** refers to this absence of control information.

To solve a combinatorial problem stated in terms of constraints, the constraint programmer usually employs a search procedure based on chronological backtracking. Driven by branching strategies, ***chronological backtracking*** [BV92] unfolds the search space in a depth-first manner. Search nodes represent partial assignments to decision variables. If a partial assignment cannot be extended without violating constraints, chronological backtracking returns to the most recent search node that has a subtree left to expand and explores an alternative. The efficiency of chronological backtracking essentially depends on the quality of the branching strategies.

To reduce the search effort, chronological backtracking is combined with constraint propagation. From a given problem, ***constraint propagation*** infers primitive constraints like 'variable x must not take value a' which are used to prune the search space. Typically, constraint propagation and search are integrated tightly: Constraint propagation is invoked after every commitment, domain wipe-outs trigger immediate backtracking, and dynamic branching strategies take domain sizes into account. This way constraints take an active role in problem solving.

Typically, constraint propagation is performed by specialized software modules which are called ***constraint solvers***[2]. Constraint solvers exist independent of each other, they communicate over variable domains only, and when a constraint solver is applied to a constraint, it is not supplied any information except for the constraint itself and the state of its variables. This modular design facilitates a plug-and-play approach to problem solving, potentially leads to problem solvers that can be adapted to new requirements easily, and allows for embedding application-specific constraint solvers.

When designing a suitable problem solver, the constraint programmer has to

[2]Constraint solvers are sometimes called ***constraint propagators***. In fact, the terms *constraint solving* and *constraint propagation* mean the same and we will use both of them.

explore a multi-dimensional design space. A problem solver is determined by the choice of a search procedure, of branching strategies, of constraint solvers, and of how to express a given problem in terms of constraints. Due to the sheer number of combinations, only a small part of the design space can be explored. If standard procedures do not apply, expertise from the field of application is likely to be a good guide in this search. In his search, the designer has to address theoretical as well as operational issues. Theoretical issues include correctness[3] (soundness) and completeness[4]. Operational issues include running time, memory consumption, and reliability in terms of problem-solving capabilities. Completeness may be traded for efficiency and reliability: Frequently, sacrificing solutions by imposing additional constraints after careful consideration simplifies the search because, in effect, the search space is reduced. The clarification of operational issues usually requires empirical investigation.

In recent years, constraint programming has emerged as a key technology for combinatorial optimization in industrial applications. Three factors have been decisive: the introduction of global constraints, redundant modeling, and the availability of powerful constraint-programming tools (e.g. CHIP[5], ECL^{i}PS^e[6], SICStus Prolog[7], ILOG Solver[8], Mozart[9], CHR [FA97, Frü98]).

Global constraints [AB93] are carefully designed abstractions that, in a concise and natural way, allow to model combinatorial problems that arise in different fields of application. The `alldiff` constraint [Rég94, vH01] mentioned earlier is a prominent representative of global constraints; it has numerous applications in timetabling and scheduling. Global constraints are important because they allow for concise and elegant constraint models as well as for efficient and effective propagation based on rich semantic properties (e.g. [Nui94, Rég94, Rég96, RP97, BP97, PB98, Bap98, Pug98, FLM99, BPN99, MT00, DPPH00, BC01, BK01]).

Redundant modeling [CLW96] refers to the extension of constraint models with redundant constraints. A constraint is called ***redundant*** wrt. a given problem if the problem implies the constraint (i.e. every solution to the problem satisfies the constraint) but does not state it explicitely. Redundant modeling is well known for its potential operational benefits. It can be considered as a kind of constraint propagation performed prior to search that results in constraints that are themselves subject to propagation during search.

[3]A problem solver is called ***correct*** iff every assignment it outputs is a solution.

[4]A problem solver is called ***complete*** iff, in a finite number steps, it either generates a solution or proves that no solution exists.

[5]`http://www.cosytec.com`

[6]`http://www.icparc.ic.ac.uk`

[7]`http://www.sics.se`

[8]`http://www.ilog.com`

[9]`http://www.mozart-oz.org`

For detailed introductions to constraint programming, we refer to the textbooks by Tsang [Tsa93], Frühwirth & Abdennadher [FA97], and Marriott & Stuckey [MS98].

1.3 Outline of Thesis

Research in automated school timetabling can be traced back to the 1960s [Jun86]. During the last decade, efforts concentrated on local search methods such as simulated annealing, tabu search, and genetic algorithms (e.g. [Sch96a, DS97, CDM98, CP01]). Constraint-programming technology has been applied to university course timetabling (e.g. [GJBP96, HW96, GM99, AM00]) but its suitability for school timetabling has not yet been investigated.

In the first place, this thesis proposes to model the common core of school-timetabling problems by means of global constraints. The presentation continues with a series of operational enhancements to the resulting problem solver which are grounded on the ***track parallelization problem*** (TPP). A TPP is specified by a set of task sets which are called ***tracks***. The problem of solving a TPP consists in scheduling the tasks s.t. the tracks are processed in parallel. We show how to expose redundant TPPs in school timetabling and we investigate two ways of TPP propagation: On the one hand, we utilize TPPs to down-size our constraint models. On the other hand, we propagate TPPs to prune the search space. To this end, we introduce the `tpp` constraint along with a suitable constraint solver for modeling and solving TPPs in a finite-domain constraint-programming framework[10]. Moreover, we propose to reduce the search space by imposing non-redundant TPPs in a systematic way. In summary, this thesis emphasizes modeling aspects. Search procedures and branching strategies are not subject to systematic study.

To investigate our problem solvers' behavior, we performed a large-scale computational study. The top priority in experimental design was to obtain results that are practically relevant and that allow for statistical inference. To this end, the problems have been generated from detailed models of ten representative schools and the sample sizes have been chosen accordingly – for each school, our problem set contains 1000 problems. The school models describe the programs offered to the students, the facilities, the teacher population, the student population, and various requirements imposed by the school management. Our timetabling engine essentially embeds network-flow techniques [Rég94, Rég96] and value sweep pruning [BC01] into chronological backtracking. It comprises about 1500 lines of code and has been built on top of a finite-domain constraint system [COC97] which itself is part of the constraint-logic programming environment SICStus Prolog 3.9.0

[10]In ***finite-domain constraint programming***, we deal with ***finite-domain constraints*** defined over variables with finite integer domains which are called ***finite-domain variables***.

[Int02]. The experiment was performed on a Linux workstation cluster by means of the high-throughput computing software Condor [Con02].

To facilitate the computational study of problem solvers for school timetabling, a comprehensive simulation environment has been developed. This software package contains a problem generator, the school models, and various tools to visualize and analyze problems and solutions. The package has been implemented using the state-of-the-art functional programming language Haskell (e.g. [PJH99, Tho99]) and comprises about 10000 lines of code. The school models account for about 2700 lines of code.

This thesis is organized as follows. Chapter 2 introduces to school timetabling. Chapter 3 deals with track parallelization: it introduces the TPP and proposes the `tpp` constraint along with a suitable solver for modeling and solving TPPs in a finite-domain constraint-programming framework. Chapter 4 presents constraint-based solvers for school timetabling, shows how to infer and exploit TPPs in this setting, and presents our computational study of solver behavior. Chapter 5 summarizes, discusses the contribution of this thesis wrt. timetabling in general, and closes with perspectives for future work. Appendix A presents a new method for proving confluence in a modular way and applies this method to the TPP solver.

1.4 Publications

The work presented in this thesis was outlined at the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT-2000) [Mar00]. The `tpp` constraint and its solver (cf. Chapter 3) were presented at the 5th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP-2001) [Mar01a] and at the Doctoral Programme of the 7th International Conference on Principles and Practice of Constraint Programming (CP-2001) [Mar01b]. The method for proving confluence in a modular way (cf. Appendix A) was presented at the 4th International Workshop on Frontiers of Combining Systems (FroCoS-2002) [Mar02].

1.5 Reduction Systems

A ***reduction system*** is a pair $(A, \rightarrow)$ where the ***reduction*** $\rightarrow$ is a binary relation on the set A, i.e. $\rightarrow \subseteq A \times A$. Reduction systems will be used throughout this thesis to describe inference algorithms including constraint solvers. In such a context, the elements of A will be states, the elements of $\rightarrow$ will be transitions that correspond to steps of computation, and a computation will proceed step by step until no more transition applies.

Describing algorithms with reduction systems has some advantages over using procedural programs:

- Reduction systems are more expressive than procedural programs because non-determinism can be modeled explicitely. So we are not required to commit to a specific execution order in case there are alternatives.

- Freed from the need to fix an execution order, we can concentrate on the essential algorithmic ideas. There is no need to worry about operational issues like which execution order will perform best in some sense or another.

- If we give a non-deterministic reduction system, every result about its computations (like correctness and termination) will apply a fortiori to every of its deterministic implementations.

- We can study the effects of non-deterministic choices: Is there a uniquely determined result for every input? If this is the case for a given reduction system, then its deterministic implementations will have one and the same functionality.

The theory of reduction systems has many results to answer questions on termination and on the effects of non-deterministic choices (for example, see Newman's Lemma below).

When talking about reductions systems, we will make use of the concepts and the notation introduced in the following (cf. [BN98]):

Definition 1.1. Let $(A, \rightarrow)$ be a reduction system.

- It is common practice to write $a \rightarrow b$ instead of $(a, b) \in \rightarrow$.

- $\rightarrow^=$, $\rightarrow^+$, and $\rightarrow^*$ denote the reflexive closure, the transitive closure, and the reflexive, transitive closure of $\rightarrow$, respectively.

- We say that y is a ***direct successor*** to x iff $x \rightarrow y$.

  We say that y is a ***successor*** to x iff $x \rightarrow^+ y$.

- x is called ***reducible*** iff $\exists y.\ x \rightarrow y$.

  x is called ***in normal form (irreducible)*** iff it is not reducible.

  y is called ***a normal form of*** x iff $x \rightarrow^* y$ and y is in normal form.

  If x has a uniquely determined normal form, the latter is denoted by $x{\downarrow}$.

- $x, y \in A$ are called ***joinable*** iff $\exists z.\ x \rightarrow^* z \leftarrow^* y$, in which case we write $x \downarrow y$.

  $\rightarrow$ is called ***locally confluent*** iff $y \leftarrow x \rightarrow z$ implies $y \downarrow z$.

  It is called ***confluent*** iff $y \leftarrow^* x \rightarrow^* z$ implies $y \downarrow z$.

- $\rightarrow$ is called ***terminating*** iff there is no infinite, descending chain $a_0 \rightarrow a_1 \rightarrow \cdots$ of transitions.

- $\rightarrow$ is called ***convergent*** iff it is terminating and confluent.

The following Lemma follows easily from the definition of convergence.

Lemma 1.1. *Let $(A, \rightarrow)$ be a reduction system. If $\rightarrow$ is convergent, then each $a \in A$ has a uniquely determined normal form.*

This Lemma may free us from worrying about whether the non-determinism that is inherent in a certain reduction system allows for computations that produce different results for the same input. With a convergent reduction system, whatever path of computation is actually taken for a given input, the result is uniquely determined. In other words, if $\rightarrow$ is a convergent reduction on some set A, then the reduction $\{(a, a\downarrow) : a \in A\}$ is a function on A.

The following statement provides a tool for proving confluence via local confluence; it is called ***Newman's Lemma***. See [BN98] for a proof.

Lemma 1.2. *A terminating reduction is confluent if it is locally confluent.*

1.6 Constraint Satisfaction

To faciliate the application of constraint technology to a school-timetabling problem, we have to translate the high-level problem description (in terms of teachers, students, facilities, and meetings) into a so-called ***constraint satisfaction problem*** (CSP). A CSP is stated in terms of constraints over variables with domains and to find a solution requires to choose a value from each variable's domain s.t. the resulting value assignment satisfies all constraints.

The usual way to describe a CSP is to give a triple (X, δ, C) with a set of variables X, a set of constraints C over X, and a total function δ on X that associates each variable with its domain. Another way to specify a CSP is to give a hypergraph with variables as nodes and constraints as hyperarcs. Such a graph is called a ***constraint network***.

When talking abount constraint satisfaction and constraint solving, we will use the concepts and the notation introduced in the following:

Definition 1.2. Let $P = (X, \delta, C)$ be a CSP.

- δ is called a ***value assignment*** iff $|\delta(x)| = 1$ for all $x \in X$.

- If δ is a value assignment and $x \in X$, then $x\delta$ denotes the single element of $\delta(x)$.

- P is called ***ground*** iff δ is a value assignment.
- P is called ***failed*** iff some $x \in X$ exists with $\delta(x) = \emptyset$.
- A domain function σ on X is called a ***solution*** to P iff σ is a value assignment, $x\sigma \in \delta(x)$ for all $x \in X$, and σ satisfies all constraints $c \in C$.
- $\mathrm{sol}(P)$ denotes the set of solutions to P.
- If Q is a CSP, too, then P and Q are called ***equivalent*** iff $\mathrm{sol}(P) = \mathrm{sol}(Q)$, in which case we write $P \equiv Q$.
- If D is a set of constraints over X, then $P \oplus D$ denotes $(X, \delta, C \cup D)$.
- A constraint c over X is called ***redundant*** wrt. P iff $c \notin C$ and $\mathrm{sol}(P) \subseteq \mathrm{sol}(P \oplus \{c\})$.

So a constraint is called redundant wrt. a given problem if every solution to the problem satisfies the constraint though the problem does not state the constraint explicitely.

The following statement about adding constraints follows easily from the properties of solutions:

Lemma 1.3. *Let P be a CSP with variables X and let C be a set of constraints over X. $P \equiv P \oplus C$ holds iff $P \equiv P \oplus \{c\}$ for all $c \in C$.*

We will consider finite CSPs with integer domains only and use the abbreviation FCSP to denote such problems. If a and b are integers, we will write $[a,b]$ to denote the set of integers greater than or equal to a and smaller than or equal to b, as usual.

1.7 Constraint Solving

In this thesis, constraint solvers will be described through non-deterministic reduction systems. As explained in Section 1.5, this way of specifying constraint solvers has a number of advantages over giving procedural programs: We may concentrate on the essential algorithmic ideas without bothering about operational issues, our results about computations will apply to every deterministic implementation, and there is a variety of technical means to study questions on confluence.

To provide a framework for defining constraint solvers, our next step is to define the FCSP transformer $\rightarrow_{\mathrm{FD}}$ as

$$(X_0, \delta_0, C_0) \rightarrow_{\mathrm{FD}} (X_1, \delta_1, C_1)$$

iff $X_1 = X_0$, $C_1 = C_0$, $\delta_1 \neq \delta_0$, and $\delta_1(x) \subseteq \delta_0(x)$ for all $x \in X_0$.

The resulting reduction system captures three important aspects of finite-domain constraint solving as we find it implemented in today's constraint-programming systems: Constraint solvers reduce the search space by domain reductions, they cooperate by communicating their conclusions, and this process will not stop until no transition applies. Further aspects of constraint solving like correctness will be considered later on.

The possibility of adding variables and constraints has not been provided for because we will not consider computations where constraint solvers add new constraints except for such that can be expressed in terms of domain reductions.

It is easy to see that $\rightarrow_{\mathrm{FD}}$ is transitive and convergent. Because of transitivity, it is not necessary to distinguish successors, that are reached by one or more reduction steps, from direct successors.

The following monotonicity properties of domain bounds and solutions sets are immediate:

Corollary 1.1. *Let P_0 and P_1 be FCSPs with $P_0 \rightarrow_{\mathrm{FD}} P_1$.*

1. *Lower (Upper) bounds of domains grow (shrink) monotonically, i.e.*

$$\min \delta_0(x) \leq \min \delta_1(x) \leq \max \delta_1(x) \leq \max \delta_0(x)$$

   for all $x \in X$ where δ_0 and δ_1 denote the domain functions of P_0 and P_1, respectively.

2. *Solution sets shrink monotonically, i.e.* $\mathrm{sol}(P_0) \supseteq \mathrm{sol}(P_1)$.

We continue by considering those subsets of $\rightarrow_{\mathrm{FD}}$ that preserve solutions: We define the reduction $\rightarrow_{\mathrm{C}}$ as

$$\{(P_0, P_1) \in \rightarrow_{\mathrm{FD}} : P_0 \equiv P_1\}$$

and call a subset of $\rightarrow_{\mathrm{FD}}$ ***correct*** iff it is contained in $\rightarrow_{\mathrm{C}}$. Clearly, $\rightarrow_{\mathrm{C}}$ is correct itself and every correct reduction is terminating.

If a correct reduction reliably detects variable assignments that violate a given constraint, then we call the reduction a solver for this particular constraint:

Definition 1.3. Let c be a constraint and let $\rightarrow \subseteq \rightarrow_{\mathrm{FD}}$. $\rightarrow$ is called a ***solver*** for c iff $\rightarrow$ is correct and satisfies the following requirement: If $P = (X, \delta, C)$ is a ground, unfailed FCSP with $c \in C$ and δ violates c, then every normal form of P wrt. $\rightarrow$ is failed.

The requirement of correctness restricts the scope of the preceding definition to those subsets of $\rightarrow_{\mathrm{FD}}$ that perform deduction by adding redundant constraints in terms of domain reductions. Returning a failed problem signals the embedding search procedure[11] that a constraint violation has occured, causes it to reject the variable assignment, and to look for another one.

By requiring that violated constraints do not go unnoticed, we can be sure that any variable assignment returned by any embedding search procedure is a solution. Nevertheless, one might object that the definition is not quite complete because it does not impose any bound on time complexity though a decent constraint solver is usually required to run in polynomial time[12]. However, such a definition would rule out generic constraint solvers with exponential worst case complexity that apply to any constraint but are not practical unless the constraint to be propagated has only a few variables with small domains. Such a constraint solver (e.g. [BR99]) is useful if a specialized constraint solver is not available and developing one does not pay off because the run times exhibited by the generic constraint solver are reasonable.

The preceding definition might be objected to for another reason: It allows for problems solvers that implement a pure generate & test approach. This contradicts the spirit of constraint programming whereby constraints should take an active role in the problem-solving process as early as possible because search becomes easier the more values are pruned and the earlier this happens[13]. The definition does not make any requirement in this direction because what kind of reasoning should be performed really depends on the application. If there is any need to specify the reasoning power of a constraint solver, there are established ways to do this like asserting the maintenance of interval or domain consistency [VSD98].

Note that there are two ways to employ reductions that are correct but that do not solve a constraint or a class of constraints in the sense of the preceding definition. Such reductions can be used to propagate redundant constraints or to provide additional pruning.

It remains to clarify how to combine constraint solvers defined as reductions. If $\rightarrow_1, \ldots, \rightarrow_n$ are constraint solvers, then there are two immediate ways to combine

[11]Constraint solvers on their own are not able to solve problems. A given problem may have many solutions and committing to one of them is beyond deduction. So we find constraint solvers embedded into search procedures like chronological backtracking.

[12]With regard to the definition of constraint solvers in terms of reductions, to run in polynomial time means that every path of computation ending in a normal form can be traversed in polynomial time. This is not possible unless every transition can be carried out in polynomial time.

[13]Of course, the overall efficiency of a problem solver depends on the nature of the problems it is applied to and on the time complexity of the propagation algorithms that are used. However, as experience has shown for many applications, it is more efficient to spend less time on search than on constraint propagation (e.g. [SF94, BR96a, SW99]).

them: Either we use their union or we use

$$\bigcup_{1 \leq i \leq n} \{(P,Q) : P \text{ is a FCSP, } Q \neq P, \text{ and } Q \text{ is a normal form of } P \text{ wrt. } \rightarrow_i\}.$$

The latter method comes closer to what actually happens in today's constraint-programming systems: When a variable domain is reduced (either by constraint propagation or by the search procedure committing to a choice[14]), then the constraints containing the variable are determined and the corresponding constraint solvers are scheduled for invocation. When a constraint solver is eventually applied to a constraint, then the former is expected to reduce the search space instantaneously and as far as it is able to do so. This way the constraint does not require further consideration until it has its wake-up conditions satisfied again.

[14] The alternatives arise from partitioning the variable's domain in some way.

Chapter 2

School Timetabling

Besides NP-completeness, there is another fundamental issue in school timetabling, namely the diversity in problem settings. Even schools of the same type may differ with regard to the requirements of their timetables. In face of this diversity, which constraints does a timetabling algorithm need to handle to be widely applicable? It is clear that a timetabling algorithm will not be very useful unless it can handle the frequent constraints. But which constraints occur frequently? This question is studied in Section 2.1 which, at the same time, introduces the reader to school timetabling.

Section 2.2 continues the presentation with introducing a special class of schools – namely German secondary schools of the *Gymnasium* type. Among other things, their timetabling problems are characterized and the process of manual timetabling is explained. The treatment is very detailed to enable the reader to assess the scope of the results presented in Chapter 4 where the operational properties of constraint-based solvers for school timetabling are studied on problems from schools of the *Gymnasium* type.

Research in automated school timetabling can be traced back to the 1960s and, during the last decade, efforts concentrated on greedy algorithms and on local search methods. Several case studies have been performed and some promising results have been obtained. Section 2.3 closes the chapter with a review of these more recent approaches.

2.1 The Core of School-Timetabling Problems

In school timetabling, we are required to schedule a given set of meetings s.t. the resulting timetables are feasible and acceptable to all people involved. Since schools differ in their educational policies, the school-timetabling problem occurs in various variations. Nevertheless, a set of entities (namely students, teachers,

	Mon	Tue	Wed	Thu	Fri
1					
2					
3					
4					
5					
6					

Figure 2.1: A typical time grid with five days and six time slots a day

rooms, and meetings of fixed duration) and constraints among them exist that are common to these variations. This section details on the core's composition from the author's point of view (c.f. Figure 2.2) which is based on a survey of more recent publications [Cos94, Sch96b, Sch96a, DS97, CDM98, FCMR99, KYN99, BK01, CP01], on a survey of commercial timetabling software[1] (gp-Untis 2001[2], Lantiv Timetabler 5.0[3], OROLOGIO 11[4], SPM 3.1[5], TABULEX 2002 (6.0)[6], Turbo-Planer 2000[7]), and on the author's own field work.

Meetings are regular events of fixed duration including lessons, consulting hours, and teacher meetings. A ***meeting*** is specified in terms of a student set (empty in case of consulting hours and teacher meetings), a non-empty teacher set (usually singleton in case of a lesson), and a non-empty set of suitable rooms. It is understood that a meeting has to be attended by every of its teachers and students and that is has to be held in one of the designated rooms.

We assume that the meetings of a problem are packed into jobs. A ***job*** is a set of meetings and we require the jobs of a problem to partition its meeting set. All the meetings of a job have to be identical with regard to their resource requirements. Typically, a job comprises all the lessons of some class in some subject.

Timetabling is based on a rectangular time grid that divides the planning period into disjoint time intervals of equal duration which are called ***time slots*** or simply ***periods***[8]. Figure 2.1 shows a typical time grid with five days and six time

[1] This survey is based on data sheets, manuals, tutorials, on-line help files, and demo versions that were available from the Internet.

[2] Gruber & Petters Software, Austria, `http://www.grupet.at`

[3] `http://www.lantiv.com`

[4] ANTINOOS Software Applications, Greece, `http://www.antinoos.gr`

[5] `http://www.stundenplanmanager.de`

[6] ZI_SOFT_KIEL, Germany, `http://www.ziso.de`

[7] Haneke Software, Germany, `http://www.haneke.de`

[8] Actually, the term *period* has at least three different uses in school timetabling: (a) In phrases like "the first period on Monday", we refer to a single time slot. (b) If we require that a meeting has to take place "in the first period", we mean that it must either take place in the first period on Monday, or in the first period on Tuesday, and so on. (c) In phrases like "for two periods a week",

slots a day. A time slot can be addressed by its day and its index (contained in the left-most column of the time grid displayed in Figure 2.1). The amount of time represented by a time slot usually equals the amount of time required to give a lesson though we do not impose any restriction on the granularity of time. Meeting durations and start times are specified in terms of time slots.

As start times and rooms are not fixed in advance of planning (the exception proves the rule), the process of timetabling involves scheduling and resource allocation with the aim to create a timetable for each student and each teacher. As a by-product, timetables for each room are obtained. The space of solutions is restricted by the following constraints:

1. Resource capacities: Resources (students, teachers, and rooms) must not be double-booked.

2. Restrictions on availability: Resources may be unavailable for given periods of time[9]. Such restrictions are expressed through sets of time slots (so-called ***time frames***). Periods of unavailability will also be referred to as ***down times***.

3. Couplings: A ***coupling*** requires to schedule a given set of jobs in parallel[10].

4. Distribution constraints (c.f. Table 2.1):

   (a) Lower and upper bounds on daily work load: This type of constraint occurs for students, teachers, student-teacher pairs[11], jobs[12], and pairs of students and job sets[13]. It does not occur for rooms. Work load is measured in time slots.

   (b) Lower and upper bounds on the number of working days: This type of constraint occurs only for students and teachers.

---

we specify a certain amount of time; acting as a unit of measure, the term *period* refers to the duration of time slots.

[9]Restrictions on availability may result from the sharing of teachers and facilities among several schools.

[10]Couplings are essentially a consequence of study options (cf. Section 2.2 and Section 4.3). Timetabling may be complicated considerably by couplings due to simultaneous resource demands. As a rule of thumb, the more study options are offered the more couplings will give a headache to the timetabler.

[11]This way, for each student and for each of its teachers, the daily amount of time spent together in class may be bounded.

[12]This way job-specific bounds on the daily amount of teaching may be imposed.

[13]This way, for each student, the daily amount of time main subjects are taught to the student (or any other subset of its subjects) may be bounded.

	Bounds on daily work load	Bounds on the number of working days	Bounds on total idle time
gp-Untis 2001	+/+	+/+	+/+
Lantiv Timetabler 5.0	+/+	-/-	+/+
OROLOGIO 11	-/+	-/-	+/+
SPM 3.1	+/+	-/+	+/+
TABULEX 2002 (6.0)	+/+	+/-	-/+
Turbo-Planer 2000	+/+	-/+	+/-
[Cos94]	-/-	-/-	-/+
[Sch96b, Sch96a]	+/+	-/+	-/+
[DS97]	-/+	-/-	-/-
[CDM98]	-/-	-/-	-/-
[FCMR99]	-/+	-/-	-/+
[KYN99]	-/+	-/-	-/-
[BK01]	+/+	-/-	-/-
[CP01]	-/+	-/+	-/+
Author's field work	+/+	-/+	+/+

Table 2.1:
Survey of the most frequent distribution constraints

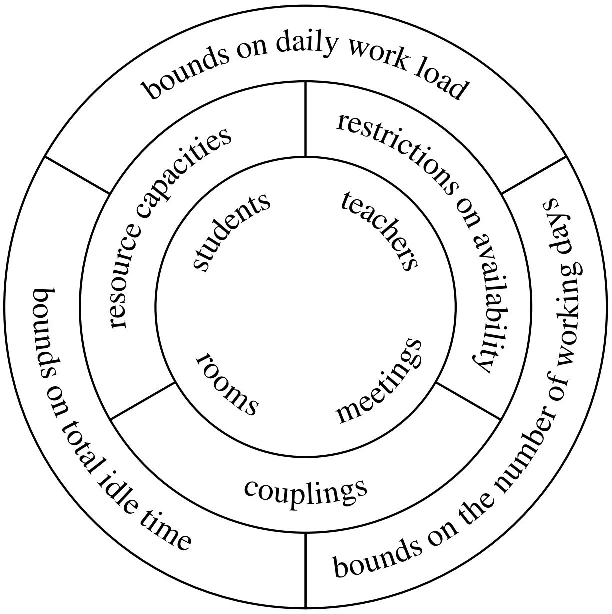


Figure 2.2: The core of school-timetabling problems

(c) Lower and upper bounds on total idle time: Timetables may contain time slots that are not occupied by any meeting. Such a time slot in the middle of a day is called ***idle***, a ***gap***, or a ***hole*** (in German ***Fensterstunde***, ***Freistunde***, ***Hohlstunde***, or ***Springstunde***), if there are meetings scheduled before and after it. The number of idle time slots that occur in a personal timetable is referred to as the person's ***total***[14] ***idle time***. (Bounds on total idle time occur only for students and teachers.)[15]

Lower bounds on daily work load and on total idle time have in common that they do not apply to free days. For example, consider a teacher with bounds l and u on her daily work load. For each day of the planning period, the teacher's timetable has to satisfy the following condition: Either the day is free or at least l and at most u time slots are occupied by meetings of the teacher.

The list actually comprises most constraints on student, teacher, and room timetables that occur in practice. Yet school-timetabling problems differ considerably regarding those distribution constraints that directly affect job timetables. These constraints are therefore not included in the common core except for bounds on the number of time slots that may be occupied a day. (Such bounds occur frequently.) Despite this restriction, the common core gives rise to NP-complete combinatorial problems [EIS76][16].

In practice, hard constraints are distinguished from soft constraints. ***Hard constraints*** are conditions that must be satisfied whereas ***soft constraints*** may be violated but should be satisfied as much as possible. By their very nature, resource capacities and restrictions on resource availability occur only as hard constraints. The author's study showed that couplings occur only as hard constraints and that distribution constraints occur mainly as soft constraints.[17]

2.2 The German *Gymnasium*

In this thesis, the operational properties of constraint-based solvers for school timetabling are studied on problems from German secondary schools of the ***Gymnasium*** type. This type of school offers nine grades of academically orientated

[14]Rarely there are bounds on daily idle time.

[15]The publications that were subject to review talk about the need to minimize the number of idle time slots but do not give any bounds. A close look into the models revealed that idle time slots are not allowed at all and hence that the upper bounds equal zero.

[16]See also [GJ79], problem SS19.

[17]With the commercial timetabling packages, the definition of a problem is carried out in two steps. To start with, the timetabling data is entered into the respective master-data module except for the constraint priorities. The constraint priorities are specified only immediately before timetabling as far as the respective planning module allows for user adjustments.

education (numbered from five through thirteen). This section characterizes the German Gymnasium and its timetabling problems and it describes the processes of problem specification and manual timetabling.

2.2.1 Educational Programs

Each school offers a specific set of educational programs and the higher the grade, the higher the number of programs. Foreign-language education starts right in the fifth grade and, usually, English is taught as first foreign-language but there may be options. Then the number of study options increases in three steps:

1. From the seventh grade, a second foreign language is introduced and there may be school-specific options like, for example, either French or Latin.

2. From the ninth grade, education may diversify further because several study directions may be offered. Official regulations [App98] define six study directions: Humanities, Modern Languages, Music and Arts, Natural Sciences, Economics, and Social Sciences. In Humanities and Modern Languages, a third foreign language is introduced and, again, there may be school-specific options.

3. For the grades twelve and thirteen (in German ***Kollegstufe***), the study directions of the previous grades are discontinued in favor of a system where each study direction is based on a pair of ***advanced-level courses*** (in German ***Leistungskurse***). Official regulations [Kol97] (updated yearly) define which base pairs are admissible (140 base pairs were admissible in 1997) and, for each base pair, define constraints on the individual choice of ***foundation courses*** (in German ***Grundkurse***). In practice, about ten to twenty study directions are implemented. The choice of study directions varies from school to school and from year to year. It is determined by what the school management offers and by what the students demand for.

The choice of a study direction does not always determine a program completely because many study directions, in particular those of the final grades, may offer options. Hence, to fix their programs, students may have to meet further decisions such as the students of Modern Languages who may have to choose a third foreign language from a given set of alternatives.

For two reasons, the actual number of programs may be up to six times higher than the number of programs resulting from the study options. On the one hand, the sexes are segregated in physical education. It follows that a boy and a girl have different programs even if their choice of study options is the same. On the other hand, students are segregated in religious education according to their religious denominations. Catholic and Protestant students receive specific religious education[18] while members of other denominations have to undergo ethical education. Consequently, if two students differ in religious denomination, they have different programs unless both of them receive ethical education.

From the fifth through the eleventh grade, a Gymnasium more or less resembles the prototypical high school (c.f. Section 1.1). There are classes with fixed classrooms though, frequently, heterogeneous classes with students of different programs (boys and girls, Catholics and Protestants, and students of different study directions) cannot be avoided. Subjects that are common to all programs occurring in a class are taught to the class as a whole. For education in program-specific subjects, classes are split and students from different classes are united. In combination with tight time frames (except for the eleventh grade, student timetables usually must not contain idle time), this procedure leads to requests for parallel education of several groups that complicate timetabling considerably due to simultaneous resource demands (cf. Section 4.3).

In the final grades, a Gymnasium more or less resembles the prototypical university. Students with identical programs are rare, there are no classes, and the student timetables have to satisfy moderate compactness constraints.

2.2.2 The Problem Specification Process

The specification of the next school year's timetabling problem lies in the hands of the headmaster. When specifying the problem, the headmaster has to keep to official regulations [App98] that for each grade, study direction, and subject specify the subject matter and the weekly number of teaching periods. In particular, those regulations restrict the possibilities for joint education of students from different study directions.

The specification process results in a set of so-called modules. A ***module*** requires to split a given set of students into a given number of groups of similar size and to educate each group for a given number of periods in a given subject. To this end, each module is assigned a set of suitable rooms and each of its groups is assigned a suitable teacher. We distinguish intra-class from inter-class modules. The students of an ***intra-class module*** all stem from the same class while an ***inter-class module*** joins students from at least two classes. To translate a module into a set of jobs (c.f. Section 2.1), we need to fix the group members and the meeting durations.

[18] Parents of a Catholic or Protestant student may require that their child undergoes ethical education instead of specific religious education.

2.2.3 The Effects of Study Options

To illustrate the effects of study options on timetabling data, two examples will be given. But first it is necessary to describe the school that the examples stem from: In the fifth grade, all students start to learn English. From the seventh grade, the foreign-language curriculum has to be extended with either Latin or French. Thus there are two programs: English-Latin (EL) and English-French (EF). From the ninth grade, three study directions are offered: Modern Languages (ML), Social Sciences (SOC), and Natural Sciences (NAT). Latin is a prerequisite for ML and ML students are taught French as third foreign language. In summary, there are five programs: ML-ELF, SOC-EL, SOC-EF, NAT-EL, and NAT-EF.

Example 2.1. Table 2.2 defines the seventh grade of a Gymnasium timetabling problem in terms of its modules. For each subject, there is a column and, for each class, there is a row. There are five classes (7a through 7e) all of which are homogeneous wrt. the foreign-language curriculum: 7a, 7b, and 7c are EL classes while 7d and 7e are EF classes.

Depending on the corresponding subject, a table field either relates to the corresponding class as a whole or only to a part of it. For example, the fields in religious education relate to the respective denominational groups only. With this policy, the meaning of the table fields is defined as follows:

- If the field (c,s) is empty, then there is no requirement to teach the subject s to any student of the class c.
- Numbers specify intra-class modules: If the field (c,s) contains n, then n time slots must be reserved weekly for teaching the subject s to the respective students of the class c. There is no requirement for joint education with students of other classes.
- Letters uniquely identify inter-class modules: If the field (c,s) contains m, then the respective students of the class c are to be educated jointly with the respective students of those classes for which the column s holds m, too.
- Subscripts and superscripts give additional information on the respective inter-class modules: The subscript specifies the required number of groups and the superscript specifies the number of time slots that need to be reserved weekly. For each inter-class module, there is exactly one field (the left- and/or top-most one) that gives this information.

From the state of the fields, we can draw conclusions on the composition of the classes: All classes are mixed wrt. gender, 7a and 7d are pure Catholic classes, and all the other classes are mixed wrt. religious denomination.

	Subject[a]															
Class	REC[b]	REP	Eth[c]	G	E	F	L	M	**B**	H	Geo	**A**	**Mu**	**PEM**[d]	**PEF**	#[e]
7a-EL[f]	2			4	4		5	4	2	2	1	2	2	d_2^2	g_1^2	30
7b-EL	a_3^2	b_1^2		4	4		5	4	2	2	1	2	2	e_1^2	h_2^2	30
7c-EL		b	c_1^2	4	4		5	4	2	2	1	2	2	d	g	30
7d-EF	a			4	4	5		4	2	2	1	2	2	f_1^2	i_1^2	30
7e-EF	a	b	c	4	4	5		4	2	2	1	2	2	e	h	30

Table 2.2:
The seventh grade of a Gymnasium timetabling problem in terms of its modules (cf. Example 2.1).

[a] Subject Codes: REC/REP = Religious Education for Catholics/Protestants, Eth = Ethics, G = German, E = English, F = French, L = Latin, M = Mathematics, B = Biology, H = History, Geo = Geography, A = Art, Mu = Music, PEM/PEF = Physical Education for Boys/Girls. If a subject code is printed in bold face, a science lab, a craft room, or some other special facility is required.

[b] By default, all Catholic students attend REC and all Protestant students attend REP.

[c] All students that are neither Catholic nor Protestant attend Eth. Furthermore, parents of a Catholic or Protestant child may require that their child attends Eth instead of REC or REP.

[d] In this school, the sexes are segregated in physical education.

[e] For each class, this column gives the total number of teaching periods per week.

[f] 7a is a pure Catholic class while all the other classes are mixed wrt. religious denomination.

The composition of the classes has lead to several inter-class modules. For example, module *e* specifies a boy group and requires that it has to attend physical education for two periods a week (cf. field (7b-EL, PEM)) while module *h* specifies two girl groups and requires that each of them has to attend physical education for two periods a week (cf. field (7b-EL, PEF)). Both modules arise from sexual segregation in physical education. Module *e* serves to save a teacher while module *h* serves to specify groups of similar size.

Interestingly, the introduction of the second foreign language has not entailed any inter-class modules because it was possible to create classes that are homogeneous wrt. the foreign-language curriculum. □

Example 2.2. Table 2.3 defines the tenth grade of a Gymnasium timetabling problem in terms of its modules. For each subject, there is a column. For each class and for each program that occurs in that class, there is a row. There are three classes (10a, 10b, and 10c) and the variety of study options has resulted in a complicated situation: 10a has students of ML-ELF, SOC-EL, and SOC-EF. 10b has students of SOC-EF and NAT-EF. 10c has students of NAT-EF and NAT-EL. Only 10b is homogeneous wrt. the foreign-language curriculum. Only 10c is homogeneous wrt. the study direction. 10a is heterogeneous in any respect. (In fact, 10a is a

Class	RE[b]	G	E	F	L	M	**Ph**	**C**	**B**	(H & Soc)[c]	EL	**A**[d]	**Mu**	**PE**[b]	**HE**	#
	Subject[a]															
10a/ML-ELF	a_4^2	b_1^3	e_1^3	5	j_1^3	k_2^3	m_2^2		q_1^2	t_1^3	w_1^1	z_2^1	α_2^1	β_2^2		30
10a/SOC-EL	a	b	e		j	k	m	o_1^2	q	u_1^4	w	z	α	β	δ_1^2	30
10a/SOC-EF	a	b	e	i_1^3		k	m	o	q	u	w	z	α	β	δ	30
10b/SOC-EF	a	c_1^3	f_1^3	h_1^3		k	m	o	r_1^2	u	x_1^1	z	α	γ_2^2	δ	30
10b/NAT-EF	a	c	f	h		l_1^4	n_1^3	p_1^3	r	t	x	z	α	γ		30
10c/NAT-EF	a	d_1^3	g_1^3	i		l	n	p	s_1^2	v_1^3	y_1^1	z	α	β		30
10c/NAT-EL	a	d	g		j	l	n	p	s	v	y	z	α	β		30

Table 2.3:
The tenth grade of a Gymnasium timetabling problem in terms of its modules (cf. Example 2.2).

[a]In addition to the subject codes introduced in Table 2.2, we use the following subject codes: RE = Religious Education, Ph = Physics, C = Chemistry, Soc = Social Studies, EL = Economics and Law, PE = Physical Education, HE = Home Economics

[b] For reasons of space, we merged REC, REP, and Eth into RE, and PEM and PEF into PE.

[c]History is taught for two periods a week in the first term and for one period a week in the second term. Depending on the study direction, Social Studies is taught for one or two periods a week in the first term and for two or three periods a week in the second term.

[d]Students must opt for either Art or Music.

strange class because its students are not educated jointly except for nine periods a week.)

The way table fields are labeled corresponds to Example 2.1 except for one extension: Letters are used to designate both inter-class and intra-class modules.

We observe two inter-class modules in foreign-language education: Modules i and j join the French and Latin takers, respectively, of 10a and 10c that learn French or Latin as second foreign language. Moreover, there are six inter-class modules in natural sciences: Modules k and m join the ML and SOC students of 10a and 10b for Mathematics and Physics, respectively. 10b had to be split because the curricula of NAT and SOC in natural sciences differ. ML students can be joined with SOC students because, except for Chemistry, their curricula in natural sciences are identical. Modules l and n join the NAT students of 10b and 10c for Mathematics and Physics, respectively. Modules o and p join the SOC and NAT students, respectively, for Chemistry. Finally, the SOC students of 10a and 10b are joined for Home Economics. All inter-class modules save teachers and avoid group sizes that differ too much. □

2.2.4 Typical Requirements of Timetables

In the course of timetabling, each lesson (and each other meeting like consulting hours) has to be assigned a start time and a suitable room. As a module specifies a teacher for each of its groups, the assignment of teachers to lessons is fixed in advance of timetabling.

Teacher timetables have to satisfy individual restrictions on availability (like do not schedule any lesson before 9 a.m.), moderate compactness constraints (like at most six periods of idle time a week), individual bounds on daily working time (like not less than two and not more than six periods), and individual bounds on the number of working days (like at least one day off).

Student timetables have to satisfy grade-specific restrictions on student availability: From the fifth through the tenth grade, compact timetables are enforced through tight time frames where the number of acceptable time slots equals the number of teaching periods as prescribed by official regulations. In the final grades, the time frames are loose and, to avoid unacceptable timetables, bounds on daily working time (like at least four and at most eight periods) and bounds on idle time (like at most two periods a day and at most six periods a week) are imposed.

Rooms timetables may have to satisfy room-specific restrictions on availability, for example, if public sports facilities have to be shared with other schools.

Job timetables have to satisfy job-specific distribution constraints (like at most two teaching periods a day).

2.2.5 Manual Timetabling

Starting out from the headmaster's problem specification, the timetables are created by dedicated teachers. In manual timetabling, it is common to proceed in an iterative fashion where each iteration selects and schedules a lesson. Scheduling a lesson requires to choose a room and a time slot s.t. the commitment to the choice will not violate any constraint. (If the school has a lot of rooms, room allocation may be performed after scheduling.) In case no good choice is available for the lesson under consideration, a conflict resolution method is used to free a good time slot by moving and interchanging lessons already scheduled. 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. Due to changes in problem structure, timetables have to be created from scratch every year.

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

Not only the timetable is subject to change in the course of local repair; the teacher assignment may be changed, too. But since consequences of changing the teacher assignment 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 teacher assignment.

Neither the next lesson to schedule, nor the next lesson to dispel is chosen arbitrarily because lessons are not equal with respect to number and tightness of constraints. Since the satisfaction of constraints gets harder and harder with the number of free time slots decreasing, it is clever to schedule 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 scheduled easier than the more constrained lessons. This strategy diminishes the average number of iterations necessary to resolve a conflict.

Usually, lesson selection is supported by a school-specific, hierarchical lesson selection scheme. 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 scheduling lessons from the highest level. After placing the most constrained lessons, timetabling continues with the next level, and so on. In general, when scheduling a level's lessons, the lessons of classes and teachers that only have a few lessons left to schedule are preferred. Again, this strategy avoids expensive conflict resolutions.

2.3 Recent Approaches in Automated School Timetabling

Research in automated school timetabling can be traced back to the 1960s [Jun86]. During the last decade, efforts concentrated on greedy algorithms and on local search methods such as simulated annealing, tabu search, and genetic algorithms

Reference	Methods						Problems	
	GA[a]	SA[b]	TS[c]	CP[d]	GrA[e]	HC[f]	artificial	real-world
[Cos94]			•		•			2
[YKYW96]				•	•	•		7
[Sch96b, Sch96a]			•			•	1	2
[DS97]	•				•		30	
[CDM98]	•	•	•			•		3
[FCMR99]	•							1
[KYN99]				•	•	•		4
[BK01]	•	•	•	•				
[CP01]	•							1

Table 2.4:
Overview of research in automated school timetabling

[a]Genetic Algorithm
[b]Simulated Annealing
[c]Tabu Search
[d]Constraint Propagation
[e]Greedy Algorithm
[f]Hill Climbing

(e.g. [Sch96a, DS97, CDM98, CP01]). Constraint-programming technology has been used to solve timetabling problems from universities (e.g. [GJBP96, HW96, GM99, AM00]) but the question whether it applies to school timetabling as well is open.

In the following, research is reviewed that has been carried out in the field of automated school timetabling during the past ten years (cf. Table 2.4). For reviews of earlier work on heuristic, mathematical-programming, and graph-theoretical approaches, the reader is refered to the well-known surveys of de Werra [dW85], Junginger [Jun86], and Schaerf [Sch95, Sch99].

Yoshikawa et al. [YKYW96] present the timetabling software ***SchoolMagic***. SchoolMagic implements a hybrid approach that combines a greedy algorithm with constraint propagation and local repair. The greedy algorithm is used to build a probably inconsistent initial assignment. Initially and after every commitment, the greedy algorithm prunes the search space by establishing arc consistency in a network of binary constraints. In scheduling, the greedy algorithm prefers most-constrained lessons and least-constraining values. (The authors do not explain their interpretation of most-constrained and least-constraining.) If a lesson cannot be scheduled, it is suspended and the greedy algorithm proceeds to schedule the remaining lessons. Finally, all suspended lessons are scheduled with the objective to minimize constraint violations. Then a hill climber is applied in an attempt to resolve the conflicts by moving single lessons. Yoshikawa et al. applied their approach to seven timetabling problems from three Japanese high schools. They characterize their solutions in terms of penalty points but do not describe the underlying objective function. However, they report that the application of SchoolMagic considerably reduced the costs of timetabling for all three schools.

Kaneko et al. [KYN99] improve on [YKYW96] by invoking local repair every time a lesson cannot be scheduled without constraint violations. Their hill climber escapes local optima by moving two lessons simultaneously while accepting minor additional constraint violations. Applying their approach to four timetabling problems from [YKYW96], the authors demonstrate that their improvements actually pay off.

Costa [Cos94] presents a timetabling algorithm based on tabu search. The algorithm starts out from a possible inconsistent initial assignment created by a greedy algorithm. In scheduling, the greedy algorithm prefers the lessons with the tightest restrictions on availability and it tries to avoid double-bookings. (Costa notes that it does not pay off to employ stronger methods in initial assignment because this might result in starting points already near to a local optima which are likely to trap tabu search.) In tabu search, the neighborhood is explored by moving single lessons. Costa applied his approach to two high-school timetabling problems from Switzerland. He reports that both problems were solved to full satisfaction.

Schaerf [Sch96b, Sch96a] presents a hybrid approach that interleaves tabu search with randomized hill climbing that allows for sideway moves to explore plateaus. The hybrid generates a random initial assignment and then hill climbing and tabu search take turns for a given number of cycles. Tabu search performs ***atomic moves*** while hill climbing performs ***double moves***. Atomic moves modify teacher timetables by moving or exchanging lessons. A double move sequences a pair of atomic moves. Hill climbing serves to make “substantial sideway modifications” to escape local optima. Schaerf applied his approach to two timetabling problems from Italian high schools and to one artificial problem. He reports that his algorithm produced conflict-free timetables which were better than timetables created manually or by commercial timetabling software.

Colorni et al. [CDM98] compare simulated annealing, tabu search, and a genetic algorithm. To escape local optima, their implementations of simulated annealing and tabu search may switch to a relaxed objective function for a few iterations. The genetic algorithm is equipped with a genetic repair function to resolve conflicts introduced by crossover and mutation. The algorithms were tested on three problems from two Italian high schools. For one problem, the algorithms started out from a hand-made timetable. For the other problems, random initializations were used. Tabu search with relaxation outperformed the genetic algorithm with repair which, in turn, performed better than simulated annealing with relaxation. All the algorithms produced conflict-free timetables that were better than timetables created manually or by commercial timetabling software.

Fernandes et al. [FCMR99] present a genetic algorithm that starts out from a random initial population and employs genetic repair to speed up evolution. The repair function tries to reduce the number of conflicts by moving lessons. It is guided by a strategy that recommends to move most-constrained lessons first. The algorithm was studied on a problem from a Portuguese high school. It produced a timetable that was conflict-free but worse than a hand-made timetable.

Drexl & Salewski [DS97] compare a randomized greedy algorithm to a genetic algorithm that optimizes scheduling strategies. The greedy algorithm is guided by scheduling strategies (***priority rules***) which rank alternatives in terms of ***priority values***. Decisions are met “randomly but according to probabilities which are proportional to priority values”. Thus no alternative is ruled out completely though those with higher probabilities are more likely to be selected. The genetic algorithm operates on genes that represent random numbers which are used to drive the greedy algorithm to obtain timetables from genetic information. To test their algorithms, Drexl & Salewski generated 30 problems from three classes of tractability (easy, medium, and hard) with 35 lessons on average. The authors report that both approaches yield solutions close to optimality.

Brucker and Knust [BK01] advocate the application of project-scheduling technology to school timetabling. Their proposal is centered around (variants of) the ***resource-constrained project scheduling problem*** (RCPSP) where a set of tasks with individual resource demands has to be scheduled within a given time horizon under consideration of precedence constraints, parallelity constraints, and resource availabilities, among others, while minimizing a cost function. For this kind of problem, the authors present a variety of deductive methods and they explain how these methods could be employed in local search (to preprocess the input) and in tree search (to prune the search space after each commitment). The authors do not report any computational results.

Carrasco and Pato [CP01] start out from the observation that the sole quest for good teacher timetables entails bad student timetables and vice versa. In face of two objectives that conflict with each other in optimization, the authors propose a genetic algorithm that searches for pareto-optimal solutions. Carrasco and Pato applied their approach to a timetabling problem from a Portuguese vocational school. They report that the genetic algorithm produced many non-dominated solutions that were better than a hand-made timetable.

Indeed, the results reported by Kaneko et al., Costa, Schaerf, and Colorni et al. are promising and lead to the hypothesis that tabu search is a good candidate for timetabling. But note that we cannot come to serious conclusions because all authors worked with samples that are too small to allow for statistical inference and thus their results do not generalize beyond their samples. This common deficiency in experimental design has a simple reason: despite 40 years of automated school timetabling, the timetabling community has not yet compiled a suitable problem set. In fact, Costa [Cos94] complains that "it is difficult to judge the value of a given timetable algorithm" because "there are no standard test problems or measures of difficulty". Schaerf [Sch96b] adds that "the absence of a common definition of the problem and of widely-accepted benchmarks prevents us from comparing with other algorithms".

Drexl & Salewski [DS97] deserve special mention because they used a self-made set of artificial problems. To make sure that their artificial problems resemble real-world problems, the problem generator was configured with characteristics of German secondary schools. Unfortunately, the approach was not brought to bear: For computational reasons, only 30 problems of 35 lessons on average were tested. The authors do not discuss how their results could generalize to real problems that have more than 500 lessons.

Chapter 3

Track Parallelization

This chapter introduces the ***track parallelization problem*** (TPP) and proposes the `tpp` constraint along with a suitable solver for modeling and solving this kind of scheduling problem in a finite-domain constraint-programming framework.

TPPs play a central role in Chapter 4 which proposes and investigates constraint-based solvers for school timetabling. TPPs are used to down-size the underlying constraint models (by a transformation a priori to search) and to prune the search space (by propagation during search).

This chapter is organized as follows. Section 3.1 introduces the TPP and demonstrates that track parallelization is NP-complete. Section 3.2 introduces the `tpp` constraint and prepares the grounds for the further treatment. Section 3.3 proposes a rule-based solver for `tpp` constraints, demonstrates its correctness, and gives performance guarantees. Section 3.4 closes the chapter with a survey of related work in operations research.

3.1 The Track Parallelization Problem

A ***track parallelization problem*** (TPP) is specified by a set of task sets $\mathcal{T}$. The task sets are called ***tracks***. For each task $t \in \bigcup \mathcal{T}$, we are given a set of ***start times*** $S(t)$ and a set of ***processing times*** $P(t)$. The problem of solving a TPP consists in scheduling the tasks s.t. the tracks are processed in parallel. More precisely, for each task $t \in \bigcup \mathcal{T}$, we are required to find a ***processing interval***

$$[s(t), s(t) + p(t) - 1]$$

with $s(t) \in S(t)$ and $p(t) \in P(t)$ s.t.

$$|\{\mathrm{vc}(T) : T \in \mathcal{T}\}| = 1$$

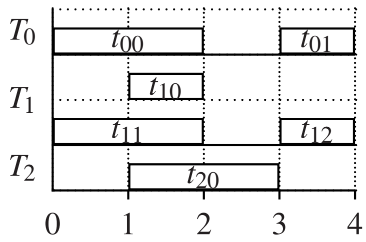
where

$$\mathrm{vc}(T) = \bigcup_{t \in T} [s(t), s(t) + p(t) - 1]$$

is called the ***value cover*** of the track T.

If some time slot a is contained in the processing interval of some task t, we say that "t covers a". In the same manner, if some time slot a is contained in the value cover $\mathrm{vc}(T)$ of some track T, we say that "T covers a" or that "the schedule of T covers a".

In its simplest form, a TPP requires to process two tasks in parallel. For a more typical example, consider the following Gantt-like chart:



The chart is based on the tracks T_0, T_1, and T_2 with $T_0 = \{t_{00}, t_{01}\}$, $T_1 = \{t_{10}, t_{11}, t_{12}\}$, $T_2 = \{t_{20}\}$, $s(t_{00}) = s(t_{11}) = 0$, $p(t_{00}) = p(t_{11}) = 2$, $s(t_{01}) = s(t_{12}) = 3$, $p(t_{01}) = p(t_{12}) = 1$, $s(t_{10}) = p(t_{10}) = 1$, $s(t_{20}) = 1$, and $p(t_{20}) = 2$. T_0 and T_1 are processed in parallel because $\mathrm{vc}(T_0) = \mathrm{vc}(T_1) = \{0, 1, 3\}$. In contrast, T_2 is not processed in parallel to the other tracks because $\mathrm{vc}(T_2) = \{1, 2\}$. Its schedule covers time slots that the other schedules do not cover and vice versa.

Proposition 3.1. *Track parallelization is NP-complete.*

Proof. It is easy to see that TPP $\in$ NP: A non-deterministic algorithm only has to guess a schedule and check in polynomial time that all tracks are processed in parallel.

We will transform SAT[1] to TPP. Let $U = \{u_1, \ldots, u_n\}$ be the set of variables and $C = \{c_1, \ldots, c_m\}$ be the set of clauses in an arbitrary SAT instance. We must

[1]SAT (e.g. [GJ79]) is a decision problem from Boolean logic. A SAT instance is defined on the basis of a variable set U. If $u \in U$, then u and $\bar{u}$ are called ***literals*** over U. A ***truth assignment*** for U is a Boolean function on U. For $u \in U$, the literal u ($\bar{u}$) is true under a truth assignment iff the variable u is true (false) under that assignment.

A SAT instance is specified by a set of clauses C over U. A ***clause*** over U is a set of literals over U. A clause represents the disjunction of its literals and hence is ***satisfied*** by a truth assignment iff it contains a literal that is true under that assignment.

A SAT instance C is called ***satisfiable*** iff a truth assignment exists that simultaneously satisfies all $c \in C$. Such a truth assignment is called a ***satisfying truth assignment*** for C.

The SAT problem is specified as follows: Given a SAT instance C, is C satisfiable? The SAT problem is NP-complete.

construct a track set $\mathcal{T}$ s.t. the corresponding TPP is satisfiable iff the SAT instance is satisfiable.

1. We introduce tasks v and w with $S(v) = S(w) = \{0\}$, $P(v) = \{1\}$, and $P(w) = \{2\}$.

2. For $1 \leq i \leq n$, we introduce tasks t_i and $\bar{t}_i$ with $S(t_i) = S(\bar{t}_i) = \{0,1\}$ and $P(t_i) = P(\bar{t}_i) = \{1\}$.

3. For $1 \leq i \leq n$, we construct a track $T_{u_i} = \{t_i, \bar{t}_i\}$.

4. For $1 \leq i \leq m$, we construct a track $T_{c_i} = \{v\} \cup \{g(l) : l \in c_i\}$ where $g(u_i) = t_i$ and $g(\bar{u}_i) = \bar{t}_i$.

Let $\mathcal{T} = \{\{w\}, T_{u_1}, \ldots, T_{u_n}, T_{c_1}, \ldots, T_{c_m}\}$.

Let σ be a satisfying truth assignment for C. To simplify matters, we represent *false* by 0 and *true* by 1. With this representation, we extend σ to literals over C in the following way: For $u \in U$, $\sigma(\bar{u}) = 1 - \sigma(u)$.

We construct a schedule as follows: For $1 \leq i \leq n$, let $s(t_i) = \sigma(u_i)$, $s(\bar{t}_i) = \sigma(\bar{u}_i)$, and $p(t_i) = p(\bar{t}_i) = 1$. Moreover, let $s(v) = s(w) = 0$, $p(v) = 1$, and $p(w) = 2$. We consider all tracks of $\mathcal{T}$ in turn:

1. Obviously, $\mathrm{vc}(\{w\}) = \{0,1\}$.

2. Let $1 \leq i \leq n$. If $s(t_i) = 0$, then $s(\bar{t}_i) = 1$. If $s(t_i) = 1$, then $s(\bar{t}_i) = 0$. With $p(t_i) = p(\bar{t}_i) = 1$, we obtain $\mathrm{vc}(T_{u_i}) = \{0,1\}$.

3. Let $1 \leq i \leq m$. Let $l_1, \ldots, l_{|c_i|}$ denote the literals of c_i. Let h be a function on $\{1, \ldots, |c_i|\}$ s.t. $l_j = u_{h(j)}$ or $l_j = \bar{u}_{h(j)}$ for $1 \leq j \leq |c_i|$. We know that $T_{c_i} = \left\{v, g(l_1), \ldots, g(l_{|c_i|})\right\}$. Obviously, $\mathrm{vc}(\{v\}) = \{0\}$. For $1 \leq j \leq |c_i|$, $\mathrm{vc}(\left\{g(l_j)\right\}) \subseteq \{0,1\}$ because $S(g(l_j)) = \{0,1\}$ and $P(g(l_j)) = \{1\}$. Moreover, $1 \leq j \leq |c_i|$ exists with $\sigma(l_j) = 1$ because σ satisfies c_i. Let $k = h(j)$. If $l_j = u_k$, then

$$s(g(l_j)) = s(g(u_k)) = s(t_k) = \sigma(u_k) = \sigma(l_j) = 1.$$

If $l_j = \bar{u}_k$, then

$$s(g(l_j)) = s(g(\bar{u}_k)) = s(\bar{t}_k) = \sigma(\bar{u}_k) = \sigma(l_j) = 1.$$

We conclude that $\mathrm{vc}(T_{c_i}) = \{0,1\}$.

We conclude that all tracks of $\mathcal{T}$ are processed in parallel.

Suppose s and p specify a schedule s.t. all tracks of $\mathcal{T}$ are processed in parallel. We observe that $\mathrm{vc}(\{w\}) = \{0,1\}$. We conclude that, for $1 \leq i \leq m$, $\mathrm{vc}(T_{c_i}) = \{0,1\}$ and that, for $1 \leq i \leq n$, $\mathrm{vc}(T_{u_i}) = \{0,1\}$ and thus $s(\bar{t}_i) = 1 - s(t_i)$.

We construct a satisfying truth assignment σ for C as follows: For $1 \leq i \leq n$, $\sigma(u_i) = s(t_i)$. We extend σ to literals over C in the following way: For $u \in U$, $\sigma(\bar{u}) = 1 - \sigma(u)$. It follows that $\sigma(\bar{u}_i) = s(\bar{t}_i)$ for $1 \leq i \leq n$.

We have to check that σ satisfies all clauses in C. Let $1 \leq i \leq m$. Let $l_1, \ldots, l_{|c_i|}$ denote the literals of c_i. Let h be a function on $\{1, \ldots, |c_i|\}$ s.t. $l_j = u_{h(j)}$ or $l_j = \bar{u}_{h(j)}$ for $1 \leq j \leq |c_i|$. $1 \in \mathrm{vc}(T_{c_i})$ and hence $1 \leq j \leq |c_i|$ exists with $s(g(l_j)) = 1$. Let $k = h(j)$. If $l_j = u_k$, then

$$\sigma(l_j) = \sigma(u_k) = s(t_k) = s(g(u_k)) = s(g(l_j)) = 1.$$

If $l_j = \bar{u}_k$, then

$$\sigma(l_j) = \sigma(\bar{u}_k) = s(\bar{t}_k) = s(g(\bar{u}_k)) = s(g(l_j)) = 1.$$

□

The same result could have been obtained by a transformation from 3SAT[2]. This shows that the TPP remains NP-complete even if the cardinalities of tracks and domains are limited to three and two, respectively.

3.2 The `tpp` Constraint

For track parallelization in a finite-domain constraint-programming framework, we introduce the `tpp` constraint. The `tpp` constraint takes one argument: a non-empty set $\mathcal{T}$ of non-empty sets of pairs of finite-domain variables. Each $T \in \mathcal{T}$ is intended to model a track and each pair $(\mathtt{S}, \mathtt{P})$ of finite-domain variables is intended to model a task in terms of start and processing time. Fixed start or processing times may be modeled by means of variables with singleton domains. We assume that processing times are greater than 0.

Notation 3.1. If $P = (X, \delta, C)$ is a FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, and $t = (\mathtt{S}, \mathtt{P}) \in T$, we write $\delta(t)$ instead of $\delta(\mathtt{S}) \times \delta(\mathtt{P})$.

[2] 3SAT is a special case of SAT. A 3SAT instance C is a SAT instance with $|c| = 3$ for all $c \in C$. The 3SAT problem is NP-complete.

Definition 3.1. Let $P = (X, \delta, C)$ be a FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. Let $T \in \mathcal{T}$ and $t = (\mathtt{S}, \mathtt{P}) \in T$. We define ***value covers*** and ***value supplies*** as follows:

$$\mathrm{vc}(t, \delta) = \begin{cases} \emptyset, & \text{if } \delta(t) = \emptyset \\ \bigcap_{(s,p) \in \delta(t)} [s, s+p-1] & \text{otherwise} \end{cases}$$

$$\mathrm{vc}(T, \delta) = \bigcup_{t \in T} \mathrm{vc}(t, \delta)$$

$$\mathrm{vc}(\mathcal{T}, \delta) = \bigcup_{T \in \mathcal{T}} \mathrm{vc}(T, \delta)$$

$$\mathrm{vs}(t, \delta) = \bigcup_{(s,p) \in \delta(t)} [s, s+p-1]$$

$$\mathrm{vs}(T, \delta) = \bigcup_{t \in T} \mathrm{vs}(t, \delta)$$

$$\mathrm{vs}(\mathcal{T}, \delta) = \bigcap_{T \in \mathcal{T}} \mathrm{vs}(T, \delta)$$

As Lemma 3.1 will show, every time slot in $\mathrm{vc}(t, \delta)$ will be covered by t (will be contained in the processing interval of t) in any solution to P. Consequently, in any solution to P, the schedule of T will cover every time slot in $\mathrm{vc}(T, \delta)$ and, for every time slot in $\mathrm{vc}(\mathcal{T}, \delta)$, there will be a track $T \in \mathcal{T}$ the schedule of which covers the time slot.

Conversely, as Lemma 3.1 and Lemma 3.4 will show, every time slot that is not contained in $\mathrm{vs}(t, \delta)$ (that is not supplied to t) will not be covered by t in any solution to P. Consequently, in any solution to P, the schedule of T will not cover any time slot that is not contained in $\mathrm{vs}(T, \delta)$ and, for every time slot that is not contained in $\mathrm{vs}(\mathcal{T}, \delta)$, there will be no track $T \in \mathcal{T}$ the schedule of which covers the time slot.

The meaning of `tpp` constraints is defined in terms of track value covers.

Definition 3.2. Let $P = (X, \delta, C)$ be a ground FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. δ satisfies $\mathtt{tpp}(\mathcal{T})$ iff

$$|\{\mathrm{vc}(T, \delta) : T \in \mathcal{T}\}| = 1,$$

i.e. iff the tracks are processed in parallel.

Lemma 3.4 will show that the meaning of `tpp` constraints could have been defined in terms of track value supplies as well.

Definition 3.3. Let $P = (X, \delta, C)$ be an unfailed FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. Let $T \in \mathcal{T}$ and $t = (\mathtt{S}, \mathtt{P}) \in T$. We define ***earliest start times*** and ***latest completion times*** as follows:

$$\begin{aligned}
\mathrm{est}(t, \delta) &= \min \delta(\mathtt{S}) \\
\mathrm{est}(T, \delta) &= \min_{t \in T} \mathrm{est}(t, \delta) \\
\mathrm{est}(\mathcal{T}, \delta) &= \max_{T \in \mathcal{T}} \mathrm{est}(T, \delta)
\end{aligned}$$

$$\begin{aligned}
\mathrm{lct}(t, \delta) &= \max \delta(\mathtt{S}) + \max \delta(\mathtt{P}) - 1 \\
\mathrm{lct}(T, \delta) &= \max_{t \in T} \mathrm{lct}(t, \delta) \\
\mathrm{lct}(\mathcal{T}, \delta) &= \min_{T \in \mathcal{T}} \mathrm{lct}(T, \delta)
\end{aligned}$$

Value supplies, value covers, earliest start times, and latest completion times have nice monotonicity properties that are summarized in Lemma 3.1 and Lemma 3.2. Lemma 3.3 shows that value supplies are closely related to earliest start and latest completion times. Lemma 3.4 summarizes properties of ground FCSPs.

Lemma 3.1. *Suppose $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ are FCSPs with $P_0 \rightarrow_{\mathrm{FD}} P_1$ and $\mathtt{tpp}(\mathcal{T}) \in C$. Let $T \in \mathcal{T}$ and $t \in T$.*

1. *Value supplies shrink monotonically, i.e.*
   $\mathrm{vs}(t, \delta_0) \supseteq \mathrm{vs}(t, \delta_1)$, $\mathrm{vs}(T, \delta_0) \supseteq \mathrm{vs}(T, \delta_1)$, *and* $\mathrm{vs}(\mathcal{T}, \delta_0) \supseteq \mathrm{vs}(\mathcal{T}, \delta_1)$.
2. *Value covers grow monotonically, i.e.*
   $\mathrm{vc}(t, \delta_0) \subseteq \mathrm{vc}(t, \delta_1)$, $\mathrm{vc}(T, \delta_0) \subseteq \mathrm{vc}(T, \delta_1)$, *and* $\mathrm{vc}(\mathcal{T}, \delta_0) \subseteq \mathrm{vc}(\mathcal{T}, \delta_1)$.

Proof. All properties follow immediately from Corollary 1.1. □

Lemma 3.2. *Suppose $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ are unfailed FCSPs with $P_0 \rightarrow_{\mathrm{FD}} P_1$ and $\mathtt{tpp}(\mathcal{T}) \in C$. Let $T \in \mathcal{T}$ and $t \in T$.*

1. *Earliest start times grow monotonically, i.e.*
   $\mathrm{est}(t, \delta_0) \leq \mathrm{est}(t, \delta_1)$, $\mathrm{est}(T, \delta_0) \leq \mathrm{est}(T, \delta_1)$, *and* $\mathrm{est}(\mathcal{T}, \delta_0) \leq \mathrm{est}(\mathcal{T}, \delta_1)$.
2. *Latest completion times shrink monotonically, i.e.*
   $\mathrm{lct}(t, \delta_0) \geq \mathrm{lct}(t, \delta_1)$, $\mathrm{lct}(T, \delta_0) \geq \mathrm{lct}(T, \delta_1)$, *and* $\mathrm{lct}(\mathcal{T}, \delta_0) \geq \mathrm{lct}(\mathcal{T}, \delta_1)$.

Proof. All properties follow immediately from Corollary 1.1. □

Lemma 3.3. *Suppose $P = (X, \delta, C)$ is an unfailed FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. Let $T \in \mathcal{T}$ and $t \in T$.*

1. *Earliest start times are equal to the least elements of value supplies, i.e.* $\mathrm{est}(t,\delta) = \min \mathrm{vs}(t,\delta)$*,* $\mathrm{est}(T,\delta) = \min \mathrm{vs}(T,\delta)$*, and* $\mathrm{est}(\mathcal{T},\delta) = \min \mathrm{vs}(\mathcal{T},\delta)$*.*

2. *Latest completion times are equal to the greatest elements of value supplies, i.e.* $\mathrm{lct}(t,\delta) = \max \mathrm{vs}(t,\delta)$*,* $\mathrm{lct}(T,\delta) = \max \mathrm{vs}(T,\delta)$*, and* $\mathrm{lct}(\mathcal{T},\delta) = \max \mathrm{vs}(\mathcal{T},\delta)$*.*

Proof. Let $(\mathtt{S},\mathtt{P}) = t$.

1. We observe that

$$\begin{aligned}
\mathrm{est}(t,\delta) &= \min \delta(\mathtt{S}) = \min_{s\in\delta(\mathtt{S})} s \\
&= \min_{s\in\delta(\mathtt{S})} \min\,[s, s+\max\delta(\mathtt{P})-1] \\
&= \min \bigcup_{s\in\delta(\mathtt{S})} [s, s+\max\delta(\mathtt{P})-1] \\
&= \min \bigcup_{(s,p)\in\delta(t)} [s, s+p-1] = \min \mathrm{vs}(t,\delta).
\end{aligned}$$

This result yields

$$\begin{aligned}
\mathrm{est}(T,\delta) &= \min_{t\in T} \mathrm{est}(t,\delta) = \min_{t\in T} \min \mathrm{vs}(t,\delta) = \min \bigcup_{t\in T} \mathrm{vs}(t,\delta) \\
&= \min \mathrm{vs}(T,\delta).
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathrm{est}(\mathcal{T},\delta) &= \max_{T\in\mathcal{T}} \mathrm{est}(T,\delta) = \max_{T\in\mathcal{T}} \min \mathrm{vs}(T,\delta) = \min \bigcap_{T\in\mathcal{T}} \mathrm{vs}(T,\delta) \\
&= \min \mathrm{vs}(\mathcal{T},\delta).
\end{aligned}$$

2. We observe that

$$\begin{aligned}
\mathrm{lct}(t,\delta) &= \max \delta(\mathtt{S}) + \max \delta(\mathtt{P}) - 1 \\
&= \max_{s\in\delta(\mathtt{S})} (s + \max\delta(\mathtt{P}) - 1) \\
&= \max_{s\in\delta(\mathtt{S})} \max\,[s, s+\max\delta(\mathtt{P})-1] \\
&= \max \bigcup_{s\in\delta(\mathtt{S})} [s, s+\max\delta(\mathtt{P})-1] \\
&= \max \bigcup_{(s,p)\in\delta(t)} [s, s+p-1] = \max \mathrm{vs}(t,\delta).
\end{aligned}$$

This result yields

$$\begin{aligned}\operatorname{lct}(T,\delta) &= \max_{t\in T}\operatorname{lct}(t,\delta) = \max_{t\in T}\max\operatorname{vs}(t,\delta) = \max\bigcup_{t\in T}\operatorname{vs}(t,\delta)\\ &= \max\operatorname{vs}(T,\delta).\end{aligned}$$

It follows that

$$\begin{aligned}\operatorname{lct}(\mathcal{T},\delta) &= \min_{T\in\mathcal{T}}\operatorname{lct}(T,\delta) = \min_{T\in\mathcal{T}}\max\operatorname{vs}(T,\delta) = \max\bigcap_{T\in\mathcal{T}}\operatorname{vs}(T,\delta)\\ &= \max\operatorname{vs}(\mathcal{T},\delta).\end{aligned}$$

□

Lemma 3.4. *Suppose $P = (X,\delta,C)$ is a ground FCSP with $\texttt{tpp}(\mathcal{T}) \in C$. Let $T \in \mathcal{T}$ and $t \in T$.*

1. *In general,* $\operatorname{vs}(t,\delta) = \operatorname{vc}(t,\delta)$ *and* $\operatorname{vs}(T,\delta) = \operatorname{vc}(T,\delta)$.
2. *Furthermore, if δ satisfies $\texttt{tpp}(\mathcal{T})$, then*
   (a) $\operatorname{vc}(T,\delta) = \operatorname{vs}(\mathcal{T},\delta) = \operatorname{vc}(\mathcal{T},\delta)$,
   (b) $\operatorname{est}(T,\delta) = \operatorname{est}(\mathcal{T},\delta)$, *and*
   (c) $\operatorname{lct}(T,\delta) = \operatorname{lct}(\mathcal{T},\delta)$.

Proof. Let $(\texttt{S},\texttt{P}) = t$.

1. We observe that

$$\begin{aligned}\operatorname{vs}(t,\delta) &= \bigcup_{(s,p)\in\delta(t)}[s,s+p-1]\\ &= [\texttt{S}\delta,\texttt{S}\delta+\texttt{P}\delta-1]\\ &= \bigcap_{(s,p)\in\delta(t)}[s,s+p-1] = \operatorname{vc}(t,\delta).\end{aligned}$$

   It follows that

$$\operatorname{vs}(T,\delta) = \bigcup_{t\in T}\operatorname{vs}(t,\delta) = \bigcup_{t\in T}\operatorname{vc}(t,\delta) = \operatorname{vc}(T,\delta).$$

2. Suppose δ satisfies $\texttt{tpp}(\mathcal{T})$.
   (a) Definition 3.2 implies that $\operatorname{vc}(\mathcal{T},\delta) = \operatorname{vc}(T,\delta)$. It follows that

$$\operatorname{vs}(\mathcal{T},\delta) = \bigcap_{T\in\mathcal{T}}\operatorname{vs}(T,\delta) = \bigcap_{T\in\mathcal{T}}\operatorname{vc}(T,\delta) = \bigcap_{T\in\mathcal{T}}\operatorname{vc}(\mathcal{T},\delta) = \operatorname{vc}(\mathcal{T},\delta).$$

(b) From Lemma 3.3 we know that $\mathrm{est}(T,\delta) = \min \mathrm{vs}(T,\delta)$. It follows that

$$\begin{aligned}\mathrm{est}(\mathcal{T},\delta) &= \max_{T\in\mathcal{T}} \mathrm{est}(T,\delta) = \max_{T\in\mathcal{T}} \min \mathrm{vs}(T,\delta) = \max_{T\in\mathcal{T}} \min \mathrm{vs}(\mathcal{T},\delta)\\ &= \min \mathrm{vs}(\mathcal{T},\delta) = \min \mathrm{vs}(T,\delta) = \mathrm{est}(T,\delta).\end{aligned}$$

(c) From Lemma 3.3 we know that $\mathrm{lct}(T,\delta) = \max \mathrm{vs}(T,\delta)$. It follows that

$$\begin{aligned}\mathrm{lct}(\mathcal{T},\delta) &= \min_{T\in\mathcal{T}} \mathrm{lct}(T,\delta) = \min_{T\in\mathcal{T}} \max \mathrm{vs}(T,\delta) = \min_{T\in\mathcal{T}} \max \mathrm{vs}(\mathcal{T},\delta)\\ &= \max \mathrm{vs}(\mathcal{T},\delta) = \max \mathrm{vs}(T,\delta) = \mathrm{lct}(T,\delta).\end{aligned}$$

□

Corollary 3.1. *Suppose $P_0 = (X,\delta_0,C)$ is a FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. If $P_0 \rightarrow_{\mathrm{FD}} \ldots \rightarrow_{\mathrm{FD}} \Sigma$ with $\Sigma = (X,\sigma,C)$, Σ is ground, and σ satisfies $\mathtt{tpp}(\mathcal{T})$, then the relations depicted in Figure 3.1 hold.*

3.3 Solving `tpp` Constraints

This section proposes reductions for propagating `tpp` constraints, demonstrates their correctness, and gives performance guarantees like convergence and the ability to recognize constraint violations.

3.3.1 Reductions

We propose five reductions for propagating `tpp` constraints, namely $\rightarrow_{\mathrm{PVS}}$, $\rightarrow_{\mathrm{PVSB}}$, $\rightarrow_{\mathrm{FC}}$, $\rightarrow_{\mathrm{IPT}}$, and $\rightarrow_{\mathrm{NC}}$. Each reduction (and every combination of these reductions by means of set union) constitutes a deduction component with every transition operating on some `tpp` constraint and the domains of its variables.

- $\rightarrow_{\mathrm{PVS}}$ identifies and prunes all start and processing times that entail the covering of a value that is not element of the value supply of the track set of the `tpp` constraint under consideration.

- $\rightarrow_{\mathrm{PVSB}}$ is similar to $\rightarrow_{\mathrm{PVS}}$ except for that it does not consider domain holes. ($\rightarrow_{\mathrm{PVS}}$ performs ***domain reasoning*** while $\rightarrow_{\mathrm{PVSB}}$ performs ***bound reasoning***.)

- Under certain conditions, $\rightarrow_{\mathrm{FC}}$ forces tasks to cover values.

- $\rightarrow_{\mathrm{IPT}}$ reveals inconsistencies by comparing bounds on the processing times of tracks.

$$
\begin{array}{ccccccc}
\mathrm{vc}(t,\delta_0) & \subseteq \ldots \subseteq & \mathrm{vc}(t,\sigma) & = & \mathrm{vs}(t,\sigma) & \subseteq \ldots \subseteq & \mathrm{vs}(t,\delta_0) \\
\rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} \\
\mathrm{vc}(T,\delta_0) & \subseteq \ldots \subseteq & \mathrm{vc}(T,\sigma) & = & \mathrm{vs}(T,\sigma) & \subseteq \ldots \subseteq & \mathrm{vs}(T,\delta_0) \\
\rotatebox{-90}{$\subseteq$} & & \Vert & & \Vert & & \rotatebox{-90}{$\subseteq$} \\
\mathrm{vc}(\mathcal{T},\delta_0) & \subseteq \ldots \subseteq & \mathrm{vc}(\mathcal{T},\sigma) & = & \mathrm{vs}(\mathcal{T},\sigma) & \subseteq \ldots \subseteq & \mathrm{vs}(\mathcal{T},\delta_0)
\end{array}
$$

and

$$
\begin{array}{ccccccc}
\mathrm{est}(t,\delta_0) & \le \ldots \le & \mathrm{est}(t,\sigma) & \le & \mathrm{lct}(t,\sigma) & \le \ldots \le & \mathrm{lct}(t,\delta_0) \\
\Vert & & \Vert & & \Vert & & \Vert \\
\min \mathrm{vs}(t,\delta_0) & \le \ldots \le & \min \mathrm{vs}(t,\sigma) & \le & \max \mathrm{vs}(t,\sigma) & \le \ldots \le & \max \mathrm{vs}(t,\delta_0) \\
\rotatebox{-90}{$\ge$} & & \rotatebox{-90}{$\ge$} & & \rotatebox{-90}{$\le$} & & \rotatebox{-90}{$\le$} \\
\mathrm{est}(T,\delta_0) & \le \ldots \le & \mathrm{est}(T,\sigma) & \le & \mathrm{lct}(T,\sigma) & \le \ldots \le & \mathrm{lct}(T,\delta_0) \\
\Vert & & \Vert & & \Vert & & \Vert \\
\min \mathrm{vs}(T,\delta_0) & \le \ldots \le & \min \mathrm{vs}(T,\sigma) & \le & \max \mathrm{vs}(T,\sigma) & \le \ldots \le & \max \mathrm{vs}(T,\delta_0) \\
\rotatebox{-90}{$\le$} & & \Vert & & \Vert & & \rotatebox{-90}{$\ge$} \\
\mathrm{est}(\mathcal{T},\delta_0) & \le \ldots \le & \mathrm{est}(\mathcal{T},\sigma) & \le & \mathrm{lct}(\mathcal{T},\sigma) & \le \ldots \le & \mathrm{lct}(\mathcal{T},\delta_0) \\
\Vert & & \Vert & & \Vert & & \Vert \\
\min \mathrm{vs}(\mathcal{T},\delta_0) & \le \ldots \le & \min \mathrm{vs}(\mathcal{T},\sigma) & \le & \max \mathrm{vs}(\mathcal{T},\sigma) & \le \ldots \le & \max \mathrm{vs}(\mathcal{T},\delta_0)
\end{array}
$$

Figure 3.1: The conclusions of Corollary 3.1.

- $\to_{\mathrm{NC}}$ reveals inconsistencies by identifying situations where values that have to be covered cannot be covered.

We will see that $\to_{\mathrm{PVS}}$ and $\to_{\mathrm{NC}}$ are full-fledged solvers for tpp constraints (in the sense of Definition 1.3) while $\to_{\mathrm{FC}}$ and $\to_{\mathrm{IPT}}$ lack the ability to detect constraint violations reliably.

Definition 3.4. Let $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ be FCSPs. We say that $P_0 \to_{\mathrm{PVS}} P_1$ iff $P_0 \to_{\mathrm{FD}} P_1$ and $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, $t = (\mathtt{S}, \mathtt{P}) \in T$, and $a \in \mathrm{vs}(t, \delta_0)$ exist s.t. $a \notin \mathrm{vs}(\mathcal{T}, \delta_0)$ and $\delta_1 = \delta_0$ except for

$$\delta_1(\mathtt{S}) = \{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\, a \notin [s, s+p-1]\}$$

and

$$\delta_1(\mathtt{P}) = \{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\, a \notin [s, s+p-1]\}.$$

Example 3.1. Consider the problem $P_0 = (X, \delta_0, \{\mathtt{tpp}(\mathcal{T})\})$ with $\mathcal{T} = \{T_0, T_1\}$, $T_0 = \{t_{00}\}$, $T_1 = \{t_{10}\}$, $t_{00} = (\mathtt{S}_{00}, \mathtt{P}_{00})$, $t_{10} = (\mathtt{S}_{10}, \mathtt{P}_{10})$, $\delta_0(\mathtt{S}_{00}) = \{1,2,4\}$, $\delta_0(\mathtt{P}_{00}) = \{1\}$, $\delta_0(\mathtt{S}_{10}) = \{1,3,4\}$, and $\delta_0(\mathtt{P}_{10}) = \{1\}$. $\to_{\mathrm{PVS}}$ applies two times:

1. $\to_{\mathrm{PVS}}$ applies to t_{00} because $2 \in \mathrm{vs}(t_{00}, \delta_0) = \{1,2,4\}$ and $2 \notin \mathrm{vs}(\mathcal{T}, \delta_0) = \{1,4\}$. We obtain the problem P_1 with $\delta_1 = \delta_0$ except for $\delta_1(\mathtt{S}_{00}) = \{1,4\}$.
2. $\to_{\mathrm{PVS}}$ applies to t_{10} because $3 \in \mathrm{vs}(t_{00}, \delta_1) = \{1,3,4\}$ and $3 \notin \mathrm{vs}(\mathcal{T}, \delta_1) = \{1,4\}$. We obtain the problem P_2 with $\delta_2 = \delta_1$ except for $\delta_2(\mathtt{S}_{10}) = \{1,4\}$.

Definition 3.5. Let $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ be FCSPs. We say that $P_0 \to_{\mathrm{PVSB}} P_1$ iff $P_0 \to_{\mathrm{FD}} P_1$, P_0 is not failed, and $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$ and $t = (\mathtt{S}, \mathtt{P}) \in T$ exist s.t. $\delta_1 = \delta_0$ except for

$$\delta_1(\mathtt{S}) = \{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\, (s,p) \text{ is feasible}\}$$

and

$$\delta_1(\mathtt{P}) = \{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\, (s,p) \text{ is feasible}\}$$

where we say that (s,p) is ***feasible*** iff

$$\mathrm{est}(\mathcal{T}, \delta_0) \le s \le \mathrm{lct}(\mathcal{T}, \delta_0) - \min \delta_0(\mathtt{P}) + 1$$

and

$$p \le \mathrm{lct}(\mathcal{T}, \delta_0) - \max\{\min \delta_0(\mathtt{S}), \mathrm{est}(\mathcal{T}, \delta_0)\} + 1.$$

Example 3.2. Consider the problem $P_0 = (X, \delta_0, \{\mathtt{tpp}(\mathcal{T})\})$ with $\mathcal{T} = \{T_0, T_1\}$, $T_0 = \{t_{00}\}$, $T_1 = \{t_{10}\}$, $t_{00} = (\mathtt{S}_{00}, \mathtt{P}_{00})$, $t_{10} = (\mathtt{S}_{10}, \mathtt{P}_{10})$, $\delta_0(\mathtt{S}_{00}) = \{1,2,4\}$, $\delta_0(\mathtt{P}_{00}) = \{1,6\}$, $\delta_0(\mathtt{S}_{10}) = \{1,3,4,5\}$, and $\delta_0(\mathtt{P}_{10}) = \{1\}$. $\to_{\mathrm{PVSB}}$ applies two times:

1. $\rightarrow_{\mathrm{PVSB}}$ applies to t_{00} because $\mathrm{est}(\mathcal{T},\delta_0) = 1$, $\mathrm{lct}(\mathcal{T},\delta_0) = 5$, and thus $(s,6)$ is not feasible for all $s \in \delta_0(\mathtt{S}_{00})$. We obtain the problem P_1 with $\delta_1 = \delta_0$ except for $\delta_1(\mathtt{P}_{00}) = \{1\}$.

2. $\rightarrow_{\mathrm{PVSB}}$ applies to t_{10} because $\mathrm{lct}(\mathcal{T},\delta_1) = 4$ and thus $(5,p)$ is not feasible for all $p \in \delta_1(\mathtt{P}_{10})$. We obtain the problem P_2 with $\delta_2 = \delta_1$ except for $\delta_2(\mathtt{S}_{10}) = \{1,3,4\}$.

Note that, as demonstrated by Example 3.1, $\rightarrow_{\mathrm{PVS}}$ applies to P_2 while $\rightarrow_{\mathrm{PVSB}}$ does not.

Definition 3.6. Let $P_0 = (X,\delta_0,C)$ and $P_1 = (X,\delta_1,C)$ be FCSPs. We say that $P_0 \rightarrow_{\mathrm{FC}} P_1$ iff $P_0 \rightarrow_{\mathrm{FD}} P_1$ and $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, $t = (\mathtt{S},\mathtt{P}) \in T$, and $a \in \mathrm{vc}(\mathcal{T},\delta_0)$ exist s.t. $a \notin \mathrm{vc}(T,\delta_0)$, $a \in \mathrm{vs}(t,\delta_0)$, $a \notin \mathrm{vs}(u,\delta_0)$ for all $u \in T$, $u \neq t$, and $\delta_1 = \delta_0$ except for

$$\delta_1(\mathtt{S}) = \{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\, a \in [s, s+p-1]\}$$

and

$$\delta_1(\mathtt{P}) = \{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\, a \in [s, s+p-1]\}\,.$$

Example 3.3. Consider the problem $P_0 = (X,\delta_0,\{\mathtt{tpp}(\mathcal{T})\})$ with $\mathcal{T} = \{T_0,T_1\}$, $T_0 = \{t_{00}\}$, $T_1 = \{t_{10},t_{11}\}$, $t_{00} = (\mathtt{S}_{00},\mathtt{P}_{00})$, $t_{10} = (\mathtt{S}_{10},\mathtt{P}_{10})$, $t_{11} = (\mathtt{S}_{11},\mathtt{P}_{11})$, $\delta_0(\mathtt{S}_{00}) = [0,2]$, $\delta_0(\mathtt{P}_{00}) = \{5\}$, $\delta_0(\mathtt{S}_{10}) = \{0,3\}$, $\delta_0(\mathtt{P}_{10}) = \{1,2\}$, $\delta_0(\mathtt{S}_{11}) = \{1,3\}$, and $\delta_0(\mathtt{P}_{10}) = \{1,2\}$. $\rightarrow_{\mathrm{FC}}$ applies four times:

1. $\rightarrow_{\mathrm{FC}}$ applies to t_{11} because $2 \in \mathrm{vc}(\mathcal{T},\delta_0) = [2,4]$, $2 \notin \mathrm{vc}(T_1,\delta_0) = \emptyset$, $2 \in \mathrm{vs}(t_{11},\delta_0) = [1,4]$, and $2 \notin \mathrm{vs}(t_{10},\delta_0) = \{0,1,3,4\}$. We obtain the problem P_1 with $\delta_1 = \delta_0$ except for $\delta_1(\mathtt{S}_{11}) = \{1\}$ and $\delta_1(\mathtt{P}_{11}) = \{2\}$.

2. $\rightarrow_{\mathrm{FC}}$ applies to t_{00} because $1 \in \mathrm{vc}(\mathcal{T},\delta_1) = [1,4]$, $1 \notin \mathrm{vc}(T_0,\delta_1) = [2,4]$, $1 \in \mathrm{vs}(t_{00},\delta_1) = [0,6]$, and t_{00} is the only task in T_0. We obtain the problem P_2 with $\delta_2 = \delta_1$ except for $\delta_2(\mathtt{S}_{00}) = \{0,1\}$.

3. $\rightarrow_{\mathrm{FC}}$ applies to t_{10} because $3 \in \mathrm{vc}(\mathcal{T},\delta_2) = [1,4]$, $3 \notin \mathrm{vc}(T_1,\delta_2) = \{1,2\}$, $3 \in \mathrm{vs}(t_{10},\delta_2) = \{0,1,3,4\}$, and $3 \notin \mathrm{vs}(t_{11},\delta_2) = \{1,2\}$. We obtain the problem P_3 with $\delta_3 = \delta_2$ except for $\delta_3(\mathtt{S}_{10}) = \{3\}$.

4. $\rightarrow_{\mathrm{FC}}$ applies to t_{10} because $4 \in \mathrm{vc}(\mathcal{T},\delta_3) = [1,4]$, $4 \notin \mathrm{vc}(T_1,\delta_3) = [1,3]$, $4 \in \mathrm{vs}(t_{10},\delta_3) = \{3,4\}$, and $4 \notin \mathrm{vs}(t_{11},\delta_3) = \{1,2\}$. We obtain the problem P_4 with $\delta_4 = \delta_3$ except for $\delta_4(\mathtt{P}_{10}) = \{2\}$.

Definition 3.7. We say that $P_1 \in \mathrm{gs}(P_0)$ (P_1 is a ***ground successor*** to P_0) iff $P_0 \rightarrow_{\mathrm{FD}} P_1$ and P_1 is ground.

Corollary 3.2. *If* $P_0 \rightarrow_{\mathrm{FD}} P_1$, *then* $\mathrm{gs}(P_0) \supseteq \mathrm{gs}(P_1)$.

Definition 3.8. Let $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ be FCSPs. We say that $P_0 \rightarrow_{\mathrm{IPT}} P_1$ iff $P_0 \rightarrow_{\mathrm{FD}} P_1$, P_1 is failed, and $\mathtt{tpp}(\mathcal{T}) \in C$, $T_0, T_1 \in \mathcal{T}$, and $l, u \geq 0$ exist s.t., for all $(X, \gamma, C) \in \mathrm{gs}(P_0)$, l is a lower bound on $|\mathrm{vc}(T_0, \gamma)|$, u is an upper bound on $|\mathrm{vc}(T_1, \gamma)|$, and $u < l$.

Example 3.4. Consider the problem $(X, \delta, \{\mathtt{tpp}(\{T_0, T_1\})\})$ with $T_0 = \{t_{00}, t_{01}\}$ and $T_1 = \{t_{10}, t_{11}\}$ where $t_{00} = (\{0,5\}, \{2\})$, $t_{01} = (\{2,6\}, \{1,2,3\})$, $t_{10} = (\{2,3\}, \{4,5\})$, and $t_{11} = (\{0,6\}, \{2,3\})$. (To simplify matters, the variables have been replaced by their domains.) We note that $\mathrm{vs}(T_0, \delta) = \mathrm{vs}(T_1, \delta) = [0, 8]$ and that T_0 cannot cover more than five values. If the tasks of T_1 are allowed to overlap, T_1 has a schedule covering five values and $\rightarrow_{\mathrm{IPT}}$ does not apply. However, if the schedules of T_1 are required to be disjunctive, each of them will cover at least six values. In consequence, the tracks cannot be processed in parallel. $\rightarrow_{\mathrm{IPT}}$ will reveal this inconsistency if the demand for disjunctiveness is considered when computing the lower bound on the number of values covered by the schedules of T_1.

Definition 3.9. Let $P = (X, \delta, C)$ be a FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. If $T = \{t_1, \ldots t_n\} \in \mathcal{T}$, then $\mathrm{vcg}(\mathcal{T}, T, \delta)$ denotes the bipartite graph (U, V, E) with

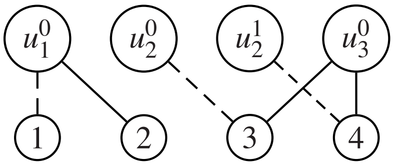
- $U = \displaystyle\bigcup_{\substack{1 \leq i \leq n \\ \delta(\mathtt{P}_i) \neq \emptyset}} \left\{ u_i^j : 0 \leq j < \max \delta(\mathtt{P}_i) \right\}$,
- $V = \mathrm{vc}(\mathcal{T}, \delta)$, and
- $E = \left\{ (u_i^j, a) : u_i^j \in U \wedge a \in V \wedge \exists s \in \delta(\mathtt{S}_i).\ s + j = a \right\}$.

We call this structure ***value-cover graph*** .

Definition 3.10. Let $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ be FCSPs. We say that $P_0 \rightarrow_{\mathrm{NC}} P_1$ iff $P_0 \rightarrow_{\mathrm{FD}} P_1$, P_1 is failed, and $\mathtt{tpp}(\mathcal{T}) \in C$ and $T \in \mathcal{T}$ exist s.t. $\mathrm{vcg}(\mathcal{T}, T, \delta_0) = (U, V, E)$ does not have a matching[3] M with $|M| = |V|$.

Example 3.5. Consider the problem $P = (X, \delta, \{\mathtt{tpp}(\{T_0, T_1\})\})$ with $T_1 = \{t_{10}, t_{11}, t_{12}\}$, $t_{10} = (\{1,2\}, \{1\})$, $t_{11} = (\{3\}, \{2\})$, and $t_{12} = (\{3,4\}, \{1\})$. (To simplify matters, the variables have been replaced by their domains.) Suppose $\mathrm{vc}(T_0, \delta) = \mathrm{vs}(T_0, \delta) = [1, 4]$. We note that $\mathrm{vs}(T_1, \delta) = [1, 4]$. Now consider the value-cover graph $\mathrm{vcg}(\{T_0, T_1\}, T_1, \delta)$:

[3] Given a bipartite graph (U, V, E), a ***matching*** is a subset of edges $M \subseteq E$ s.t., for all vertices $v \in U \cup V$, at most one edge of M is incident on v.



The dotted edges constitute a matching of cardinality 3 and it is easy to verify that it has maximum cardinality. Hence only three values out of $[1,4]$ can be covered simultaneously. $\rightarrow_{\mathrm{NC}}$ detects this inconsistency and signals a failure by transforming P into a failed FCSP.

3.3.2 Correctness

To demonstrate the correctness of a reduction $\rightarrow \subseteq \rightarrow_{\mathrm{FD}}$, we have to show that each of its transitions does not restrict the solution space. More precisely, we have to show that $\mathrm{sol}(P_0) = \mathrm{sol}(P_1)$ for $(P_0, P_1) \in \rightarrow$. Irrespective of $\rightarrow$, $\mathrm{sol}(P_1) \subseteq \mathrm{sol}(P_0)$ does not require any work as it follows easily from Lemma 1.1 with $P_0 \rightarrow_{\mathrm{FD}} P_1$. To show the other direction for the reductions proposed in the previous section, we derive a contradiction from the assumption that a solution to P_0 exists that does not solve P_1.

Proposition 3.2. $\rightarrow_{\mathrm{PVS}}$ *is correct.*

Proof. Let $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ be FCSPs with $P_0 \rightarrow_{\mathrm{PVS}} P_1$. We have to show that $\mathrm{sol}(P_0) = \mathrm{sol}(P_1)$. By Lemma 1.1, $\mathrm{sol}(P_1) \subseteq \mathrm{sol}(P_0)$ because $P_0 \rightarrow_{\mathrm{FD}} P_1$. To show that $\mathrm{sol}(P_0) \subseteq \mathrm{sol}(P_1)$, let $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, $t = (\mathtt{S}, \mathtt{P}) \in T$, and $a \in \mathrm{vs}(t, \delta_0)$ s.t. $a \notin \mathrm{vs}(\mathcal{T}, \delta_0)$ and $\delta_1 = \delta_0$ except for

$$\delta_1(\mathtt{S}) = \{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\ a \notin [s, s+p-1]\}$$

and

$$\delta_1(\mathtt{P}) = \{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\ a \notin [s, s+p-1]\}.$$

Obviously,

$$\mathrm{sol}(P_1) = \{\sigma \in \mathrm{sol}(P_0) : \mathtt{S}\sigma \in \delta_1(\mathtt{S}) \wedge \mathtt{P}\sigma \in \delta_1(\mathtt{P})\}.$$

Let $\sigma \in \mathrm{sol}(P_0)$ and $(s, p) = (\mathtt{S}\sigma, \mathtt{P}\sigma)$. Suppose $\sigma \notin \mathrm{sol}(P_1)$, or equivalently, $s \notin \delta_1(\mathtt{S}) \vee p \notin \delta_1(\mathtt{P})$. In either case, $a \in [s, s+p-1] = \mathrm{vs}(t, \sigma)$. By Lemma 3.1, $\mathrm{vs}(\mathcal{T}, \delta_0) \supseteq \mathrm{vs}(\mathcal{T}, \sigma)$. By Lemma 3.4, $\mathrm{vs}(\mathcal{T}, \sigma) = \mathrm{vs}(T, \sigma)$. Putting it all together we obtain the contradiction

$$a \notin \mathrm{vs}(\mathcal{T}, \delta_0) \supseteq \mathrm{vs}(\mathcal{T}, \sigma) = \mathrm{vs}(T, \sigma) = \bigcup_{t \in T} \mathrm{vs}(t, \sigma) \supseteq \mathrm{vs}(t, \sigma) = [s, s+p-1] \ni a.$$

$\square$

Proposition 3.3. $\rightarrow_{\text{PVSB}}$ *is correct.*

Proof. Let $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ be FCSPs with $P_0 \rightarrow_{\text{PVSB}} P_1$. We have to show that $\text{sol}(P_0) = \text{sol}(P_1)$. By Lemma 1.1, $\text{sol}(P_1) \subseteq \text{sol}(P_0)$ because $P_0 \rightarrow_{\text{FD}} P_1$. To show that $\text{sol}(P_0) \subseteq \text{sol}(P_1)$, let $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, and $t \in T$ s.t. $\delta_1 = \delta_0$ except for

$$\delta_1(\mathtt{S}) = \{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\ (s,p) \text{ is feasible}\}$$

and

$$\delta_1(\mathtt{P}) = \{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\ (s,p) \text{ is feasible}\}$$

Obviously,

$$\text{sol}(P_1) = \{\sigma \in \text{sol}(P_0) : \mathtt{S}\sigma \in \delta_1(\mathtt{S}) \wedge \mathtt{P}\sigma \in \delta_1(\mathtt{P})\}.$$

Let $\sigma \in \text{sol}(P_0)$ and $(s,p) = (\mathtt{S}\sigma, \mathtt{P}\sigma)$. Suppose $\sigma \notin \text{sol}(P_1)$, or equivalently, $s \notin \delta_1(\mathtt{S}) \vee p \notin \delta_1(\mathtt{P})$. In either case, (s,p) is not feasible. Before examining the single cases, it is useful to picture some facts.

- Obviously, $s = \min \sigma(\mathtt{S})$ and $p = \min \sigma(\mathtt{P})$.
- By Corollary 1.1, $\min \sigma(\mathtt{S}) \geq \min \delta_0(\mathtt{S})$ and $\min \sigma(\mathtt{P}) \geq \min \delta_0(\mathtt{P})$ because $P_0 \rightarrow_{\text{FD}} (X, \sigma, C)$.
- By Lemma 3.2, $\text{est}(\mathcal{T}, \delta_0) \leq \text{est}(\mathcal{T}, \sigma)$ and $\text{lct}(\mathcal{T}, \delta_0) \geq \text{lct}(\mathcal{T}, \sigma)$ because $P_0 \rightarrow_{\text{FD}} (X, \sigma, C)$.
- By Lemma 3.4, $\text{est}(\mathcal{T}, \sigma) = \text{est}(T, \sigma)$ and $\text{lct}(\mathcal{T}, \sigma) = \text{lct}(T, \sigma)$ because σ is a solution.
- By Definition 3.3, $\text{est}(T, \sigma) = \min_{t \in T} \text{est}(t, \sigma) \leq \text{est}(t, \sigma) = \min \sigma(\mathtt{S})$.

We have to consider three cases.

1. $s < \text{est}(\mathcal{T}, \delta_0)$: From the facts we collected, we obtain the contradiction

$$\text{est}(\mathcal{T}, \sigma) = \text{est}(T, \sigma) \leq \min \sigma(\mathtt{S}) = s < \text{est}(\mathcal{T}, \delta_0) \leq \text{est}(\mathcal{T}, \sigma).$$

2. $s > \text{lct}(\mathcal{T}, \delta_0) - \min \delta_0(\mathtt{P}) + 1$: We note that

$$\begin{aligned} \text{lct}(t, \sigma) &= s + p - 1 \\ &> \text{lct}(\mathcal{T}, \delta_0) - \min \delta_0(\mathtt{P}) + p \\ &= \text{lct}(\mathcal{T}, \delta_0) - \min \delta_0(\mathtt{P}) + \min \sigma(\mathtt{P}) \\ &\geq \text{lct}(\mathcal{T}, \delta_0) \end{aligned}$$

and hence

$$\text{lct}(\mathcal{T}, \sigma) = \text{lct}(T, \sigma) = \max_{t \in T} \text{lct}(t, \sigma) \geq \text{lct}(t, \sigma) > \text{lct}(\mathcal{T}, \delta_0) \geq \text{lct}(\mathcal{T}, \sigma).$$

3. From the facts we collected, we obtain

$$\begin{aligned} p &> \text{lct}(\mathcal{T},\delta_0) - \max\{\min\delta_0(\mathtt{S}), \text{est}(\mathcal{T},\delta_0)\} + 1 \\ &\geq \text{lct}(\mathcal{T},\sigma) - \max\{\min\sigma(\mathtt{S}), \text{est}(\mathcal{T},\sigma)\} + 1 \\ &= \text{lct}(\mathcal{T},\sigma) - \max\{\min\sigma(\mathtt{S}), \text{est}(T,\sigma)\} + 1 \\ &= \text{lct}(\mathcal{T},\sigma) - \min\sigma(\mathtt{S}) + 1 \\ &= \text{lct}(\mathcal{T},\sigma) - s + 1 \end{aligned}$$

It follows that $\text{lct}(t,\sigma) = s + p - 1 > \text{lct}(\mathcal{T},\sigma)$. Using this relationship, we obtain the contradiction

$$\text{lct}(\mathcal{T},\sigma) = \text{lct}(T,\sigma) = \max_{t\in T}\text{lct}(t,\sigma) \geq \text{lct}(t,\sigma) > \text{lct}(\mathcal{T},\sigma).$$

□

Proposition 3.4. $\rightarrow_{\text{FC}}$ *is correct.*

Proof. Let $P_0 = (X,\delta_0,C)$ and $P_1 = (X,\delta_1,C)$ be FCSPs with $P_0 \rightarrow_{\text{FC}} P_1$. We have to show that $\text{sol}(P_0) = \text{sol}(P_1)$. By Lemma 1.1, $\text{sol}(P_1) \subseteq \text{sol}(P_0)$ because $P_0 \rightarrow_{\text{FD}} P_1$. To show that $\text{sol}(P_0) \subseteq \text{sol}(P_1)$, let $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, $t = (\mathtt{S},\mathtt{P}) \in T$, and $a \in \text{vc}(\mathcal{T},\delta_0)$ s.t. $a \notin \text{vc}(T,\delta_0)$, $a \in \text{vs}(t,\delta_0)$, $a \notin \text{vs}(u,\delta_0)$ for all $u \in T$, $u \neq t$, and $\delta_1 = \delta_0$ except for

$$\delta_1(\mathtt{S}) = \{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\ a \in [s, s+p-1]\}$$

and

$$\delta_1(\mathtt{P}) = \{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\ a \in [s, s+p-1]\}.$$

Obviously,

$$\text{sol}(P_1) = \{\sigma \in \text{sol}(P_0) : \mathtt{S}\sigma \in \delta_1(\mathtt{S}) \wedge \mathtt{P}\sigma \in \delta_1(\mathtt{P})\}.$$

Let $\sigma \in \text{sol}(P_0)$ and $(s,p) = (\mathtt{S}\sigma, \mathtt{P}\sigma)$. Suppose $\sigma \notin \text{sol}(P_1)$, or equivalently, $s \notin \delta_1(\mathtt{S}) \vee p \notin \delta_1(\mathtt{P})$. In either case, $a \notin [s, s+p-1] = \text{vc}(t,\sigma)$. Moreover, for $u \in T$ with $u \neq t$, we have $a \notin \text{vs}(u,\delta_0) \supseteq \text{vs}(u,\sigma) = \text{vc}(u,\sigma)$ by Lemma 3.1 and by Lemma 3.4. Now, because no task in T can cover a, $a \notin \text{vc}(T,\sigma)$. On the other hand, we have $a \in \text{vc}(\mathcal{T},\delta_0) \subseteq \text{vc}(\mathcal{T},\sigma) = \text{vc}(T,\sigma)$ by Lemma 3.1 and by Lemma 3.4. □

Proposition 3.5. $\rightarrow_{\text{IPT}}$ *is correct.*

Proof. Let $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ be FCSPs with $P_0 \rightarrow_{\mathrm{IPT}} P_1$. We have to show that $\mathrm{sol}(P_0) = \mathrm{sol}(P_1)$. $\mathrm{sol}(P_1) = \emptyset$ because P_1 is failed. Let $\sigma \in \mathrm{sol}(P_0)$. We know that $\mathtt{tpp}(\mathcal{T}) \in C$, $T_0, T_1 \in \mathcal{T}$, and $l, u \geq 0$ exist s.t. l is a lower bound on $|\mathrm{vc}(T_0, \sigma)|$, u is an upper bound on $|\mathrm{vc}(T_1, \sigma)|$, and $u < l$. Furthermore, by Lemma 3.4, we know that $\mathrm{vc}(T_0, \sigma) = \mathrm{vc}(T_1, \sigma) = \mathrm{vc}(\mathcal{T}, \sigma)$. Putting it all together we obtain the contradiction

$$|\mathrm{vc}(\mathcal{T}, \sigma)| = |\mathrm{vc}(T_1, \sigma)| \leq u < l \leq |\mathrm{vc}(T_0, \sigma)| = |\mathrm{vc}(\mathcal{T}, \sigma)|.$$

Hence $\mathrm{sol}(P_0)$ is empty and equals $\mathrm{sol}(P_1)$. □

Lemma 3.5. *Let $P = (X, \delta, C)$ be a FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. Let $T \in \mathcal{T}$. If $\mathrm{sol}(P) \neq \emptyset$, then $\mathrm{vcg}(\mathcal{T}, T, \delta)$ has a matching M with $|M| = |V|$.*

Proof. Let $(U, V, E) = \mathrm{vcg}(\mathcal{T}, T, \delta)$ (cf. Definition 3.9), $\sigma \in \mathrm{sol}(P)$, and

$$M = \left\{ (u_i^{a - \mathtt{S}_i\sigma}, a) : 1 \leq i \leq n \wedge a \in V \wedge a \in \mathrm{vc}(t_i, \sigma) \wedge \forall 1 \leq j < i.\ a \notin \mathrm{vc}(t_j, \sigma) \right\}.$$

We will show that $M \subseteq E$, M is a matching, and $|M| = |V|$.

1. $M \subseteq E$: Let $(u_i^j, a) \in M$. From the definition of M, we know that $1 \leq i \leq n$, $j = a - \mathtt{S}_i\sigma$, $a \in V$, and $a \in \mathrm{vc}(t_i, \sigma)$. First, we have to show that $s \in \delta(\mathtt{S}_i)$ exists s.t. $s + j = a$. Let $s = \mathtt{S}_i\sigma$. $s + j = \mathtt{S}_i\sigma + a - \mathtt{S}_i\sigma = a$ and, by Lemma 1.1, $s \in \delta(\mathtt{S}_i)$. It remains to show that $u_i^j \in U$, or equivalently, $0 \leq j < \max \delta(\mathtt{P}_i)$. From $a \in \mathrm{vc}(t_i, \sigma) = [\mathtt{S}_i\sigma, \mathtt{S}_i\sigma + \mathtt{P}_i\sigma - 1]$ and by Lemma 1.1, we conclude that
$$0 \leq a - \mathtt{S}_i\sigma = j \leq \mathtt{P}_i\sigma - 1 < \mathtt{P}_i\sigma = \max \sigma(\mathtt{P}_i) \leq \max \delta(\mathtt{P}_i).$$
2. M is a matching: Let $(u_i^j, a) \in M$.
   (a) u_i^j is the only mate of a in M: Suppose $(u_k^l, a) \in M$ s.t. $(k, l) \neq (i, j)$. We note that $j = l = a - \mathtt{S}_i\sigma$ which implies $k \neq i$. Suppose $k < i$. Then $(u_i^j, a) \notin M$ because $a \in \mathrm{vc}(t_k, \sigma)$. Suppose $k > i$. Then $(u_k^l, a) \notin M$ because $a \in \mathrm{vc}(t_i, \sigma)$.
   (b) a is the only mate of u_i^j in M: Suppose $(u_i^j, b) \in M$ s.t. $b \neq a$. However, by the definition of M, $a = j + \mathtt{S}_i\sigma = b$.
3. $|M| = |V|$: Let $a \in V = \mathrm{vc}(\mathcal{T}, \delta)$. By Lemma 3.1, $a \in \mathrm{vc}(\mathcal{T}, \sigma)$ and, by Lemma 3.4, $a \in \mathrm{vc}(T, \sigma)$. We have to show that $1 \leq i \leq n$ exists s.t. $(u_i^{a - \mathtt{S}_i\sigma}, a) \in M$. Let $I = \{1 \leq i \leq n : a \in \mathrm{vc}(t_i, \sigma)\}$. $I \neq \emptyset$ because otherwise $a \notin \mathrm{vc}(T, \sigma) = \bigcup_{1 \leq i \leq n} \mathrm{vc}(t_i, \sigma)$. Let $i = \min I$. It is easy to verify that $(u_i^{a - \mathtt{S}_i\sigma}, a) \in M$. Furthermore, $u_i^{a - \mathtt{S}_i\sigma}$ is the only mate of a in M because M is a matching. Thus $|M| = |V|$.

□

Proposition 3.6. *$\rightarrow_{\mathrm{NC}}$ is correct.*

Proof. Let $P_0 = (X, \delta_0, C)$ and $P_1 = (X, \delta_1, C)$ be FCSPs with $P_0 \rightarrow_{\mathrm{NC}} P_1$. We have to show that $\mathrm{sol}(P_0) = \mathrm{sol}(P_1)$. $\mathrm{sol}(P_1) = \emptyset$ because P_1 is failed. Suppose $\mathrm{sol}(P_0) \neq \emptyset$. By Lemma 3.5, $\mathrm{vcg}(\mathcal{T}, T, \delta_0) = (U, V, E)$ has a matching M with $|M| = |V|$ for all $\mathtt{tpp}(\mathcal{T}) \in C$ and $T \in \mathcal{T}$. This contradicts the definition of $\rightarrow_{\mathrm{NC}}$. □

Corollary 3.3. *If $R \subseteq \{\rightarrow_{\mathrm{PVS}}, \rightarrow_{\mathrm{PVSB}}, \rightarrow_{\mathrm{FC}}, \rightarrow_{\mathrm{IPT}}, \rightarrow_{\mathrm{NC}}\}$ and $\rightarrow_R = \bigcup R$, then $\rightarrow_R$ is correct.*

3.3.3 Convergence

A reduction is called convergent if it is terminating and confluent. Termination guarantees that there are no infinite computations; confluence guarantees that every element of the underlying set has at most one normal form. With a convergent reduction system, whatever path of computation is actually taken for a given input (there may be many due to non-determinism), the result is uniquely determined (c.f. Section 1.5).

Each of the reductions for solving `tpp` constraints is terminating because each of them is a subset of $\rightarrow_{\mathrm{FD}}$ and $\rightarrow_{\mathrm{FD}}$ is terminating itself. The following statement is an easy consequence.

Proposition 3.7. *If $R \subseteq \{\rightarrow_{\mathrm{PVS}}, \rightarrow_{\mathrm{PVSB}}, \rightarrow_{\mathrm{FC}}, \rightarrow_{\mathrm{IPT}}, \rightarrow_{\mathrm{NC}}\}$ and $\rightarrow_R = \bigcup R$, then $\rightarrow_R$ is terminating.*

To obtain the following convergence result, a considerable amount of work is required. Consult Appendix A or [Mar02] for the details.

Proposition 3.8. *If $R \subseteq \{\rightarrow_{\mathrm{PVS}}, \rightarrow_{\mathrm{FC}}, \rightarrow_{\mathrm{IPT}}, \rightarrow_{\mathrm{NC}}\}$ and $\rightarrow_R = \bigcup R$, then $\rightarrow_R$ is convergent.*

A corresponding result for $\rightarrow_{\mathrm{PVSB}}$ is currently not available.

The following statement on normal forms follows easily with Lemma 1.1.

Corollary 3.4. *Let P be a FCSP. If $R \subseteq \{\rightarrow_{\mathrm{PVS}}, \rightarrow_{\mathrm{FC}}, \rightarrow_{\mathrm{IPT}}, \rightarrow_{\mathrm{NC}}\}$ and $\rightarrow_R = \bigcup R$, then P has a uniquely determined normal form wrt. $\rightarrow_R$.*

3.3.4 Other Performance Guarantees

To show that $\rightarrow_{\mathrm{PVS}}$ is a solver for `tpp` constraints (in the sense of Definition 1.3), it remains to verify that $\rightarrow_{\mathrm{PVS}}$ recognizes constraint violations. We start out from a ground, unfailed FCSP P_0 that contains at least one violated `tpp` constraint. In the first place, we show that there is at least one way to apply $\rightarrow_{\text{PVS}}$ to P_0 that results in a failed FCSP. In a second step, we combine the first result with Lemma 1.1 to demonstrate that the normal form of P_0 wrt. $\rightarrow_{\text{PVS}}$ is necessarily failed.

Proposition 3.9. *Let $P_0 = (X, \delta_0, C)$ be a ground, unfailed FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. If δ_0 violates $\mathtt{tpp}(\mathcal{T})$, then a failed FCSP P_1 exists s.t. $P_0 \rightarrow_{\text{PVS}} P_1$.*

Proof. We know that $|\{\mathrm{vc}(T, \delta_0) : T \in \mathcal{T}\}| > 1$, because $\mathcal{T} \neq \emptyset$ and δ_0 violates $\mathtt{tpp}(\mathcal{T})$. Let $T_0, T_1 \in \mathcal{T}$ s.t. $\mathrm{vc}(T_0, \delta_0) \neq \mathrm{vc}(T_1, \delta_0)$. If $\mathrm{vc}(T_0, \delta_0) \subset \mathrm{vc}(T_1, \delta_0)$, let $a \in \mathrm{vc}(T_1, \delta_0) \setminus \mathrm{vc}(T_0, \delta_0)$ and $t = (\mathtt{S}, \mathtt{P}) \in T_1$ s.t. $a \in \mathrm{vc}(t, \delta_0)$. Otherwise, let $a \in \mathrm{vc}(T_0, \delta_0) \setminus \mathrm{vc}(T_1, \delta_0)$ and $t = (\mathtt{S}, \mathtt{P}) \in T_0$ s.t. $a \in \mathrm{vc}(t, \delta_0)$. Let $P_1 = (X, \delta_1, C)$ with $\delta_1 = \delta_0$ except for $\delta_1(\mathtt{S}) = \delta_1(\mathtt{P}) = \emptyset$. Obviously, P_1 is failed and $P_0 \rightarrow_{\text{FD}} P_1$. It remains to verify that $a \in \mathrm{vs}(t, \delta_0)$, $a \notin \mathrm{vs}(\mathcal{T}, \delta_0)$,

$$\{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\ a \notin [s, s + p - 1]\} = \emptyset$$

and

$$\{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\ a \notin [s, s + p - 1]\} = \emptyset.$$

The latter equations hold because $\mathtt{S}\delta_0$ and $\mathtt{P}\delta_0$ are the only candidates and, by assumption, $a \in \mathrm{vc}(t, \delta_0) = [\mathtt{S}\delta_0, \mathtt{S}\delta_0 + \mathtt{P}\delta_0 - 1]$. $a \in \mathrm{vs}(t, \delta_0)$ because, by Lemma 3.4, $\mathrm{vs}(t, \delta_0) = \mathrm{vc}(t, \delta_0)$ and, by assumption, $a \in \mathrm{vc}(t, \delta_0)$. $a \notin \mathrm{vs}(\mathcal{T}, \delta_0)$ because, by Definition 3.1 and by Lemma 3.4,

$$\mathrm{vs}(\mathcal{T}, \delta_0) = \bigcap_{T \in \mathcal{T}} \mathrm{vs}(T, \delta_0) \subseteq \mathrm{vs}(T_0, \delta_0) \cap \mathrm{vs}(T_1, \delta_0) = \mathrm{vc}(T_0, \delta_0) \cap \mathrm{vc}(T_1, \delta_0)$$

and a has been chosen s.t. $a \notin \mathrm{vc}(T_0, \delta_0) \cap \mathrm{vc}(T_1, \delta_0)$. □

Proposition 3.10. *Let $R \subseteq \{\rightarrow_{\text{PVS}}, \rightarrow_{\text{FC}}, \rightarrow_{\text{IPT}}, \rightarrow_{\text{NC}}\}$ s.t. $\rightarrow_{\text{PVS}} \in R$ and let $\rightarrow_R = \bigcup R$. Let $P_0 = (X, \delta_0, C)$ be a ground, unfailed FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$. If δ_0 violates $\mathtt{tpp}(\mathcal{T})$, then $P_0{\downarrow_R}$ is failed.*

Proof. By Proposition 3.9, a failed FCSP P_1 exists s.t. $P_0 \rightarrow_{\text{PVS}} P_1$. From Corollary 3.4, we know that P_0 and P_1 have uniquely determined normal forms wrt. $\rightarrow_R$, namely $P_0{\downarrow_R}$ and $P_1{\downarrow_R}$. So we have $P_0{\downarrow_R} \leftarrow_R^* P_0 \rightarrow_{\text{PVS}} P_1 \rightarrow_R^* P_1{\downarrow_R}$ and we see that $P_1{\downarrow_R}$ is a normal form of P_0 wrt. $\rightarrow_R$. It follows that $P_1{\downarrow_R} = P_0{\downarrow_R}$ and thus $P_1 \rightarrow_R^* P_0{\downarrow_R}$. Since $\rightarrow_R \subseteq \rightarrow_{\text{FD}}$, $P_0{\downarrow_R}$ is failed like its predecessor P_1. □

So we have a guarantee that violated tpp constraint will be detected by $\rightarrow_{\text{PVS}}$. This will happen at the latest when all variables have become ground. There is no guarantee, however, that violations will be detected as soon as possible. Note that such a solver would implement a decision procedure for the TPP and, unless P = NP, it would have exponential time complexity because the TPP is NP-complete.

Next we show that $\rightarrow_{\text{NC}}$ is a solver for tpp constraints. We proceed in the same manner as before.

Proposition 3.11. *Let $P_0 = (X, \delta_0, C)$ be a ground, unfailed FCSP with* $\mathtt{tpp}(\mathcal{T}) \in C$. *If δ_0 violates* $\mathtt{tpp}(\mathcal{T})$, *then a failed FCSP P_1 exists s.t. $P_0 \rightarrow_{\mathrm{NC}} P_1$.*

Proof. We know that $|\{\mathrm{vc}(T, \delta_0) : T \in \mathcal{T}\}| > 1$, because $\mathcal{T} \neq \emptyset$ and δ_0 violates $\mathtt{tpp}(\mathcal{T})$. Let $T_0, T_1 \in \mathcal{T}$ s.t. $\mathrm{vc}(T_0, \delta_0) \neq \mathrm{vc}(T_1, \delta_0)$. Let $G = (U, V, E) = \mathrm{vcg}(\mathcal{T}, T_0, \delta_0)$ and let M be a matching in G. Considering the structure of $\mathrm{vcg}(\mathcal{T}, T_0, \delta_0)$ (cf. Definition 3.9), we find that

$$\{a : \exists u \in U.\, (u, a) \in E\} = \mathrm{vc}(T_0, \delta_0).$$

Thus, $|\mathrm{vc}(T_0, \delta_0)|$ is an upper on $|M|$. We obtain

$$|M| \leq |\mathrm{vc}(T_0, \delta_0)| < |\mathrm{vc}(T_0, \delta_0) \cup \mathrm{vc}(T_1, \delta_0)| \leq \left| \bigcup_{T \in \mathcal{T}} \mathrm{vc}(T, \delta_0) \right| = |\mathrm{vc}(\mathcal{T}, \delta_0)| = |V|$$

and conclude that G does not have a matching M with $|M| = |V|$. Thus, for any failed FCSP P_1 s.t. $P_0 \rightarrow_{\mathrm{FD}} P_1$, $P_0 \rightarrow_{\mathrm{NC}} P_1$. $\square$

Proposition 3.12. *Let $R \subseteq \{\rightarrow_{\mathrm{PVS}}, \rightarrow_{\mathrm{FC}}, \rightarrow_{\mathrm{IPT}}, \rightarrow_{\mathrm{NC}}\}$ s.t. $\rightarrow_{\mathrm{NC}} \in R$ and let $\rightarrow_R = \bigcup R$. Let $P_0 = (X, \delta_0, C)$ be a ground, unfailed FCSP with* $\mathtt{tpp}(\mathcal{T}) \in C$. *If δ_0 violates* $\mathtt{tpp}(\mathcal{T})$, *then $P_0 \downarrow_R$ is failed.*

Proof. Similar to the proof of Proposition 3.10 by employing Proposition 3.11. $\square$

Note that $\rightarrow_{\mathrm{FC}}$ and $\rightarrow_{\mathrm{IPT}}$ are not solvers for `tpp` constraints. $\rightarrow_{\mathrm{FC}}$ simply does not apply to ground problems and $\rightarrow_{\mathrm{IPT}}$ does not complain about tracks with different value covers as long as the number of values covered is the same. So both $\rightarrow_{\mathrm{FC}}$ and $\rightarrow_{\mathrm{IPT}}$ may let constraint violations go unnoticed. Despite this shortcoming, there are two ways to employ $\rightarrow_{\mathrm{FC}}$ and $\rightarrow_{\mathrm{IPT}}$: They can be used to propagate redundant `tpp` constraints or in combination with the other reductions to compute more consequences.

3.4 Related Work

To our best knowledge, the TPP has not yet been described. In terminology, the TPP is related to the ***track assignment problem*** (TAP) that has been investigated in operations research. A TAP is stated in terms of a task set and a track set. Tasks are specified by intervals, i.e. start and processing times are fixed. A track is either a single machine [BN94, SL94, KL95] or a bank of identical parallel machines [FKN99] with time windows of availability. A schedule is an assignment of tasks to tracks that respects the capacity and the availability of the machines. Objectives include the minimization of the number of used tracks [SL94] and the maximization of the number of scheduled tasks [BN94, FKN99]. Applications include the assignment of transmission tasks to satellites [FKN99] and the assignment of clients to summer cottages [KL95].

The TPP differs from the TAP in several aspects. The chief difference is that solving a TAP requires to assign tracks while solving a TPP requires to assign times s.t. tracks are processed in parallel. Furthermore, a TAP includes restrictions on the availability of machines and the implicit assumption that each task requires a single machine for processing. In a constraint-programming fashion, the TPP does not impose any constraint except for that tracks have to be processed in parallel.

Chapter 4

Constraint-Based Solvers for School Timetabling

Research in automated school timetabling can be traced back to the 1960s and, during the last decade, efforts concentrated on greedy algorithms and on local search methods such as simulated annealing, tabu search, and genetic algorithms. Constraint-programming technology has been used to solve timetabling problems from universities but the question whether it applies to school timetabling as well is open (c.f. Section 2.3). This chapter takes a long stride towards an answer by proposing, investigating, and comparing constraint-based solvers for school timetabling.

The approach proposed in this chapter is based on a transformation of high-level problem descriptions (in terms of teachers, students, facilities, and meetings, c.f. Section 2.1) to constraint models (in terms of finite constraint networks, c.f. Section 1.6). This transformation is distinguished by its use of global constraints and deals with all common requirements of timetables (c.f. Section 2.1) except for bounds on idle time. Fixing the search strategy and the constraint-propagation algorithms yields a first solver for school-timetabling problems.

By a series of additional post-processing steps to the fundamental transformation, several solvers are obtained from the primal solver. Post processing is grounded on track parallelization problems (cf. Chapter 3) which are used to down-size the constraint models (by an additional transformation a priori to search) and to prune the search space (by propagation during search).

To study the operational properties of the primal solver and to compare it to its offspring, a large-scale computational experiment has been performed. The top priority in experimental design was to obtain results that are practically relevant and that allow for statistical inference. Inspired by a study of Drexl & Salewski [DS97], a configurable problem generator was developed, field work was performed to obtain configurations (i.e. detailed school descriptions), and for each of ten representative schools a sample of 1000 problems has been generated.

This chapter is organized as follows. Section 4.1 introduces finite-domain constraints that are relevant to this chapter, namely arithmetic constraints, the global cardinality constraint, and a constraint to place rectangles without overlaps. Section 4.2 presents the fundamental transformation of high-level problem specifications to constraint models. Sections 4.3 through 4.6 show how to obtain track parallelization problems and how to exploit them to obtain leaner constraint models. Section 4.7 reports the computational study (in terms of objectives, experimental design, and results) that was performed to investigate and to compare the various solvers. Section 4.8 closes the chapter with a review of related work.

4.1 Constraints for School Timetabling

This section introduces finite-domain constraints that will be used to model school-timetabling problems.

4.1.1 Arithmetic Constraints

An ***arithmetic constraint*** takes the form $e_1 \circ e_2$ where e_1 and e_2 are integer-valued arithmetic expressions over finite-domain variables and $\circ$ is a binary relation symbol like $=$ and $\leq$.

4.1.2 The Global Cardinality Constraint

Let X be a set of finite-domain variables that take their values in a set V. For each value in V, the ***global cardinality constraint*** (`gcc`) [Rég96] allows to constrain the frequency of its assignment to variables in X. The constraints on frequency are specified by a set of pairs of the form $(v, \mathtt{F})$ were the domain of the ***frequency variable*** `F` defines the frequencies that are admissible for the value v.

Definition 4.1. Let (X, δ, C) be a ground FCSP with

$$c = \mathtt{gcc}(\{\mathtt{X}_1, \ldots, \mathtt{X}_n\}, \{(v_1, \mathtt{F}_1), \ldots, (v_m, \mathtt{F}_m)\}) \in C.$$

δ satisfies c iff

$$\forall 1 \leq i \leq n.\ \exists 1 \leq j \leq m.\ \mathtt{X}_i\delta = v_j$$

and

$$\forall 1 \leq j \leq m.\ \left|\left\{1 \leq i \leq n : \mathtt{X}_i\delta = v_j\right\}\right| = \mathtt{F}_j\delta.$$

The `alldiff` constraint [Rég94, MT00, vH01] is well-known special case of the global cardinality constraint. It takes a set of finite-domain variables and requires that the variables are assigned pairwise different values.

4.1.3 Non-Overlapping Placement of Rectangles

We are given n sets of rectangles: For $1 \leq i \leq n$, we are given a set of origins $X_i \times Y_i$ and a set of sizes $W_i \times H_i$. We are required to select n rectangles s.t. the rectangles do not overlap pairwise. More precisely, for $1 \leq i \leq n$, we are required to find an origin $(x_i, y_i) \in X_i \times Y_i$ and a size $(w_i, h_i) \in W_i \times H_i$ s.t.

$$\forall 1 \leq i < j \leq n.\, x_i + w_i \leq x_j \vee x_j + w_j \leq x_i \vee y_i + h_i \leq y_j \vee y_j + h_j \leq y_i.$$

By a straightforward polynomial transformation from problem SS1 of Garey & Johnson [GJ79] (sequencing with release times and deadlines) we can show that this kind of placement problem is NP-complete.

To solve this kind of placement problem in a finite-domain constraint-programming framework, we use the `disjoint` constraint. The `disjoint` constraint takes one argument: a set

$$\{(\mathtt{X}_1, \mathtt{Y}_1, \mathtt{W}_1, \mathtt{H}_1), \ldots, (\mathtt{X}_n, \mathtt{Y}_n, \mathtt{W}_n, \mathtt{H}_n)\}$$

of tuples of finite-domain variables.

Definition 4.2. Let (X, δ, C) be a ground FCSP with

$$c = \mathtt{disjoint}(\{(\mathtt{X}_1, \mathtt{Y}_1, \mathtt{W}_1, \mathtt{H}_1), \ldots, (\mathtt{X}_n, \mathtt{Y}_n, \mathtt{W}_n, \mathtt{H}_n)\}) \in C.$$

For $1 \leq i \leq n$, let $x_i = \mathtt{X}_i\delta$, $y_i = \mathtt{Y}_i\delta$, $w_i = \mathtt{W}_i\delta$, and $h_i = \mathtt{H}_i\delta$. δ satisfies c iff the arrangement specified by $(x_i)_{1 \leq i \leq n}$, $(y_i)_{1 \leq i \leq n}$, $(w_i)_{1 \leq i \leq n}$, and $(h_i)_{1 \leq i \leq n}$ satisfies the requirement stated above.

4.2 A Basic Model Generator

This section shows how to transform school-timetabling problems into finite constraint networks by using global constraints. The following notation will be used:

Notation 4.1. Let P be a school-timetabling problem (c.f. Section 2.1).

- $S(P)$, $T(P)$, $R(P)$, $M(P)$, and $J(P)$ denote the students, the teachers, the rooms, the meetings, and the jobs of P.
- $S(m)$, $T(m)$, $R(m)$, and $p(m)$ denote the students, the teachers, the rooms, and the duration (the ***processing time***) of meeting m.
- $M(s)$ and $D(s)$ denote the meetings and the down time of student s.
- $M(t)$ and $D(t)$ denote the meetings and the down time of teacher t.

- $D(r)$ denotes the down time of room r.
- $M(j)$ denotes the meetings of job j.

In later parts of this thesis, the model generator specified in this place will be referred to as μ_0.

4.2.1 Representation of Time Slots

We assume that the (rectangular) time grid of P is derived from its width $w(P)$ (the number of days) and its height $h(P)$ (the number of time slots a day):

$$G(P) = [0, w(P) - 1] \times [0, h(P) - 1]$$

Hence $(0,0)$ represents the first time slot of the first day, $(0, h(P) - 1)$ represents the last time slot of the first day, $(1,0)$ represents the first time slot of the second day, and so on.

This representation is convenient but it cannot be used in finite-domain constraint programming where variables range over finite sets of integers. This problem can be solved easily by mapping (i, j) to $i * h(P) + j$. So the first time slot of the first day is represented by 0, the last time slot of the first day is represented by $h(P) - 1$, the first time slot of the second day is represented by $h(P)$, and so on.

4.2.2 Variables and Initial Domains

For each meeting $m \in M(P)$, three finite-domain variables are introduced: a period-level start-time variable $\mathtt{S}(m)$, a day-level start-time variable $\bar{\mathtt{S}}(m)$, and a room variable $\mathtt{R}(m)$. The set-up procedure takes the following steps:

1. The domain of $\mathtt{S}(m)$ is initialized with the time grid of P under consideration of the relevant down times

   $$D(m) = \bigcup \left\{ \bigcup_{s \in S(m)} D(s), \bigcup_{t \in T(m)} D(t), \bigcap_{r \in R(m)} D(r) \right\},$$

   the day boundaries, and the duration of m:

   $$(i, j) \in \delta(\mathtt{S}(m)) \leftrightarrow \forall j \leq k < j + p(m).\ (i,k) \in G(P) \setminus D(m)$$

2. The initial domain of $\bar{\mathtt{S}}(m)$ is obtained from the initial domain of $\mathtt{S}(m)$ by projection:

   $$\delta(\bar{\mathtt{S}}(m)) = \{i : (i, j) \in \delta(\mathtt{S}(m))\}$$

3. To propagate changes in the domain of $\mathtt{S}(m)$ to the domain of $\bar{\mathtt{S}}(m)$ and vice versa, the following constraint is imposed:

$$\bar{\mathtt{S}}(m) = \lfloor \mathtt{S}(m)/h(P) \rfloor$$

4. The domain of $\mathtt{R}(m)$ is initialized with the rooms that are suitable for m:

$$\delta(\mathtt{R}(m)) = R(m)$$

4.2.3 Symmetry Exclusion

Let j be a job in $J(P)$. If the meetings of j are identical with regard to their resource requirements and their durations, then an arbitrary total order is imposed on them to exclude symmetries and to reduce the search space. More precisely, if $m_1, \ldots, m_n$ are the meetings of j, then the constraint $\mathtt{S}(m_i) + p(m_i) \leq \mathtt{S}(m_{i+1})$ is imposed for $1 \leq i < n$. If the meetings differ in resource requirements or duration, then symmetry exclusion is performed separately for each maximum subset that satisfies the prerequisites.

4.2.4 Couplings

Let $J \in C(P)$ be a set of jobs to be scheduled in parallel. To enforce the coupling J, a `tpp` constraint is employed where each job constitutes a track and each meeting constitutes a task:

$$\mathtt{tpp}(\{\{(\mathtt{S}(m), p(m)) : m \in M(j)\} : j \in J\})$$

4.2.5 Resource Capacities

To avoid the double-booking of students and teachers, `alldiff` constraints are used. The procedure is demonstrated on the example of a teacher t. For each meeting $m \in M(t)$, the following steps are taken:

1. m is decomposed into $p(m)$ meetings $m_1, \ldots, m_{p(m)}$ with $p(m_k) = 1$ for $1 \leq k \leq p(m)$.

2. For $1 \leq k \leq p(m)$, a period-level start-time variable $\mathtt{S}(m_k)$ with

$$\delta(\mathtt{S}(m_k)) = \{(i, j+k-1) : (i, j) \in \delta(\mathtt{S}(m))\}$$

is introduced. Let X denote the set of variables obtained in this way.

3. To achieve that the meetings of the decomposition are scheduled in succession and without any gaps, the constraint $\mathtt{S}(m_{k+1}) = \mathtt{S}(m_k) + 1$ is imposed for $1 \leq k < p(m)$.

4. To relate m to the meetings of its decomposition, the constraint $\mathtt{S}(m_1) = \mathtt{S}(m)$ is imposed.

Using this infrastructure, double-bookings are avoided by means of

$$\mathtt{alldiff}(X).$$

The `alldiff` constraint applies only if the assignment of consumers to resources is fixed. Hence it cannot be used to avoid the double-booking of rooms unless rooms are allocated prior to time-slot assignment. To allow for a problem solver that alternates between time-slot assignment and room allocation, an approach has been implemented that is based on the disjoint placement of rectangles in the plane spanned by the time grid $G(P)$ and by the room set $R(P)$. Each meeting $m \in M(P)$ is assigned a rectangle of width $p(m)$ and height 1 the origin of which is not fixed. Clearly, a disjoint placement of the rectangles corresponds to a timetable where no room is double-booked. To enforce a disjoint placement, a single `disjoint` constraint is used:

$$\mathtt{disjoint}(\{(\mathtt{S}(m), \mathtt{R}(m), p(m), 1) : m \in M(P)\})$$

Restrictions on room availability are modeled by means of fixed rectangles.

4.2.6 Bounds on Daily Work Load

To enforce bounds on daily work load, global cardinality constraints are used. The procedure is demonstrated on the example of a teacher t with bounds l and u on her daily work load. Suppose that $0 \leq l \leq u \leq h(P)$, let $I = \bigcup_{m \in M(t)} \delta(\bar{\mathtt{S}}(m))$, and let $n = |I|$.

1. For $1 \leq i \leq n$, a filling task $f_{t,i}$ with duration u is introduced.

2. For $1 \leq i \leq n$, a day-level start-time variable $\bar{\mathtt{S}}(f_{t,i})$ with domain $I \cup \{-i\}$ is introduced.

3. For each meeting $m \in M(t) \cup \{f_{t,i} : 1 \leq i \leq n\}$, the following steps are taken:

   (a) m is decomposed into $p(m)$ meetings $m_1, \ldots, m_{p(m)}$ with $p(m_k) = 1$ for $1 \leq k \leq p(m)$.

(b) For $1 \leq k \leq p(m)$, a day-level start-time variable $\bar{\mathtt{S}}(m_k)$ with $\delta(\bar{\mathtt{S}}(m_k)) = \delta(\bar{\mathtt{S}}(m))$ is introduced. Let X denote the set of variables obtained in this way.

(c) To achieve that the meetings of the decomposition are scheduled for the same day, the constraint $\bar{\mathtt{S}}(m_k) = \bar{\mathtt{S}}(m)$ is imposed for $1 \leq k \leq p(m)$.

4. For $i \in I$, a frequency variable $\mathtt{F}_{t,i}$ with domain $[l,u]$ is introduced.

   Let $F_1 = \{(i, \mathtt{F}_{t,i}) : i \in I\}$.

5. For $1 \leq i \leq n$, a frequency variable $\mathtt{F}_{t,-i}$ with domain $[0,u]$ is introduced.

   Let $F_2 = \{(-i, \mathtt{F}_{t,-i}) : 1 \leq i \leq n\}$.

With this infrastructure, the bounds on daily work load are enforced by means of

$$\mathtt{gcc}(X, F_1 \cup F_2).$$

The filling tasks are required to implement the lower bound on daily work load (c.f. Section 2.1). Suppose $l > 0$ and consider some day $i \in I$. With the filling task $f_{t,i}$, there are two alternatives for the timetable of t: Either the day i is free or at least l time slots are occupied by meetings of t. The former situation occurs if the filling task is scheduled for the day i. Then at the same time, the lower bound is satisfied and no other meeting of t can be scheduled for the day i because the duration of the filling task equals the upper bound. The latter situation occurs if the filling task is scheduled for a negative 'day'.

4.2.7 Bounds on the Number of Working Days

To enforce bounds on the number of working days, global cardinality constraints are used. The procedure is demonstrated on the example of a teacher t with bounds l and u on the number of her working days. Let $I = \bigcup_{m \in M(t)} \delta(\bar{\mathtt{S}}(m))$, suppose that $0 \leq l \leq u \leq |I|$, and let $n = |I| - l$.

1. For $1 \leq i \leq n$, a filling task $f_{t,i}$ with duration $h(P)$ is introduced.

2. For $1 \leq i \leq n$, a day-level start-time variable $\bar{\mathtt{S}}(f_{t,i})$ with

$$\delta(\bar{\mathtt{S}}(f_{t,i})) = \begin{cases} I \cup \{-i\}, & \text{if } 1 \leq i \leq u - l, \\ I & \text{otherwise} \end{cases}$$

   is introduced.

3. The meetings and the filling tasks are decomposed and day-level start variables are introduced as described in the previous section. Let X denote the set of variables obtained in this way.
4. For $i \in I$, a frequency variable $\mathtt{F}_{t,i}$ with domain $[1, h(P)]$ is introduced.

   Let $F_1 = \{(i, \mathtt{F}_{t,i}) : i \in I\}$.
5. For $1 \leq i \leq u - l$, a frequency variable $\mathtt{F}_{t,-i}$ with domain $[0, h(P)]$ is introduced.

   Let $F_2 = \{(-i, \mathtt{F}_{t,-i}) : 1 \leq i \leq u - l\}$.

With this infrastructure, the bounds on the number of working days are enforced by means of

$$\mathtt{gcc}(X, F_1 \cup F_2).$$

The filling tasks with domain I are required to implement the upper bound u. As there are $|I| - u$ filling tasks of this kind, there will be at least $|I| - u$ free days and thus at most u working days. The lower bound l is enforced by requiring that at least one time slot is occupied each day. Yet this amounts to a lower bound on daily work load that rules out solutions with more than $|I| - u$ free days. To allow for such solutions, $u - l$ filling tasks are introduced each of which can either be scheduled for some day in I or be put aside. The former decision leads to an additional free day while the latter decision entails an additional working day.

If there are both bounds on daily work load and on the number of working days, all the bounds can be enforced with a single global cardinality constraint by combining the approaches presented in this and the previous section.

4.3 Inference Processes

This section proposes two reductions for TPP inference in school timetabling, demonstrates their correctness and convergence, and gives extensive examples.

To keep the presentation simple, a TPP is specified by a set of job sets. This differs from the representation used in Chapter 3 where a TPP was specified by a set of task sets. Yet, as each job is a set of meetings and each meeting constitutes a task, the translation is straightforward. Note that couplings (a special kind of TPP with only one job per track, c.f. 2.1) form an exception; a coupling is specified by means of a job set.

We will make heavy use of the notation that has been introduced in Section 4.2. In addition, we will use the following concepts and shortcuts related to students and their programs:

Definition 4.3. Let P be a school-timetabling problem.

- If $j \in J(P)$, then
  - $S(j)$ denotes the students of j and
  - $p(j)$ denotes $\sum_{m \in M(j)} p(m)$ (the so-called ***processing time*** of j).
- If $s \in S(P)$, then
  - $J(s)$ denotes the jobs s participates in (the so-called ***program*** of s) and
  - $F(s)$ denotes $G(P) \setminus D(s)$ (the so-called ***time frame*** of s).

  $F(s)$ is called ***tight*** iff $|F(s)| = \sum_{j \in J(s)} p(j)$.
- Students $s_0, s_1 \in S(P)$ are called ***equivalent*** iff $J(s_0) = J(s_1)$.
- $\mathcal{H}(P)$ denotes the equivalence classes of $S(P)$ wrt. the student-equivalence relation (the so-called ***homogeneous groups*** of P).
- If $H \in \mathcal{H}(P)$, then
  - $J(H)$ is the subset of $J(P)$ with $J(H) = J(s)$ for all students $s \in H$ (the so-called ***program*** of H) and
  - $F(H)$ denotes $\bigcap_{s \in H} F(s)$ (the so-called ***time frame*** of H).

  $F(H)$ is called ***tight*** iff $|F(H)| = \sum_{j \in J(H)} p(j)$.

So all students of a homogeneous group have the same program and if two students have the same program, they belong to the same homogeneous group. Hence the program of any homogeneous group is well-defined.

Next, as TPPs are specified in terms of jobs instead of tasks, we need to adapt the definitions of value covers and value supplies:

Definition 4.4. Let P be a school-timetabling problem and let δ denote the domain function of the constraint model $\mu_0(P)$ (c.f. Section 4.2).

- If $j \in J(P)$, then
  - $\mathrm{vc}(j, \delta)$ denotes $\mathrm{vc}(\{(\mathtt{S}(m), p(m)) : m \in M(j)\}, \delta)$ and
  - $\mathrm{vs}(j, \delta)$ denotes $\mathrm{vs}(\{(\mathtt{S}(m), p(m)) : m \in M(j)\}, \delta)$.

  ($S(m)$ is the period-level start-time variable for the meeting m as introduced by the model generator μ_0.)

- If $T \subseteq J(P)$ (i.e. if T is a track), then
  - $\mathrm{vc}(T,\delta)$ denotes $\bigcup_{j\in T} \mathrm{vc}(j,\delta)$ and
  - $\mathrm{vs}(T,\delta)$ denotes $\bigcup_{j\in T} \mathrm{vs}(j,\delta)$.
- If $\mathcal{T}$ is a set of subsets of $J(P)$ (i.e. if $\mathcal{T}$ is a track set), then
  - $\mathrm{vc}(\mathcal{T},\delta)$ denotes $\bigcup_{T\in\mathcal{T}} \mathrm{vc}(T,\delta)$ and
  - $\mathrm{vs}(\mathcal{T},\delta)$ denotes $\bigcap_{T\in\mathcal{T}} \mathrm{vs}(T,\delta)$.

The reader is encouraged to verify that all properties of value covers and supplies readily transfer from task to job level.

To simplify the translation of TPPs to `tpp` constraints, we introduce the following shortcuts:

Definition 4.5. Let P be a school-timetabling problem.

- If $\mathcal{T}$ is a set of subsets of $J(P)$ (i.e. if $\mathcal{T}$ is a track set), then $\|\mathcal{T}\|$ denotes the constraint

$$\mathtt{tpp}(\{\{(\mathtt{S}(m), p(m)) : j \in T \wedge m \in M(j)\} : T \in \mathcal{T}\}).$$

- If $J \subseteq J(P)$, then $\mathcal{J}(J)$ denotes $\{\{j\} : j \in J\}$.

Finally, $\mathcal{P}(X)$ denotes the power set of X as usual.

4.3.1 Reductions

We start by defining the reduction system $(A(P), \rightarrow_{A,P})$ for any school-timetabling problem P. It serves to infer couplings from the students' individual programs and time frames. $A(P)$ comprises all sets of couplings that are possible on the basis of $J(P)$. An application of $\rightarrow_{A,P}$ transforms a set of couplings into one of smaller cardinality by merging couplings. The merging operation takes a pair of couplings J_0 and J_1 and returns the coupling $J_0 \cup J_1$. Such a merging is admissible only if J_0 and J_1 are not disjoint or if there are jobs $j_0 \in J_0$ and $j_1 \in J_1$ with students $s_0 \in S(j_0)$ and $s_1 \in S(j_1)$ the programs of whom do not differ except for j_0 and j_1 and who have identical and tight time frames.

Definition 4.6. Let $(A(P), \rightarrow_{A,P})$ be a reduction system with

$$A(P) = \mathcal{P}(\mathcal{P}(J(P)))$$

and $R_0 \rightarrow_{A,P} R_1$ iff $J_0, J_1 \in R_0$ exist s.t.

- $J_0 \neq J_1$;
- either $J_0 \cap J_1 \neq \emptyset$ or $j_0 \in J_0$, $j_1 \in J_1$, $s_0 \in S(j_0)$, and $s_1 \in S(j_1)$ exist with
  - $J(s_0) \setminus \{j_0\} = J(s_1) \setminus \{j_1\}$,
  - $F(s_0) = F(s_1)$, and
  - both $F(s_0)$ and $F(s_1)$ are tight; and
- $R_1 = R_0 \setminus \{J_0, J_1\} \cup \{J_0 \cup J_1\}$.

$\mathcal{P}(J(P))$ is the set of couplings that are possible on the basis of $J(P)$. The requirement $J_0 \neq J_1$ is necessary to obtain a terminating reduction.

Next we define the reduction system $(B(P), \rightarrow_{B,P})$ that serves to infer TPPs from the couplings of P. $B(P)$ comprises all sets of TPPs that are possible on the basis of $J(P)$ and that satisfy the following requirement: Every track's jobs belong to a single homogeneous group. (Hence the meetings of a track must not overlap.) An application of $\rightarrow_{B,P}$ transforms a set of TPPs by reducing some TPP. The reduction operation takes a TPP $\mathcal{T}_0$ and a job set J and returns the TPP $\mathcal{T}_1 = \{T \setminus J : T \in \mathcal{T}_0\}$. Such a reduction is admissible only if the jobs in J are coupled according to $C(P)$ and if J intersects with each track of $\mathcal{T}_0$.

Definition 4.7. Let $(B(P), \rightarrow_{B,P})$ be a reduction system with

$$B(P) = \mathcal{P}\left\{\mathcal{T} \in \mathcal{P}(\mathcal{P}(J(P))) : \forall T \in \mathcal{T}.\, \exists H \in \mathcal{H}(P).\, T \subseteq J(H)\right\}$$

and $R_0 \rightarrow_{B,P} R_1$ iff $J \in C(P)$ and $\mathcal{T}_0 \in R_0$ exist s.t.

- $\forall T \in \mathcal{T}_0.\, J \cap T \neq \emptyset$ and
- $R_1 = R_0 \setminus \{\mathcal{T}_0\} \cup \{\mathcal{T}_1\}$ with $\mathcal{T}_1 = \{T \setminus J : T \in \mathcal{T}_0\}$.

$\mathcal{P}(J(P))$ is the set of tracks that are possible on the basis of $J(P)$ and $\mathcal{P}(\mathcal{P}(J(P)))$ is the set of TPPs that are possible on the basis of these tracks.

4.3.2 Examples

We give four examples to demonstrate the inference capabilities of the reductions we have introduced. As a side effect, it becomes clear how study options complicate timetabling by entailing the frequent need for parallel education.

This section's examples take up the examples of Section 2.2.3 where a school-timetabling problem has been presented in part. Suppose the entire problem is known and let P denote this problem.

	Subject														
Class	REC	REP	Eth	G	E	F	L	M	**B**	H	Geo	**A**	**Mu**	**PEM**	**PEF**
7a-EL														d_2^2	g_1^2
7b-EL	a_3^2	b_1^2												e_1^2	h_2^2
7c-EL		b	c_1^2											d	g
7d-EF	a													f_1^2	i_1^2
7e-EF	a	b	c											e	h

Table 4.1:
Four couplings inferred from the modules specified in Table 2.2
(cf. Example 4.1).

Example 4.1. Table 4.1 shows four couplings that have been inferred on the basis of Table 2.2 which specifies the modules of a seventh grade. The couplings are set off from each other by means of gray backgrounds of varying intensity (including white).

To explain how these couplings have been inferred by means of $\rightarrow_{A,P}$, we need to give some facts that are not apparent from Table 2.2. Let a_1, a_2, a_3, b_1, c_1, d_1, d_2, e_1, f_1, g_1, h_1, h_2, and i_1 denote the jobs due to the modules a through i.

1. All students of the seventh grade have the same time frame and this time frame is tight for each of them.

2. For $1 \leq i \leq 3$, students $s_0 \in S(a_i)$ and $s_1 \in S(b_1)$ with $J(s_0) \setminus \{a_i\} = J(s_1) \setminus \{b_1\}$ exist.

3. Students $s_0 \in S(a_3)$ and $s_1 \in S(c_1)$ with $J(s_0) \setminus \{a_3\} = J(s_1) \setminus \{c_1\}$ exist.

4. For $i \in \{1,2\}$, students $s_0 \in S(d_i)$ and $s_1 \in S(g_1)$ with $J(s_0) \setminus \{d_i\} = J(s_1) \setminus \{g_1\}$ exist.

5. For $i \in \{1,2\}$, students $s_0 \in S(h_i)$ and $s_1 \in S(e_1)$ with $J(s_0) \setminus \{h_i\} = J(s_1) \setminus \{e_1\}$ exist.

6. Students $s_0 \in S(f_1)$ and $s_1 \in S(i_1)$ with $J(s_0) \setminus \{f_1\} = J(s_1) \setminus \{i_1\}$ exist.

Moreover, as Table 2.2 shows, the job sets $J_0 = \{a_1, a_2, a_3, b_1, c_1\}$ (religious education) and $J_1 = \{d_1, d_2, e_1, f_1, g_1, h_1, h_2, i_1\}$ (physical education) are not related in the following sense: For $j_0 \in J_0$ and $j_1 \in J_1$, students $s_0 \in S(j_0)$ and $s_1 \in S(j_1)$ with $J(s_0) \setminus \{j_0\} = J(s_1) \setminus \{j_1\}$ do not exist. In consequence, $\rightarrow_{A,P}$ will never infer a coupling that contains jobs from both religious and physical education. This allows to split the inference process and we obtain two computations:

1. We consider J_0 and start out from $\mathcal{J}(J_0)$:

$$\begin{array}{rcll} \mathcal{J}(J_0) & = & \{\{a_1\},\{a_2\},\{a_3\},\{b_1\},\{c_1\}\} & \\ & \to_{A,P} & \{\{a_1,b_1\},\{a_2\},\{a_3\},\{c_1\}\} & \text{by fact (2)} \\ & \to_{A,P} & \{\{a_1,a_2,b_1\},\{a_3\},\{c_1\}\} & \text{by fact (2)} \\ & \to_{A,P} & \{\{a_1,a_2,a_3,b_1\},\{c_1\}\} & \text{by fact (2)} \\ & \to_{A,P} & \{\{a_1,a_2,a_3,b_1,c_1\}\} & \text{by fact (3)} \end{array}$$

2. We consider J_1 and start out from $\mathcal{J}(J_1)$:

$$\begin{array}{rcll} \mathcal{J}(J_1) & = & \{\{d_1\},\{d_2\},\{e_1\},\{f_1\},\{g_1\},\{h_1\},\{h_2\},\{i_1\}\} & \\ & \to_{A,P} & \{\{d_1,g_1\},\{d_2\},\{e_1\},\{f_1\},\{h_1\},\{h_2\},\{i_1\}\} & \text{by fact (4)} \\ & \to_{A,P} & \{\{d_1,d_2,g_1\},\{e_1\},\{f_1\},\{h_1\},\{h_2\},\{i_1\}\} & \text{by fact (4)} \\ & \to_{A,P} & \{\{d_1,d_2,g_1\},\{e_1,h_1\},\{f_1\},\{h_2\},\{i_1\}\} & \text{by fact (5)} \\ & \to_{A,P} & \{\{d_1,d_2,g_1\},\{e_1,h_1,h_2\},\{f_1\},\{i_1\}\} & \text{by fact (5)} \\ & \to_{A,P} & \{\{d_1,d_2,g_1\},\{e_1,h_1,h_2\},\{f_1,i_1\}\} & \text{by fact (6)} \end{array}$$

The following table surveys the resource requirements of the couplings that we have obtained:

Coupling	Classes	Teachers	Rooms	Total
$\{a_1,a_2,a_3,b_1,c_1\}$	4	5	5	14
$\{d_1,d_2,g_1\}$	2	3	3	8
$\{e_1,h_1,h_2\}$	2	3	3	8
$\{f_1,i_1\}$	1	2	2	5

For example, we see that to schedule one meeting of the first coupling, we have to make sure that four given classes, five given teachers, and five suitable rooms are available at the same time.

Example 4.2. Table 4.2 shows seven couplings that have been inferred on the basis of Table 2.3 which specifies the modules of a tenth grade. To make sure that the backgrounds can be distinguished from each other by the human eye, only four intensity levels (including white) have been employed. However, since there are seven couplings, three levels had to be used twice. We rule out any ambiguities by annotating that the couplings in the left part of Table 4.2 are unrelated to those in the right part.

Again, if μ is a module with n groups, then let $(\mu_i)_{1\leq i\leq n}$ denote the jobs due to μ. The couplings have been inferred from the following facts which are not apparent from Table 2.3:

Class	Subject														
	RE	G	E	F	L	M	**Ph**	**C**	**B**	(H & Soc)	EL	**A**	**Mu**	**PE**	**HE**
10a/ML-ELF	a_4^2				j_1^3	k_2^3	m_2^2					z_2^1	α_2^1	β_2^2	
10a/SOC-EL	a				j	k	m					z	α	β	
10a/SOC-EF	a			i_1^3		k	m					z	α	β	
10b/SOC-EF	a					k	m					z	α	γ_2^2	
10b/NAT-EF	a											z	α	γ	
10c/NAT-EF	a			i								z	α	β	
10c/NAT-EL	a				j							z	α	β	

Table 4.2:
Seven couplings inferred from the modules specified in Table 2.3 (cf. Example 4.2).

1. All students of the tenth grade have the same time frame and this time frame is tight for each of them.

2. For $1 \leq i \leq 3$, students $s_0 \in S(a_i)$ and $s_1 \in S(a_4)$ with $J(s_0) \setminus \{a_i\} = J(s_1) \setminus \{a_4\}$ exist.

3. Students $s_0 \in S(i_1)$ and $s_1 \in S(j_1)$ with $J(s_0) \setminus \{i_1\} = J(s_1) \setminus \{j_1\}$ exist.

4. For each module $\mu \in \{k, m, z, \beta, \gamma\}$ and its jobs μ_1 and μ_2, students $s_0 \in S(\mu_1)$ and $s_1 \in S(\mu_2)$ with $J(s_0) \setminus \{\mu_1\} = J(s_1) \setminus \{\mu_2\}$ exist.

5. For $i \in \{1, 2\}$, students $s_0 \in S(z_i)$ and $s_1 \in S(\alpha_i)$ with $J(s_0) \setminus \{z_i\} = J(s_1) \setminus \{\alpha_i\}$ exist.

Like in the previous example, the inference process may be split because $\rightarrow_{A,P}$ will never infer a coupling that contains, for example, jobs of religious and foreign-language education. To mention one of the seven computations that result from splitting, $\{\{a_1, \ldots, a_4\}\}$ is obtained by computing a normal form of $\{\{a_1\}, \ldots, \{a_4\}\}$ wrt. $\rightarrow_{A,P}$ by employing the first fact three times.

As in the previous example, the couplings' resource requirements are considerable:

Coupling	Classes	Teachers	Rooms	Total
$\{a_1, a_2, a_3, a_4\}$	3	4	4	11
$\{i_1, j_1\}$	2	2	2	6
$\{k_1, k_2\}$	2	2	2	6
$\{m_1, m_2\}$	2	2	2	6
$\{z_1, z_2, \alpha_1, \alpha_2\}$	3	4	4	11
$\{\beta_1, \beta_2\}$	2	2	2	6
$\{\gamma_1, \gamma_2\}$	1	2	2	5

	Subject															
Class	REC	REP	Eth	G	E	F	L	M	**B**	H	Geo	**A**	**Mu**	**PEM**	**PEF**	#[a]
7a-EL																
7b-EL				4	4		5	4	2	2	1	2	2			26
7c-EL																
7d-EF																
7e-EF				4	4	5		4	2	2	1	2	2			26
7a-EL	2			4	4		5	4	2	2	1	2	2			28
7b-EL		b_1^2														
7c-EL		b	c_1^2	4	4		5	4	2	2	1	2	2			28
7d-EF																
7e-EF		b	c													

Table 4.3:
Two TPPs inferred from the couplings depicted in Table 4.1 (cf. Example 4.3). The upper TPP stems from the classes 7b-EL and 7e-EF. The lower TPP stems from the classes 7a-EL and 7c-EL.

[a] For each track, this column gives the number of teaching periods per week.

Again, it becomes clear how study options may complicate timetabling in combination with tight time frames.

Example 4.3. Table 4.3 shows two TPPs that have been inferred from the couplings depicted in Table 4.1. The TPPs are set off from each other by means of gray backgrounds of varying intensity. The first TPP stems from the classes 7b-EL and 7e-EF. Since the students of theses classes are joined for religious and for physical education and their time frames are tight, the remaining subjects have to be scheduled for the remaining slots. This conclusion has been modeled by a TPP with two tracks: The upper track contains all lessons that involve 7b-EL but not 7e-EF while the lower track contains all lessons that involve 7e-EF but not 7b-EL. Each track contains all lessons its class is involved in except for the lessons both classes are involved in. Thus, in this case, religious and physical education are missing. The second TPP stems from the classes 7a-EL and 7c-EL. Since the students of theses classes are joined for physical education only, we obtain a TPP with two tracks where physical education is missing.

Example 4.4. Table 4.4 shows three TPPs that have been inferred from the couplings depicted in Table 4.2. The first TPP stems from the classes 10a/ML-ELF and 10a/SOC-EL. It essentially reflects the curricular differences of the study directions Modern Languages (ML) and Social Sciences (SOC). While ML students

Class	RE	G	E	F	L	M	**Ph**	**C**	**B**	(H & Soc)	EL	**A**	**Mu**	**PE**	**HE**	#[a]
10a/ML-ELF				5						t_1^3						8
10a/SOC-EL								o_1^2		u_1^4					δ_1^2	8
10a/SOC-EF								o		u					δ	
10b/SOC-EF								o		u					δ	
10b/NAT-EF										t						
10c/NAT-EF																
10c/NAT-EL																
10a/ML-ELF						k_2^3	m_2^2			t_1^3						
10a/SOC-EL						k	m	o_1^2		u_1^4					δ_1^2	
10a/SOC-EF						k	m	o		u					δ	
10b/SOC-EF						k	m	o		u					δ	13
10b/NAT-EF						l_1^4	n_1^3	p_1^3		t						13
10c/NAT-EF						l	n	p								
10c/NAT-EL						l	n	p								
10a/ML-ELF		b_1^3	e_1^3		j_1^3				q_1^2		w_1^1			β_2^2		
10a/SOC-EL		b	e		j				q		w			β		
10a/SOC-EF		b	e	i_1^3					q		w			β		14
10b/SOC-EF		c_1^3	f_1^3	h_1^3					r_1^2		x_1^1			γ_2^2		14
10b/NAT-EF		c	f	h					r		x			γ		
10c/NAT-EF				i										β		
10c/NAT-EL					j									β		

Table 4.4:
Three TPPs inferred from the couplings depicted in Table 4.2 (cf. Example 4.4). The first TPP stems from the the classes 10a/ML-ELF and 10a/SOC-EL. The second TPP stems from the classes 10b/SOC-EF and 10b/NAT-EF. The third TPP stems from the classes 10a/SOC-EF and 10b/SOC-EF.

[a]For each track, this column gives the number of teaching periods per week.

are taught a third foreign language and Social Studies, SOC students are taught Chemistry, Social Studies, and Home Economics. The second TPP stems from the classes 10b/SOC-EF and 10b/NAT-EF. As the previous TPP, it reflects the curricular differences of two study directions. Students in Natural Sciences (NAT) attend ten periods of Mathematics, Physics, and Chemistry a week. In the SOC direction, only seven periods a week are allocated for these subjects. In return, SOC students attend more periods in Social Studies than their NAT colleagues and receive education in Home Economics. The third TPP stems from the classes 10a/SOC-EF and 10b/SOC-EF. This TPP is basically due to students from the same study direction but from different classes being joined for education specific to the study direction.

4.3.3 Correctness

Consider some school-timetabling problem P and its constraint model $\mu_0(P)$. We will show that $\rightarrow_{A,P}$ is correct in the following sense: If $R_0 \rightarrow_{A,P} R_1$, then every solution to $\mu_0(P)$ that satisfies the couplings in R_0 also satisfies the couplings in R_1 and vice versa. To formalize this statement, we translate the couplings to `tpp` constraints (J becomes $\|\mathcal{J}(J)\|$) and employ the constraint-addition operator $\oplus$:

Proposition 4.1. *If* $R_0 \rightarrow_{A,P} R_1$, *then* $P_0 \equiv P_1$ *with* $P_i = \mu_0(P) \oplus \{\|\mathcal{J}(J)\| : J \in R_i\}$, $i \in \{0,1\}$.

Proof. Let $J_0, J_1 \in R_0$ s.t. the preconditions for a transition are satisfied and $R_1 = R_0 \setminus \{J_0, J_1\} \cup \{J_0 \cup J_1\}$. We have to show that $\mathrm{sol}(P_0) = \mathrm{sol}(P_1)$.

$\mathrm{sol}(P_1) \subseteq \mathrm{sol}(P_0)$: Let $\sigma \in \mathrm{sol}(P_1)$. σ satisfies $\|\mathcal{J}(J_0 \cup J_1)\|$ and thus it satisfies $\|\mathcal{J}(J_0)\|$ as well as $\|\mathcal{J}(J_1)\|$.

$\mathrm{sol}(P_0) \subseteq \mathrm{sol}(P_1)$ if $J_0 \cap J_1 = \emptyset$: Choose any $j_0 \in J_0$, $j_1 \in J_1$, $s_0 \in S(j_0)$, and $s_1 \in S(j_1)$ s.t. $J(s_0) \setminus \{j_0\} = J(s_1) \setminus \{j_1\}$, $F(s_0) = F(s_1)$, and both $F(s_0)$ and $F(s_1)$ are tight. Let $\sigma \in \mathrm{sol}(P_0)$, $T_0 = J(s_0)$, and $T_1 = J(s_1)$. Lemma 3.4, Lemma 3.1, and the definition of μ_0 yield

$$\mathrm{vc}(T_i, \sigma) = \mathrm{vs}(T_i, \sigma) \subseteq \mathrm{vs}(T_i, \delta_0) \subseteq F(s_i)$$

for $i \in \{0,1\}$. $\mathrm{vc}(T_i, \sigma) \subset F(s_i)$ can be ruled out because $F(s_i)$ is tight for $i \in \{0,1\}$ and thus $|F(s_i)|$ slots are required to schedule the lessons of s_i without double-booking s_i. With $F(s_0) = F(s_1)$, we obtain $\mathrm{vc}(T_0, \sigma) = \mathrm{vc}(T_1, \sigma)$, i.e. σ satisfies $\|\{T_0, T_1\}\|$. To finish the proof, we need to take into account that students must not be double-booked. This yields

$$\mathrm{vc}(j_i, \sigma) = \mathrm{vc}(T_i, \sigma) \setminus \mathrm{vc}(T_i \setminus \{j_i\}, \sigma)$$

for $i \in \{0,1\}$. Putting it all together, we obtain

$$\begin{aligned} \mathrm{vc}(j_0,\sigma) &= \mathrm{vc}(T_0,\sigma) \setminus \mathrm{vc}(T_0 \setminus \{j_0\},\sigma) \\ &= \mathrm{vc}(T_1,\sigma) \setminus \mathrm{vc}(T_1 \setminus \{j_1\},\sigma) = \mathrm{vc}(j_1,\sigma), \end{aligned}$$

i.e. j_0 and j_1 and thus, by Lemma 3.4, all jobs of J_0 and J_1 are processed in parallel under σ. In consequence, σ satisfies $\|\mathcal{J}(J_0 \cup J_1)\|$.

$\mathrm{sol}(P_0) \subseteq \mathrm{sol}(P_1)$ if $J_0 \cap J_1 \neq \emptyset$: Let $\sigma \in \mathrm{sol}(P_0)$ and $j \in J_0 \cap J_1$. σ satisfies $\|\mathcal{J}(J_0)\|$ and $\|\mathcal{J}(J_1)\|$ and thus, by Lemma 3.4, the jobs of J_0 and J_1 are processed in parallel with j under σ. In consequence, σ satisfies $\|\mathcal{J}(J_0 \cup J_1)\|$ as well. □

To formalize the correctness of $\to_{B,P}$, we follow the pattern used with $\to_{A,P}$:

Proposition 4.2. *If $R_0 \to_{B,P} R_1$, then $P_0 \equiv P_1$ with $P_i = \mu_0(P) \oplus \{\|\mathcal{T}\| : \mathcal{T} \in R_i\}$, $i \in \{0,1\}$.*

Proof. Let $J \in C(P)$ and $\mathcal{T}_0 \in R_0$ s.t. the preconditions for a transition are satisfied and $R_1 = R_0 \setminus \{\mathcal{T}_0\} \cup \{\mathcal{T}_1\}$ with $\mathcal{T}_1 = \{T \setminus J : T \in \mathcal{T}_0\}$. Let $T \in \mathcal{T}_0$ and $I = T \cap J$. We start by making some observations for $\sigma \in \mathrm{sol}(\mu_0(P))$:

- For all $j_0, j_1 \in T$, $\mathrm{vc}(j_0,\sigma) \cap \mathrm{vc}(j_1,\sigma) = \emptyset$ because a homogeneous group must not be double-booked.
- $\mathrm{vc}(T,\sigma) = \mathrm{vc}(T \setminus I,\sigma) \cup \mathrm{vc}(I,\sigma)$ because $I \subseteq T$.
- $\mathrm{vc}(T \setminus I,\sigma) \cap \mathrm{vc}(I,\sigma) = \emptyset$ because $I \subseteq T$ and, for all $j_0, j_1 \in T$, $\mathrm{vc}(j_0,\sigma) \cap \mathrm{vc}(j_1,\sigma) = \emptyset$.
- $\mathrm{vc}(T \setminus I,\sigma) = \mathrm{vc}(T,\sigma) \setminus \mathrm{vc}(I,\sigma)$ follows easily from the previous facts.
- $\mathrm{vc}(I,\sigma) = \mathrm{vc}(\mathcal{J}(J),\sigma)$: σ satisfies $\|\mathcal{J}(J)\|$ and hence, by Lemma 3.4, $\mathrm{vc}(\{j\},\sigma) = \mathrm{vc}(\mathcal{J}(J),\sigma)$ for all $j \in J$. The claim follows easily with $I \subseteq T$.

These relations readily specialize to any solution to P_i, $i \in \{0,1\}$, because $\mathrm{sol}(P_i) \subseteq \mathrm{sol}(\mu_0(P))$.

$\mathrm{sol}(P_0) \subseteq \mathrm{sol}(P_1)$: Let $\sigma \in \mathrm{sol}(P_0)$. σ satisfies $\|\mathcal{T}_0\|$ and hence, by Lemma 3.4, $\mathrm{vc}(T,\sigma) = \mathrm{vc}(\mathcal{T}_0,\sigma)$. We obtain

$$\begin{aligned} \mathrm{vc}(T \setminus J,\sigma) &= \mathrm{vc}(T \setminus I,\sigma) \\ &= \mathrm{vc}(T,\sigma) \setminus \mathrm{vc}(I,\sigma) \\ &= \mathrm{vc}(\mathcal{T}_0,\sigma) \setminus \mathrm{vc}(\mathcal{J}(J),\sigma), \end{aligned}$$

i.e. $\mathrm{vc}(T \setminus J,\sigma)$ does not depend on T.

$\mathrm{sol}(P_1) \subseteq \mathrm{sol}(P_0)$: Let $\sigma \in \mathrm{sol}(P_1)$. σ satisfies $\|\mathcal{T}_1\|$ and hence, by Lemma 3.4, $\mathrm{vc}(T \setminus J, \sigma) = \mathrm{vc}(\mathcal{T}_1, \sigma)$. We obtain

$$\begin{aligned}
\mathrm{vc}(T,\sigma) &= \mathrm{vc}(T \setminus I,\sigma) \cup \mathrm{vc}(I,\sigma) \\
&= \mathrm{vc}(T \setminus J,\sigma) \cup \mathrm{vc}(I,\sigma) \\
&= \mathrm{vc}(\mathcal{T}_1,\sigma) \cup \mathrm{vc}(\mathcal{J}(J),\sigma),
\end{aligned}$$

i.e. $\mathrm{vc}(T,\sigma)$ does not depend on T. □

4.3.4 Convergence

A reduction is called convergent if it is terminating and confluent. Termination guarantees that there are no infinite computations; confluence guarantees that every element of the underlying set has at most one normal form. With a convergent reduction system, whatever path of computation is actually taken for a given input (there may be many due to non-determinism), the result is uniquely determined (c.f. Section 1.5).

Let P be a school-timetabling problem. To demonstrate the convergence of $\rightarrow_{A,P}$, we take the route via Newman's Lemma which states that a terminating reduction is confluent if it is locally confluent, i.e. if every single-step fork is joinable. To ensure termination, we need to require that $J(P)$ is finite. This restriction is without any practical relevance.

Proposition 4.3. *If $J(P)$ is finite, then $\rightarrow_{A,P}$ is terminating.*

Proof. Remember that $\rightarrow_{A,P}$ is a reduction on the set $A(P) = \mathcal{P}(\mathcal{P}(J(P)))$. If $J(P)$ is finite, then $\mathcal{P}(J(P))$ is finite as well and hence every element of $A(P)$ is finite.

Let $R \in A(P)$ and consider some computation that starts out from R. Since R is a finite set of couplings and every transition of $\rightarrow_{A,P}$ reduces the number of couplings by one, there are at most $|R| - 1$ transitions. □

Proposition 4.4. *$\rightarrow_{A,P}$ is locally confluent.*

Proof. There is only one kind of single-step fork that requires a close look. Let $R_1 \leftarrow_{A,P} R_0 \rightarrow_{A,P} R_2$ s.t.

- $J_0, J_1, J_2 \in R_0$ and J_0, J_1, and J_2 are pairwise different;
- $R_1 = R_0 \setminus \{J_0, J_1\} \cup \{J_0 \cup J_1\}$; and
- $R_2 = R_0 \setminus \{J_0, J_2\} \cup \{J_0 \cup J_2\}$.

We will show that $R_1 \rightarrow^*_{A,P} R_3 \leftarrow^*_{A,P} R_2$ with $R_3 = R_0 \setminus \{J_0, J_1, J_2\} \cup \{J_0 \cup J_1 \cup J_2\}$.

To show that $R_1 \rightarrow^*_{A,P} R_3$, we consider $J_0 \cup J_1, J_2 \in R_1$ and distinguish two cases: If $J_2 = J_0 \cup J_1$, then

$$\begin{aligned} R1 &= R_0 \setminus \{J_0, J_1\} \cup \{J_0 \cup J_1\} \\ &= R_0 \setminus \{J_0, J_1, J_0 \cup J_1\} \cup \{J_0 \cup J_1\} \\ &= R_0 \setminus \{J_0, J_1, J_2\} \cup \{J_0 \cup J_1 \cup J_2\} = R_3. \end{aligned}$$

Otherwise, we have $R_1 \rightarrow_{A,P} R_3$:

- From $R_0 \rightarrow_{A,P} R_2$, we conclude that either $J_0 \cup J_1$ and J_2 are not disjoint or that $j_0 \in J_0 \cup J_1$, $j_2 \in J_2$, $s_0 \in S(j_0)$, and $s_2 \in S(j_2)$ exist s.t. $J(s_0) \setminus \{j_0\} = J(s_2) \setminus \{j_2\}$, $F(s_0) = F(s_2)$, and both $F(s_0)$ and $F(s_2)$ are tight.

- Obviously, $R_1 \setminus \{J_0 \cup J_1, J_2\} \cup \{J_0 \cup J_1 \cup J_2\} = R_3$.

$R_2 \rightarrow^*_{A,P} R_3$ can be established in a similar way. □

Corollary 4.1. *If $J(P)$ is finite, then $\rightarrow_{A,P}$ is convergent.*

The following statement on normal forms follows easily with Lemma 1.1:

Corollary 4.2. *Let $R \in A(P)$. If $J(P)$ is finite, then R has a uniquely determined normal form wrt. $\rightarrow_{A,P}$.*

To demonstrate the convergence of $\rightarrow_{B,P}$, we proceed as before. Again, we need to require that $J(P)$ is finite.

Proposition 4.5. *If $J(P)$ is finite, then $\rightarrow_{B,P}$ is terminating.*

Proof. Remember that $\rightarrow_{B,P}$ is a reduction on the set $B(P) \subseteq \mathcal{P}(\mathcal{P}(\mathcal{P}(J(P))))$. If $J(P)$ is finite, then $\mathcal{P}(\mathcal{P}(J(P)))$ is finite as well and hence every element of $B(P)$ is finite. $C(P)$ is finite because $C(P) \in A(P)$ and every element of $A(P)$ is finite as we know from the proof to the claim that $\rightarrow_{A,P}$ is terminating.

Let $R \in B(P)$. To reduce a TPP $\mathcal{T}_0 \in R$, we need a coupling $J \in C(P)$ which intersects with each track $T \in \mathcal{T}_0$. Given such a coupling, $\mathcal{T}_0$ is replaced with the TPP $\mathcal{T}_1 = \{T \setminus J : T \in \mathcal{T}_0\}$. It is obvious that $\mathcal{T}_1$ cannot be reduced further on the basis of J. Hence any computation that starts out from $\{\mathcal{T}_0\}$ performs at most $|C(P)|$ transitions. It follows that any computation that starts out from R performs at most $|R|\,|C(P)|$ transitions. □

For our proof of local confluence, we require that the couplings of P are disjoint. This restriction is without any practical relevance because, if $C(P)$ does not have this property, $\rightarrow_{A,P}$ can be used to establish it (cf. Proposition 4.7).

Proposition 4.6. *If the elements of $C(P)$ are disjoint, then $\rightarrow_{B,P}$ is locally confluent.*

Proof. There is only one kind of single-step fork that requires a close look. Let $R_1 \leftarrow_{B,P} R_0 \rightarrow_{B,P} R_2$ s.t.

- $J_1, J_2 \in C(P)$ with $J_1 \neq J_2$,
- $\mathcal{T}_0 \in R_0$,
- $R_1 = R_0 \setminus \{\mathcal{T}_0\} \cup \{\mathcal{T}_1\}$ with $\mathcal{T}_1 = \{T \setminus J_1 : T \in \mathcal{T}_0\}$, and
- $R_2 = R_0 \setminus \{\mathcal{T}_0\} \cup \{\mathcal{T}_2\}$ with $\mathcal{T}_2 = \{T \setminus J_2 : T \in \mathcal{T}_0\}$.

We will show that $R_1 \rightarrow^*_{B,P} R_3 \leftarrow^*_{B,P} R_2$ with $R_3 = R_0 \setminus \{\mathcal{T}_0\} \cup \{\mathcal{T}_3\}$ and $\mathcal{T}_3 = \{(T \setminus J_1) \setminus J_2 : T \in \mathcal{T}_0\}$.

$R_1 \rightarrow_{B,P} R_3$: We use J_2 to reduce $\mathcal{T}_1 \in R_1$. This is possible because J_1 and J_2 are disjoint and thus $\forall T \in \mathcal{T}_0.\ \exists j \in J_2.\ j \in T \setminus J_1$. It is straightforward to verify that $R_3 = R_1 \setminus \{\mathcal{T}_1\} \cup \{\mathcal{T}_3\}$.

$R_2 \rightarrow_{B,P} R_3$: We use J_1 to reduce $\mathcal{T}_2 \in R_2$. This is possible because J_1 and J_2 are disjoint and thus $\forall T \in \mathcal{T}_0.\ \exists j \in J_1.\ j \in T \setminus J_2$. We obtain the state $R_2 \setminus \{\mathcal{T}_2\} \cup \{\{(T \setminus J_2) \setminus J_1 : T \in \mathcal{T}_0\}\}$ which equals R_3. □

Corollary 4.3. *If $J(P)$ is finite and the elements of $C(P)$ are disjoint, then $\rightarrow_{B,P}$ is convergent.*

The following statement on normal forms follows easily with Lemma 1.1:

Corollary 4.4. *Let $R \in B(P)$. If $J(P)$ is finite and the elements of $C(P)$ are disjoint, then R has a uniquely determined normal form wrt. $\rightarrow_{B,P}$.*

Note that $\rightarrow_{B,P}$ may derive couplings represented as sets of singleton job sets. These couplings are not fed back though they might trigger further applications of $\rightarrow_{A,P}$ and $\rightarrow_{B,P}$.

4.4 Conservative Model Extensions

This section defines the model generators μ_A, μ_B, and μ_{AB}. Given a school-timetabling problem P, the generators extend the constraint model $\mu_0(P)$ with `tpp` constraints derived by means of $\rightarrow_{A,P}$ and $\rightarrow_{B,P}$. We study the question whether the generators are correct and investigate the redundancy status of the constraints they add.

4.4.1 Model Generators

When applied to a school-timetabling problem P, μ_A computes a set of couplings, transforms the couplings into `tpp` constraints, and adds the `tpp` constraints to $\mu_0(P)$:

Definition 4.8. Let

$$\mu_A(P) = \mu_0(Q)$$

with $Q = P$ except for

$$C(Q) = I_A(P)\downarrow_{A,P}$$

where

$$I_A(P) = C(P) \cup \{\{j\} : j \in J(P)\}.$$

The couplings are computed by means of $\rightarrow_{A,P}$ starting out from the initial state $I_A(P)$ that contains the couplings of P and a singleton coupling for each job of P. To make sure that the computation converges (terminates in the uniquely determined state $I_A(P)\downarrow_{A,P}$), we require that $J(P)$ is finite. The resulting set of couplings has an interesting property:

Proposition 4.7. *The elements of $I_A(P)\downarrow_{A,P}$ are pairwise disjoint.*

Proof. Let $J_0, J_1 \in I_A(P)\downarrow_{A,P}$ with $J_0 \neq J_1$ and suppose J_0 and J_1 are not disjoint. Then $\rightarrow_{A,P}$ applies to $I_A(P)\downarrow_{A,P}$. This contradicts the premise that $I_A(P)\downarrow_{A,P}$ is a normal form. $\square$

In structure, μ_B is very similar to μ_A and, like μ_A, μ_B generates and processes TPPs. However, the outcome of the inference process employed by μ_B is not limited to couplings.

Definition 4.9. Let Let

$$\mu_B(P) = \mu_0(P) \oplus \{\|\mathcal{T}\| : \mathcal{T} \in I_B(P)\downarrow_{B,P}\}$$

with

$$I_B(P) = \big\{\{J(H) : H \in \mathcal{H}\} : \mathcal{H} \subseteq \mathcal{H}(P) \wedge f(\mathcal{H})\big\}$$

where a set of homogeneous groups $\mathcal{H}$ passes the filter f iff $F(H)$ is tight for all $H \in \mathcal{H}$ and $F(H_0) = F(H_1)$ for all $H_0, H_1 \in \mathcal{H}$.

The TPPs are computed by means of $\rightarrow_{B,P}$ starting out from the initial state $I_B(P)$ that contains a TPP for every subset of homogeneous groups with identical and tight time frames. To make sure that the computation converges (terminates in the uniquely determined state $I_B(P)\downarrow_{B,P}$), we require that $J(P)$ is finite and that the couplings in $C(P)$ are disjoint. (Remember that the definition of $\rightarrow_{B,P}$ refers to $C(P)$.)

μ_{AB} connects the inference processes of μ_A and μ_B in series. The couplings obtained in the manner of μ_A are used to drive the inference process of μ_B:

Definition 4.10. Let
$$\mu_{AB}(P) = \mu_B(Q)$$
with $Q = P$ except for
$$C(Q) = I_A(P)\downarrow_{A,P}.$$

The link is established by means of the intermediate problem Q which is obtained from the input P by replacing its couplings with $I_A(P)\downarrow_{A,P}$. $\mu_{AB}(P)$ is well-defined if $C(P)$ is finite: Then $C(Q)$ is finite, its elements are pairwise disjoint by Proposition 4.7, and thus $\mu_B(Q)$ is well-defined.

4.4.2 Correctness

This section demonstrates that μ_A, μ_B, and μ_{AB} are correct in the sense that the constraints they add do not restrict the solution space wrt. μ_0, i.e.

$$\mu_A(P) \equiv \mu_B(P) \equiv \mu_{AB}(P) \equiv \mu_0(P)$$

for any school-timetabling problem P the model generators apply to.

For μ_A, we show that adding any coupling in $I_A(P)$ is safe wrt. correctness and, on the basis of this knowledge, we prove that replacing the couplings of P with $I_A(P)\downarrow_{A,P}$ is safe, too.

Lemma 4.1. *If $J \in I_A(P)$, then $\mu_0(P) \equiv \mu_0(P) \oplus \{\|\mathcal{J}(J)\|\}$.*

Proof. Let $J \in I_A(P)$, $c = \|\mathcal{J}(J)\|$, and $P_0 = \mu_0(P)$. To show that $\mathrm{sol}(P_0) \subseteq \mathrm{sol}(P_0 \oplus \{c\})$, let $\sigma \in \mathrm{sol}(P_0)$. J is singleton and thus c has only one track. As `tpp` constraints with singleton track sets are satisfied by any value assignment, σ satisfies c. It follows that $\sigma \in \mathrm{sol}(P_0 \oplus \{c\})$. □

Proposition 4.8. $\mu_A(P) \equiv \mu_0(P)$

Proof. By the definition of μ_A, we have to show that $\mu_0(P) \equiv \mu_0(Q)$ with $Q = P$ except for $C(Q) = I_A(P)\downarrow_{A,P}$.

$\mathrm{sol}(\mu_0(P)) \subseteq \mathrm{sol}(\mu_0(Q))$: Let $\sigma \in \mathrm{sol}(\mu_0(P))$. By Lemma 4.1, σ satisfies all $\|\mathcal{J}(J)\|$ with $J \in I_A(P)$. By Lemma 1.3 and by the correctness of $\rightarrow_{A,P}$, it follows that σ satisfies all $\|\mathcal{J}(J)\|$ with $J \in I_A(P)\downarrow_{A,P}$ as well. Hence $\sigma \in \mathrm{sol}(\mu_0(Q))$.

$\mathrm{sol}(\mu_0(Q)) \subseteq \mathrm{sol}(\mu_0(P))$: Let $\sigma \in \mathrm{sol}(\mu_0(P))$ and $J \in C(P)$. If $J \in C(Q)$, then σ satisfies $\|\mathcal{J}(J)\|$, of course. Otherwise, by the definition of $\rightarrow_{A,P}$, a $K \in C(Q)$ with $J \subseteq K$ exists. As σ satisfies $\|\mathcal{J}(K)\|$, it satisfies $\|\mathcal{J}(J)\|$ as well. □

To show the correctness of μ_B, we follow the pattern used with μ_A:

Lemma 4.2. *If $\mathcal{T} \in I_B(P)$, then $\mu_0(P) \equiv \mu_0(P) \oplus \{\|\mathcal{T}\|\}$.*

Proof. Let $\mathcal{H} \subseteq \mathcal{H}(P)$ s.t. $\mathcal{T} = \{J(H) : H \in \mathcal{H}\}$. Let $P_0 = \mu_0(P)$, let δ_0 denote the domain function of P_0, let $\sigma \in \mathrm{sol}(P_0)$, and let $T = J(H)$. Lemma 3.4, Lemma 3.1, and the definition of μ_0 yield

$$\mathrm{vc}(T,\sigma) = \mathrm{vs}(T,\sigma) \subseteq \mathrm{vs}(T,\delta_0) \subseteq F(H_i).$$

$\mathrm{vc}(T,\sigma) \subset F(H)$ can be ruled out because $F(H)$ is tight and thus $|F(H)|$ slots are required to schedule the lessons of H without double-booking H. With $F(H_0) = F(H_1)$ for all $H_0, H_1 \in \mathcal{H}$, we obtain $\mathrm{vc}(J(H_0),\sigma) = \mathrm{vc}(J(H_1),\sigma)$, i.e. σ satisfies $\|\mathcal{T}\|$. It follows that $\mathrm{sol}(P_0) \subseteq \mathrm{sol}(P_0 \oplus \{\|\mathcal{T}\|\})$. The other direction is obvious and hence $P_0 \equiv P_0 \oplus \{\|\mathcal{T}\|\}$. □

Proposition 4.9. $\mu_B(P) \equiv \mu_0(P)$

Proof. $\mathrm{sol}(\mu_B(P)) \subseteq \mathrm{sol}(\mu_0(P))$: By the definition of μ_B, every constraint of $\mu_0(P)$ occurs in $\mu_B(P)$ and thus every solution to $\mu_B(P)$ is a solution to $\mu_0(P)$.

$\mathrm{sol}(\mu_0(P)) \subseteq \mathrm{sol}(\mu_B(P))$: Let $\sigma \in \mathrm{sol}(\mu_0(P))$. By Lemma 4.2, σ satisfies all $\|\mathcal{T}\|$ with $\mathcal{T} \in I_B(P)$. By Lemma 1.3 and by the correctness of $\rightarrow_{B,P}$, it follows that σ satisfies all $\|\mathcal{T}\|$ with $\mathcal{T} \in I_B(P)\downarrow_{B,P}$ as well. Hence $\sigma \in \mathrm{sol}(\mu_B(P))$. □

With the results we have obtained up to now, the correctness of μ_{AB} follows easily:

Proposition 4.10. $\mu_{AB}(P) \equiv \mu_0(P)$

Proof. Let $Q = P$ except for $C(Q) = I_A(P)\downarrow_{A,P}$. By the definition of μ_{AB}, the correctness of μ_B, the definition of μ_A, and the correctness of μ_A, we obtain

$$\mu_{AB}(P) = \mu_B(Q) \equiv \mu_0(Q) = \mu_A(P) \equiv \mu_0(P).$$

□

4.4.3 Redundancy

A constraint is called redundant wrt. a given problem, if the problem implies the constraint (i.e. every solution to the problem satisfies the constraint) but does not state it explicitely (c.f. Definition 1.2).

This section shows that every constraint of $\mu_A(P)$, $\mu_B(P)$, and $\mu_{AB}(P)$ is redundant wrt. $\mu_0(P)$ if it does not occur in $\mu_0(P)$.

Proposition 4.11. *Let C_0, C_A, C_B, and C_{AB} denote the constraints of $\mu_0(P)$, $\mu_A(P)$, $\mu_B(P)$, and $\mu_{AB}(P)$ respectively. If $c \in C_A \setminus C_0$, $c \in C_B \setminus C_0$, or $c \in C_{AB} \setminus C_0$, then c is redundant wrt. $\mu_0(P)$.*

Proof. We give a proof for the case $c \in C_A \setminus C_0$. The proof idea works for the other cases as well.

We know that $\mu_A(P) \equiv \mu_0(P)$ and hence every solution to $\mu_0(P)$ satisfies c or, equivalently, $\mathrm{sol}(\mu_0(P)) \subseteq \mathrm{sol}(\mu_0(P) \oplus \{c\})$, as required. $c \notin C_0$ follows from the choice of c. □

4.4.4 Operational Concerns

Let P be a school-timetabling problem. $\mu_A(P)$, $\mu_B(P)$, and $\mu_{AB}(P)$ are very likely to include `tpp` constraints that are useless from an operational point of view:

- Both $\mu_A(P)$ and $\mu_B(P)$ may include single-track `tpp` constraints. In the case of $\mu_A(P)$, these constraints go back to members of $I_A(P)$ that have not been merged with other members of $I_A(P)$ in the course of computing $I_A(P)\downarrow_{A,P}$. In the case of $\mu_B(P)$, they go back to singleton members of $I_B(P)$. Irrespective of P, all these constraints are trivially satisfied by any variable assignment and hence their propagation is futile.

- $\mu_B(P)$ may include `tpp` constraints that are subsumed by other `tpp` constraints. A case of subsumption is present if the track set of a `tpp` constraint is a subset of another `tpp` constraint's track set. Every occurence of subsumption traces back to a pair of TPPs that have been reduced with the same couplings. The propagation of subsumed constraints is useless because it does not lead to additional pruning in comparison to what the propagation of the subsuming constraints achieves anyway.

- $\mu_B(P)$ may include `tpp` constraints where each track holds a complete program of some homogeneous group; if present, these constraints go back to members of $I_B(P)$ that have not been reduced in the course of computing $I_B(P)\downarrow_{B,P}$. The propagation of such constraints by means of the `tpp` solver is useless because it does not result in additional pruning in comparison to what the solver in charge of the single groups' capacity constraints achieves anyway.

- Since μ_{AB} connects the inference processes of μ_A and μ_B in series, $\mu_{AB}(P)$ may contain useless `tpp` constraints of all three kinds.

To save memory, we are free to leave out the useless `tpp` constraints. Extending the model generators with appropriate filters is safe because not adding constraints does not restrict solution spaces, of course.

4.5 Non-Conservative Model Extensions

On the basis of μ_0, this section defines model generators that sacrifice solutions by imposing non-redundant `tpp` constraints in a systematic way. The resulting constraint models are leaner and have smaller search spaces in comparison to what μ_0 produces. This entails considerable operational benefits as our computational study will demonstrate.

To generate non-redundant `tpp` constraints, we introduce the reduction $\rightarrow_{C,P}$ for any school-timetabling problem P. $\rightarrow_{C,P}$ is a variant of $\rightarrow_{A,P}$; in comparison to $\rightarrow_{A,P}$, it allows for more transitions because of weaker preconditions and hence $\rightarrow_{A,P}\subseteq\rightarrow_{C,P}$.

Definition 4.11. Let $(A(P), \rightarrow_{C,P})$ be a reduction system with $R_0 \rightarrow_{C,P} R_1$ iff $J_0, J_1 \in R_0$ exist s.t.

- $J_0 \neq J_1$;
- either $J_0 \cap J_1 \neq \emptyset$ or $j_0 \in J_0$, $j_1 \in J_1$, $s_0 \in S(j_0)$, and $s_1 \in S(j_1)$ exist with $J(s_0) \setminus \{j_0\} = J(s_1) \setminus \{j_1\}$; and
- $R_1 = R_0 \setminus \{J_0, J_1\} \cup \{J_0 \cup J_1\}$.

$\rightarrow_{C,P}$ differs from $\rightarrow_{A,P}$ in that it does not require the students to have identical and tight time frames.

To define model generators on the basis of $\rightarrow_{C,P}$, we need to be sure that every computation converges. This can be shown by reference to the arguments that served to prove the convergence of $\rightarrow_{A,P}$.

Proposition 4.12. *If $J(P)$ is finite, then $\rightarrow_{C,P}$ is convergent.*

Next we define the model generator μ_{AC}. When applied to a school-timetabling problem P, μ_{AC} computes a set of couplings, removes obviously unsatisfiable couplings, and adds the remaining couplings to $\mu_A(P)$ after transforming them into `tpp` constraints. In the definition, $R(j)$ denotes the set of rooms that are suitable for the meetings of any job j.

Definition 4.12. Let

$$\mu_{AC}(P) = \mu_A(P) \oplus \{\|\mathcal{J}(J)\| : J \in I_A(P)\downarrow_{C,P} \wedge f(J)\}$$

where a coupling J passes the filter f iff J satisfies the following conditions:

1. No pair of jobs has a teacher or a student in common and all jobs have the same processing time:

   For all $j_0, j_1 \in J$, if $j_0 \neq j_1$, then $T(j_0) \cap T(j_1) = \emptyset$, $S(j_0) \cap S(j_1) = \emptyset$, and $p(j_0) = p(j_1)$.

2. The bipartite graph $(J, R(P), E)$ with $(j, r) \in E$ iff $r \in R(j)$ has a matching M with $|M| = |J|$.

   This condition is necessary for the existence of a room assignment without double-bookings.

$\mu_{AC}(P)$ is well-defined if $C(P)$ is finite: Then $\mu_A(P)$ is well-defined and $\rightarrow_{C,P}$ is convergent.

The filter built into μ_{AC} is necessary because the couplings created in the course of computation are possibly non-redundant. Adding them without prior checking might render the resulting problem inconsistent, for example by forcing the parallel processing of jobs that have a teacher in common. μ_A does not require this kind of filter because μ_A adds redundant couplings only. Using this filter would even mean to prevent the revelation of inconsistencies. Note that building the filter into $\rightarrow_{C,P}$ would render $\rightarrow_{C,P}$ non-confluent.

Finally, we define the model generator μ_{ABC} which combines the ideas behind μ_{AB} and μ_{AC}. When applied to a school-timetabling problem P, μ_{ABC} extends $\mu_{AC}(P)$ with `tpp` constraints that have been computed in the manner of μ_{AB}.

Definition 4.13. Let

$$\mu_{ABC}(P) = \mu_{AC}(P) \oplus \{\|\mathcal{T}\| : \mathcal{T} \in I_B(Q)\downarrow_{B,Q}\}$$

with $Q = P$ except for

$$C(Q) = I_A(P)\downarrow_{A,P}.$$

The inference process is based on the intermediate problem Q which is obtained from the input P by replacing its couplings with $I_A(P)\downarrow_{A,P}$. $\mu_{ABC}(P)$ is well-defined if $C(P)$ is finite: Then $C(Q)$ is finite, its elements are pairwise disjoint by Proposition 4.7, and thus $\rightarrow_{B,Q}$ is convergent.

4.6 Reducing Memory Requirements

Before handing over a model to our timetabling engine, we perform a post-processing step that exploits couplings to reduce the number of variables and scheduling problems by combining meetings.

Let P be a school-timetabling problem and let X denote the variables of $\mu_0(P)$. We define the postprocessor π_P as a function that applies to any FCSP with variables X. Let C denote the constraints of such a FCSP and consider any $\mathtt{tpp}(\mathcal{T}) \in C$. If a job set $J \subseteq J(P)$ exists s.t.

$$\mathcal{T} = \{\{(\mathtt{S}(m), p(m)) : m \in M(j)\} : j \in J\}$$

and if all the meetings that occur in J have the same duration, then π_P removes the constraint $\mathtt{tpp}(\mathcal{T})$ from C and enforces the coupling J by systematic variable replacements instead. For example, if $J = \{j_1, j_2\}$ with $M(j_1) = \{u_1, u_2\}$, $M(j_2) = \{v_1, v_2\}$, and $p(u_1) = p(u_2) = p(v_1) = p(v_2)$, then the following replacements are performed throughout the model:

$$\begin{aligned} \mathtt{S}(v_1) &\to \mathtt{S}(u_1) \\ \bar{\mathtt{S}}(v_1) &\to \bar{\mathtt{S}}(u_1) \\ \mathtt{S}(v_2) &\to \mathtt{S}(u_2) \\ \bar{\mathtt{S}}(v_2) &\to \bar{\mathtt{S}}(u_2) \end{aligned}$$

If the meetings differ in duration, then the `tpp` constraint is retained.

Definition 4.14. Let $\mu_1(P) = \pi_P(\mu_A(P))$, $\mu_2(P) = \pi_P(\mu_{AC}(P))$, and $\mu_3(P) = \pi_P(\mu_{ABC}(P))$.

Note that the variable replacements carried out by π_P can be considered as a kind of constraint propagation performed prior to search.

4.7 Computational Study

4.7.1 The Objectives

To investigate the operational effects of TPP propagation in school timetabling, we performed a large-scale computational study. More precisely, we investigated three questions:

Q1 How does the propagation of couplings as performed by the model generators μ_1 and μ_2 affect the model size?

Q2 How does the propagation of couplings as performed by the model generators μ_1 and μ_2 affect the probability that a problem can be solved?

Q3 How does the propagation of TPPs other than couplings affect the probability that a problem can be solved and which combination of propagation rules performs best?

Due to limited computational resources, we did not employ a full factorial design wrt. the third question but confined ourselves to the separate investigation of $\rightarrow_{\text{PVS}}$, $\rightarrow_{\text{PVSB}}$, $\rightarrow_{\text{FC}}$, $\rightarrow_{\text{NC}}$, and $\bigcup\{\rightarrow_{\text{PVS}}, \rightarrow_{\text{FC}}, \rightarrow_{\text{NC}}\}$. We did not investigate $\rightarrow_{\text{IPT}}$ because, in our application, the processing times of tracks are fixed.

		R_1	R_2	R_3	R_4	R_5	R_6	A_1	A_2	A_3	A_4
	Programs[a]	2	5	3	3	4	7	3	5	7	7
	Labs[b]	17	19	23	13	13	18	19	23	19	23
	Classrooms	23	25	45	26	38	47	25	45	25	45
	Students	540	718	1165	740	760	1030	718	1165	718	1165
	HGs[c]	204	226	314	248	184	382	216	323	232	321
	Teachers	57	64	91	63	64	86	63	94	64	93
	Modules	319	328	447	328	316	401	329	450	330	453
	Lessons	707	779	1156	790	798	1053	785	1156	786	1151
μ_0	Variables	1414	1558	2312	1580	1596	2106	1570	2312	1572	2302
	SPs[d]	795	874	1288	917	808	1328	862	1306	876	1273
μ_1	Couplings	21	23	37	33	22	38	22	40	23	31
	Variables	1358	1472	2157	1452	1484	1932	1488	2161	1491	2168
	SPs	697	739	1084	752	687	1113	732	1094	750	1093
μ_2, μ_3	Couplings	45	49	71	59	44	74	48	74	47	61
	Variables	1300	1412	2057	1389	1424	1815	1423	2069	1433	2083
	SPs	477	487	669	479	479	620	483	680	498	686
μ_3	TPPs[e]	56	69	100	81	61	114	63	108	74	97

Table 4.5:
Characteristics of schools, problems, and models (μ_0, μ_1, μ_2, and μ_3) from top to bottom. Problem and model characteristics are described in terms of their medians.

[a] This row gives the number of programs resulting from what study directions and language curricula are offered and how these may combined. Other factors that increase the number of programs (like the segregation of the sexes in physical education, c.f. Section 2.2.1) have not been considered here.
[b] Including craft rooms, sports facilities, etc.
[c] HG = homogeneous group
[d] SP = scheduling problem
[e] The number of TPPs includes the number of couplings.

4.7.2 The Problem Set

To ensure the practical relevance of our results, problems close to reality have been generated randomly on the basis of detailed models of representative schools. We modeled German secondary schools of the Gymnasium type (cf. Section 2.2). A school model describes the options offered to the students, the facilities, the teacher population, the student population, and various requirements of the school management that concern the design of timetables. We modeled ten secondary schools (R_1 through R_6, A_1 through A_4) that are briefly described in Table 4.5. Schools R_1 through R_6 have real-world counterparts while schools A_1 through

A_4 are artificial schools that were created by hybridizing schools in the following way:

- A_1 hybridizes R_2 and R_3. The facilities and the student numbers are inherited from R_2, the study options are inherited from R_3.
- A_2 hybridizes R_2 and R_3. The facilities and the student numbers are inherited from R_3, the study options are inherited from R_2.
- A_3 hybridizes R_2 and R_5. The facilities and the student numbers are inherited from R_2. A_3 offers all the study options that its parents offer.
- A_4 hybridizes R_3 and A_3. The facilities and the student numbers are inherited from R_3, the study options are inherited from A_3.

For each school, we generated and tested 1000 problems with the following requirements of timetables:

- Most teachers with less than seventeen teaching periods a week need a day off and for some teachers this day is fixed.
- Some teachers are not available before 9 a.m. and some teachers are not available for Friday afternoon.
- The working time of teachers must not exceed six periods a day.
- The daily working time of the students with loose time frames (those of the upper grades) must not be lower than four periods and must not exceed eight periods.
- There are job-specific bounds on the daily number of teaching periods: This number must not exceed two in any case and is limited to one if the lessons' total duration equals two.
- Individual timetables must not contain more than six periods of idle time a week.
- In the upper grades, double lessons are compulsory in most cases. In the lower grades, however, double lessons are required only for physical education.
- Some schools have access to facilities of other public institutions for settled periods of time which have to be considered in timetabling.

Moreover, note that our problems do not contain couplings. Couplings are generated and added only when a problem is prepared for submission to a problem solver.

4.7.3 The Timetabling Engine

Our timetabling engine embeds constraint propagation into chronological backtracking. At each node of the search space, a task is chosen and scheduled and rooms are allocated. All decisions required to unfold the search space are guided by strategies. Constraint propagation takes place after each commitment that is issued to the constraint solver and is performed in a fixed-point manner.

`gcc` constraints are propagated by means of network-flow techniques[1] [Rég94, Rég96]. `disjoint` constraints are propagated by means of value sweep pruning[3] [BC01]. Our TPP solver implements $\rightarrow_{\mathrm{PVS}}$, $\rightarrow_{\mathrm{PVSB}}$, $\rightarrow_{\mathrm{FC}}$, and $\rightarrow_{\mathrm{NC}}$. The implementation of $\rightarrow_{\mathrm{NC}}$ is based on the Ford and Fulkerson algorithm (e.g. [MN99]). The propagation rules can be applied in any combination and the solver can be switched off completely.

In task selection, we exploit heuristic knowledge gained from prior failures. If there are unscheduled tasks that were met at a dead end[4] before (in a chronological sense, not wrt. the path from the root of the search tree to the current node), only this task set is subject to selection. Otherwise, all unscheduled tasks are subject to selection. The search procedure prefers tasks the period-level start-time variables of which have smallest domains. Ties are broken by considering the processing time and the number of resources required; tasks that maximize the product of processing time and resource demand are preferred. Periods are assigned in ascending order. If the task has even duration (double lessons), odd periods are preferred to maximize the utilization of scarce resources. In room allocation, the search procedure prefers rooms that are as small or supply as few equipment as possible.

The timetabling engine comprises about 1500 lines of code and has been built on top of a finite-domain constraint system [COC97] which itself is part of the constraint-logic programming environment SICStus Prolog 3.9.0 [Int02]. The TPP solver accounts for about 600 lines of code.

---

[1] More precisely, we used the `all_distinct` and `global_cardinality` constraints as provided by SICStus Prolog 3.9.0 [Int02] that maintain domain consistency[2] for `alldiff` [Rég94, vH01] and global cardinality constraints [Rég96], respectively. SICStus Prolog 3.9.0 propagates both types of constraints by means of algorithms due to Régin [Rég94, Rég96].

[2] Suppose $P = (X, \delta, C)$ is a FCSP and $c = p(x_1, \ldots, x_n) \in C$. According to van Hentenryck et al. [VSD98], c ***is domain-consistent in*** P, if, for each variable x_i and for each value $v_i \in \delta(x_i)$, there exist values $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n$ in $\delta(x_1), \ldots, \delta(x_{i-1}), \delta(x_{i+1}), \ldots, \delta(x_n)$ s.t. $p(v_1, \ldots, v_n)$ holds.

[3] More precisely, we used the `disjoint2` constraint as provided by SICStus Prolog 3.9.0 [Int02]. SICStus Prolog 3.9.0 propagates this type of constraint by means of an algorithm due to Beldiceanu & Carlsson [BC01].

[4] A ***dead end*** is a state of search where some variable cannot be assigned a value without some constraint solver detecting an inconsistency.

	μ_0	μ_1	μ_3					
	S_1	S_1	S_1	S_2	S_3	S_4	S_5	S_6
R_1	13.5 (2.16)	14.4 (2.22)	77.5 (2.64)	77.5 (2.64)	77.5 (2.64)	77.5 (2.64)	77.5 (2.64)	77.5 (2.64)
R_2	21.5 (2.60)	30.2 (2.91)	71.8 (2.85)	83.9 (2.33)	75.0 (2.74)	71.9 (2.84)	71.8 (2.85)	84.1 (2.31)
R_3	3.4 (1.15)	23.8 (2.69)	82.0 (2.43)	83.3 (2.36)	83.4 (2.35)	82.0 (2.43)	82.0 (2.43)	83.3 (2.36)
R_4	13.5 (2.16)	32.1 (2.95)	78.2 (2.61)	79.5 (2.55)	78.6 (2.60)	78.2 (2.61)	78.2 (2.61)	79.5 (2.55)
R_5	17.7 (2.42)	29.7 (2.89)	87.4 (2.10)	86.8 (2.14)	87.6 (2.09)	87.4 (2.10)	87.4 (2.10)	86.8 (2.14)
R_6	1.9 (0.86)	12.3 (2.08)	62.0 (3.07)	60.6 (3.09)	61.3 (3.08)	62.0 (3.07)	62.0 (3.07)	60.6 (3.09)
A_1	25.4 (2.75)	39.0 (3.09)	85.8 (2.21)	86.7 (2.15)	86.1 (2.19)	85.8 (2.21)	85.8 (2.21)	86.7 (2.15)
A_2	5.4 (1.43)	20.9 (2.57)	74.6 (2.75)	84.0 (2.32)	76.6 (2.68)	74.8 (2.75)	74.6 (2.75)	83.9 (2.33)
A_3	17.2 (2.39)	24.9 (2.74)	66.2 (2.99)	71.7 (2.85)	65.8 (3.00)	66.2 (2.99)	66.2 (2.99)	71.6 (2.85)
A_4	6.8 (1.59)	19.0 (2.48)	63.7 (3.04)	72.4 (2.83)	64.2 (3.03)	63.7 (3.04)	63.7 (3.04)	72.4 (2.83)

Table 4.6:
Probability of solving a problem ($100\bar{x}$ with $200\hat{\sigma}_{\bar{x}}\%$ in parentheses)

4.7.4 Results

The effects of TPP propagation on model size are reported in Table 4.5. We observe that, for the grades five through ten, where all classes have tight time frames, μ_1 produces less variables and scheduling problems than μ_0. Moreover, by assuming tight time frames for all classes, μ_2 reduces the model size considerably in comparison to μ_1. Note that Table 4.5 does neither consider TPPs with only one track, nor TPPs where each track holds a complete program of some homogeneous group, nor subsumed TPPs (c.f. Section 4.4.4).

Table 4.6 reports the effects of TPP propagation on the reliability[5] of our timetabling engine. We tested six solvers that differ in how `tpp` constraints are propa-

[5]Here ***reliability*** refers to the probability of solving a problem from a certain problem class. For example, we say that solver S_0 is more reliable than solver S_1 wrt. a certain problem class C, if S_0 is more likely to solve problems from C than S_1.

gated. S_1 does not propagate `tpp` constraints while, according to our experimental design outlined earlier, S_2 through S_6 employ $\rightarrow_{\mathrm{PVS}}$, $\rightarrow_{\mathrm{PVSB}}$, $\rightarrow_{\mathrm{FC}}$, $\rightarrow_{\mathrm{NC}}$, and $\bigcup\{\rightarrow_{\mathrm{PVS}}, \rightarrow_{\mathrm{FC}}, \rightarrow_{\mathrm{NC}}\}$, respectively. Table 4.6 reports the percentage of solved problems out of 1000 together with the 95% confidence intervals[6]. Runs were constrained to abort after 1000 dead ends[7]. Due to the large sample size, the 95% confidence intervals are quite narrow. We make the following observations:

- The propagation of couplings derived and added by μ_A considerably increases the reliability of our timetabling engine.

- The propagation of couplings derived and added by μ_{AC} tremendously increases the reliability of our timetabling engine.

- The propagation of TPPs other than couplings pays off only for the schools R_2, A_2, A_3, and A_4. $\rightarrow_{\mathrm{PVS}}$ is effective for each of these schools while $\rightarrow_{\mathrm{PVSB}}$ is effective for school R_2 only. So domain reasoning is more effective than bound reasoning. The other reductions do not contribute considerably, not even when applied in combination.

- In no case, the reliability of our timetabling engine was considerably impaired by TPP propagation.

Figures 4.1 through 4.10 show plots that give more insight into the search process by relating the number of solved problems

- to the number of dead ends that were encountered during search and

- to the number of backtracking steps that were necessary to obtain a solution.

When interpreting the results, keep in mind that bounds on idle time were ignored.

[6]Suppose we are given a sample with mean $\bar{x}$ and standard error $\hat{\sigma}_{\bar{x}}$. Then the ***95% confidence interval*** (e.g. [Coh95]) is approximately $\bar{x} \pm 2\hat{\sigma}_{\bar{x}}$. This interval is likely to contain the population mean μ with 95% probability. For example, according to Table 4.6, $\mu_3 + S_1$ solved $\bar{x} = 77.5\%$ of the R_1 problems. In this case, $2\hat{\sigma}_{\bar{x}} \approx 2.64\%$. This means that we can be 95% sure that, in the long term, S_1 will be able to solve $77.5\% \pm 2.64\%$ of the R_1 problems.

[7]Note that the number of dead ends is different from the number of backtracking steps. With a limit of 1000 dead ends, the timetabling engine backtracked up to 40000 times to come up with a solution.

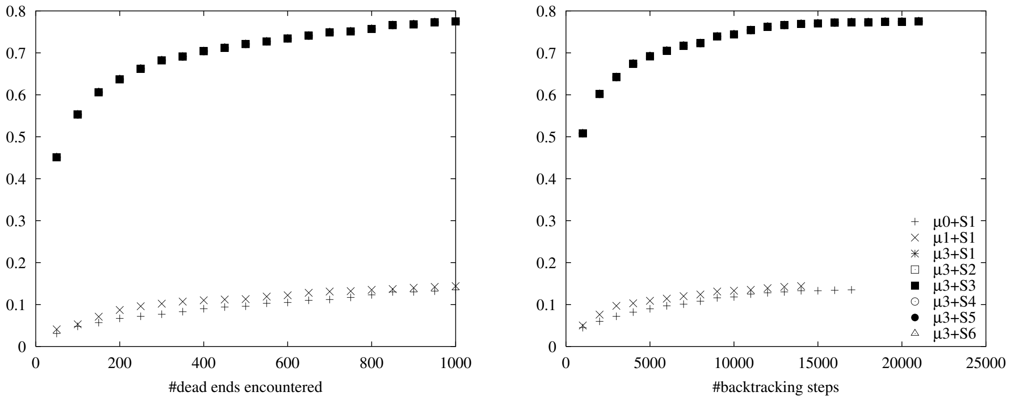


Figure 4.1: Behavior of solvers for school R_1

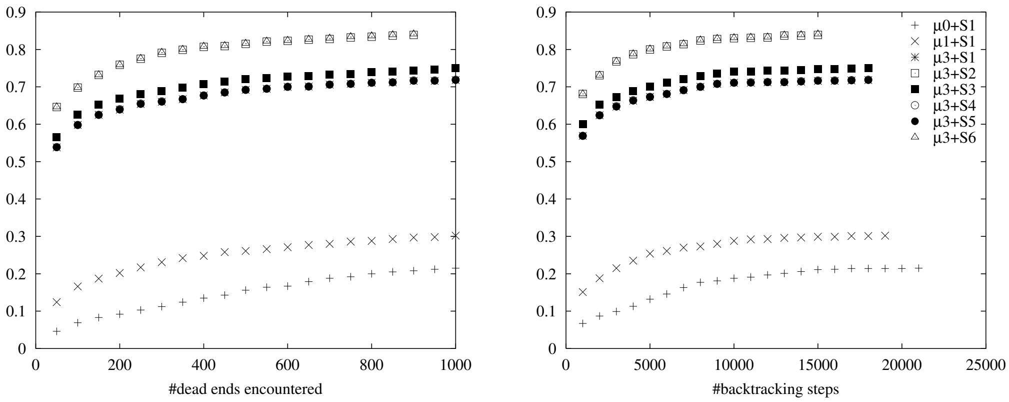


Figure 4.2: Behavior of solvers for school R_2

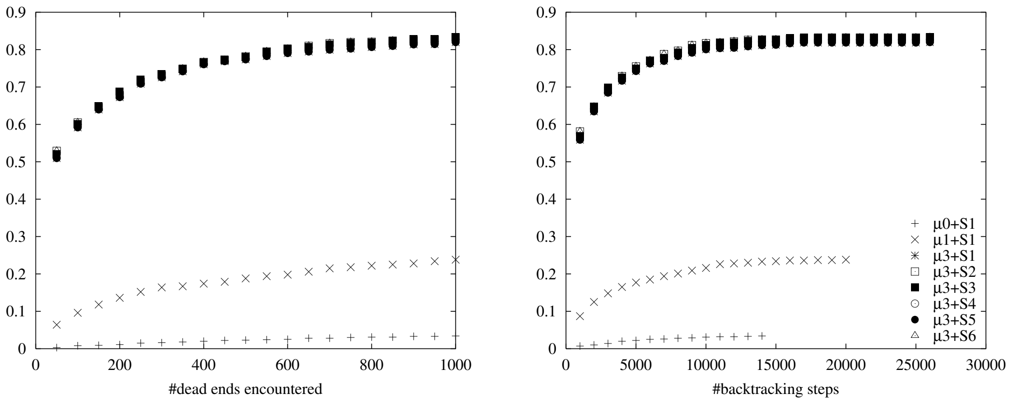


Figure 4.3: Behavior of solvers for school R_3

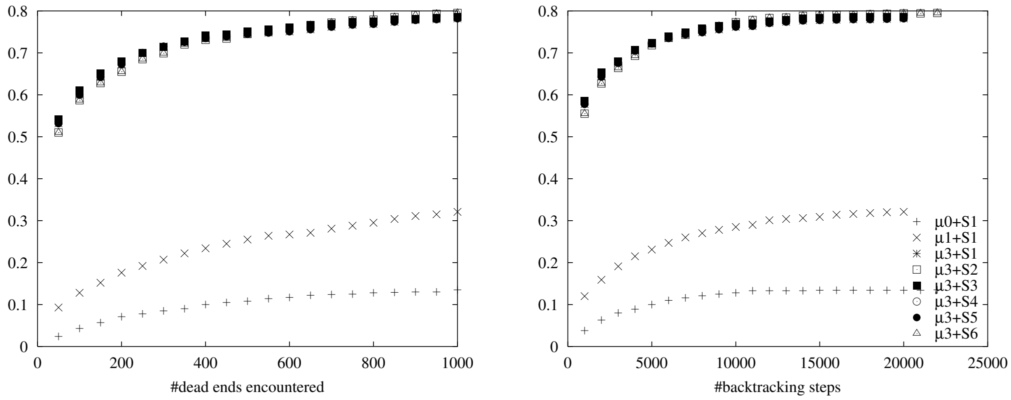


Figure 4.4: Behavior of solvers for school R_4

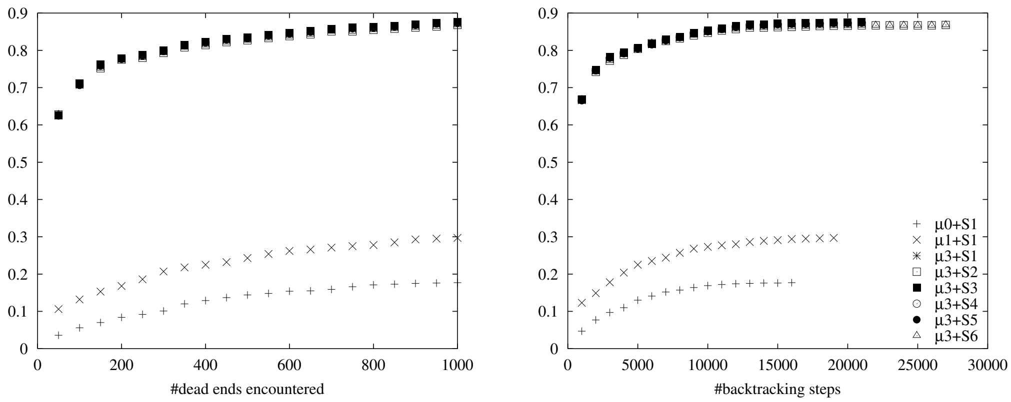


Figure 4.5: Behavior of solvers for school R_5

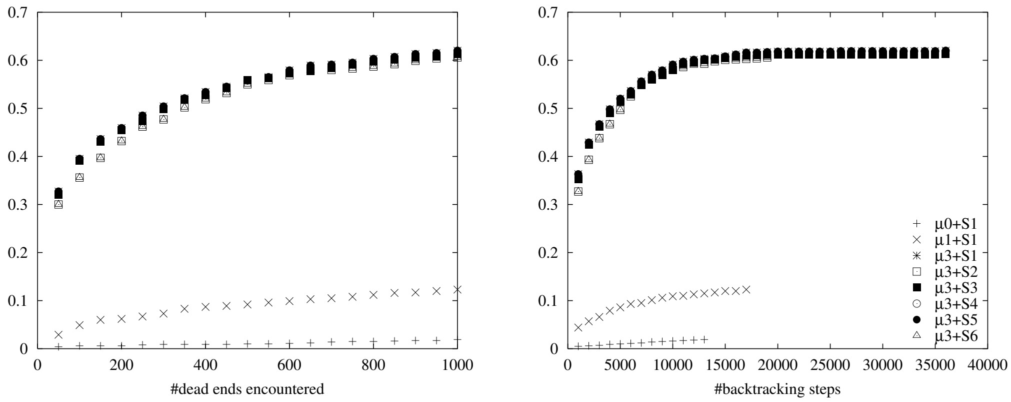


Figure 4.6: Behavior of solvers for school R_6

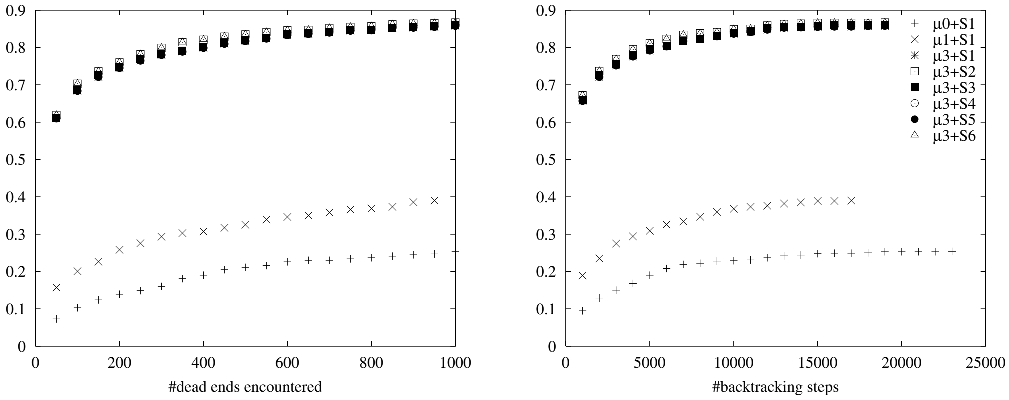


Figure 4.7: Behavior of solvers for school A_1

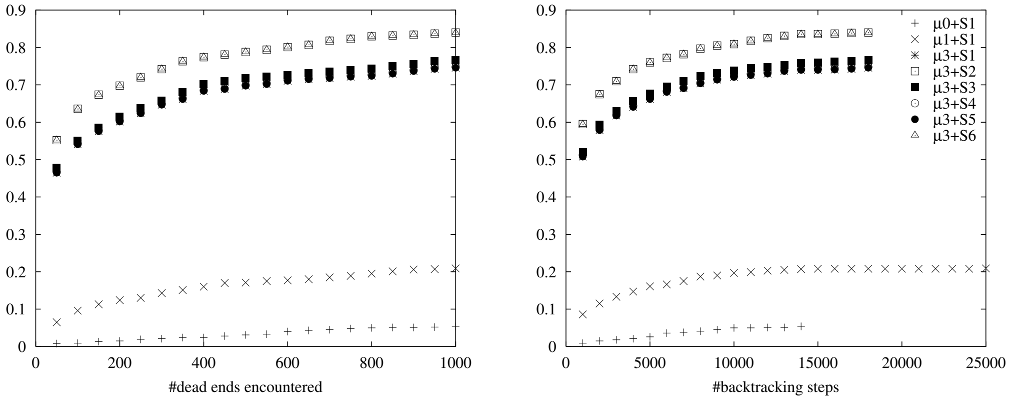


Figure 4.8: Behavior of solvers for school A_2

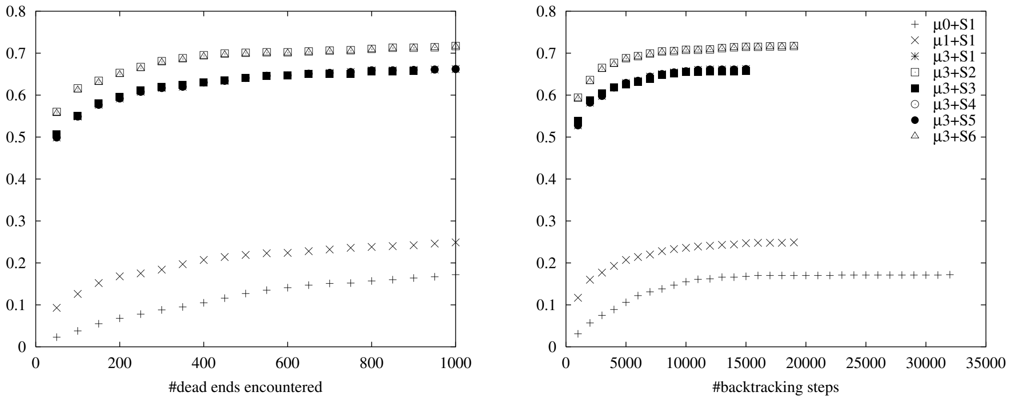


Figure 4.9: Behavior of solvers for school A_3

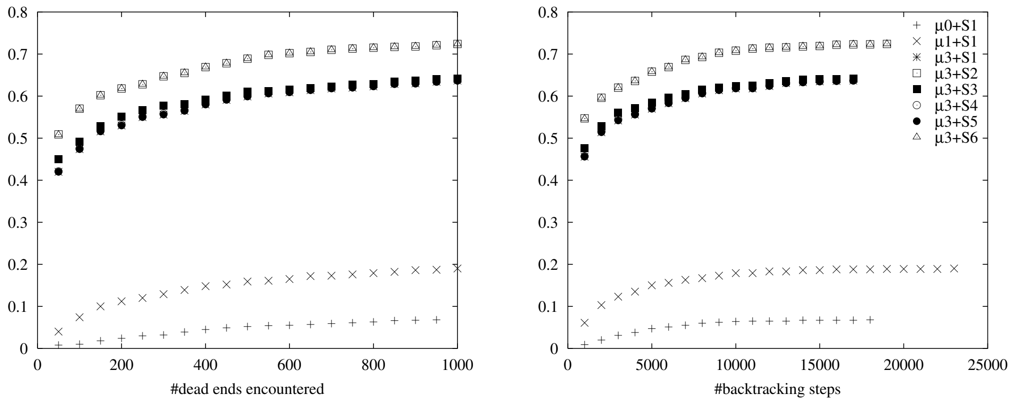


Figure 4.10: Behavior of solvers for school A_4

4.8 Related Work

Couplings have occurred in publications on school timetabling as part of the problem statement but only limited attention was paid to them:

- In his computational study of a timetabling system based on tabu search, Costa [Cos94] investigated two timetabling problems from Switzerland. He reports that about 10% and 15%, respectively, of the lessons are involved in couplings. He notes that couplings with three classes and four groups complicated timetabling because they required to find a fourth free room (in addition to the classrooms).

- Schaerf [Sch96b] stated that couplings "are necessary for tackling practical cases" and reported that couplings in physical education complicated timetabling considerably.

- Drexl & Salewski [DS97] utilized couplings to down-size their mathematical-programming models but neither discussed nor investigated the operational effects of this transformation.

- Kaneko et al. [KYN99] used a greedy algorithm to compute a probably inconsistent initial assignment for a local search procedure. Initially and after every commitment, the greedy algorithm established arc-consistency and it seems that couplings were considered in this step.

- Fernandes et al. [FCMR99] employed couplings to obtain more concise genetic timetable representations but neither discussed nor investigated the operational effects of this transformation.

Drexl & Salewski and Schaerf were required to produce compact student timetables and their respective comments lead to the conclusion that they actually dealt with redundant couplings. The problem descriptions of Costa, Kaneko et al., and Fernandes et al. are not detailed enough to allow for such a conclusion.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we developed and studied constraint-based solvers for school timetabling. To facilitate the computational studies, a lot of infrastructure had to be developed. In the following, we detail on our achievements.

- Emphasizing modeling aspects, we demonstrated how to tackle school-timetabling problems by means of constraint-programming technology. We considered various requirements that frequently occur in school timetabling including compactness and distribution constraints like bounds on daily work load and bounds on the weekly number of working days.

  We developed a basic model generator and a suitable timetabling engine, both on the basis of global constraints. The problem solver resulting from their combination performed pretty poorly in terms of reliability.

  We continued with a series of modifications to the basic model generator with the aim to produce operationally enhanced constraint models. In this process, the track parallelization problem (TPP) played a central role. We proceeded as follows:

  1. We developed a convergent reduction system to infer redundant couplings (a special kind of TPP). These couplings served to reduce the number of variables and scheduling problems by variable replacement. This way we obtained smaller constraint models and better albeit still poor results.
  2. We traded completeness for efficiency: We sacrificed solutions by imposing non-redundant couplings in a systematic way. Exploiting them

like redundant constraints, we obtained even smaller constraint models and, finally, promising results. Of course, though this measure increases reliability on average, it may rule out all solutions to a given problem. However, this is no obstacle to practical problem solving because we are always free to fall back to a model generator that preserves solutions.

3. We developed a convergent reduction system to infer more redundant TPPs on the basis of redundant couplings. These TPPs were propagated at each search node to prune the search space. This measure increased our problem solver's reliability even further.

Because of its generality, our basic model generator is suitable for a large variety of educational institutions. The rules for TPP inference even apply to any educational institution. The question arises whether our problem solver applies to problem classes other than those we investigated, namely German secondary schools. The variety of the schools (with regard to student numbers, educational programs, and facilities) that served as basis for our computational study gives cause to an affirmative answer which, of course, has to be considered a hypothesis as long as it has not been confirmed empirically.

- We introduced the `tpp` constraint along with a suitable solver for modeling and solving TPPs in a finite-domain constraint-programming framework. We demonstrated correctness and gave some performance guarantees including convergence.

- In our computational studies, we obtained results that are both reliable from a statistical point of view and practically relevant. We modeled ten representative German secondary schools and, for each school, we generated and tested 1000 timetabling problems. This problem set may serve as a reference point for future research in automated school timetabling. It facilitates the assessment and the comparison of timetabling algorithms. Its publication on the Web is in preparation.

- Last but not least, we gave a new condition that is sufficient for local confluence and, by applying it to the TPP solver, we demonstrated that it is well suited to study the behavior of cooperating constraint solvers.

5.2 Future Work

As explained in Section 1.2, there are many dimensions to explore in problem-solver design like models, constraint-propagation algorithms, and branching strategies. This thesis dealt with modeling aspects but did not explore the other dimensions. The composition of our timetabling engine (cf. Section 4.7.3) is rather based on results from non-systematic pilot studies. (This procedure is not unusual, cf. [DS97].) Hence it is near at hand to investigate the contribution of each single component:

- Branching strategies: Our strategies take problem characteristics into account. How do they compare to more standard strategies? What is the impact of preferring lessons that were met at a dead end?

- Constraint propagation: We employ network-flow techniques [Rég94, Rég96] that achieve maximum consistency. How do they compare to other methods that achieve lower levels of consistency?

Moreover, the following issues require consideration:

- Bounding idle time: We ignored bounds on idle time because there is no way to enforce them by means of pre-implemented constraints. This lack of functionality affects teachers and students with loose time frames. For all of them, we obtained timetables with too much idle time.

  To obtain a practically useful problem solver, it is crucial to develop means to control idle time. This kind of research cannot be conducted without suitable infrastructure. We need problem sets to test new algorithms and we need a good understanding of how to cope with the other requirements. By providing this infrastructure, this thesis prepared the grounds for research in controlling idle time.

- Constraints that directly affect job timetables: We presented model generators that deal with the common core of school-timetabling problems (c.f. Section 2.1) except for bounds on idle time. This core comprises all the constraints that occur frequently. Yet constraints that directly affect job timetables are not contained in the common core (except for bounds on the daily number of teaching periods) because of their diversity. Nevertheless, such constraints are required in some form or another.

- Interactivity: An interactive timetabling system allows the user to interfere in the timetabling process and supports manual changes to complete timetables. The need for manual changes may arise in the following situations:

- The timetable is finished and probably in operation already. Suddenly timetabling data changes (like a teacher is replaced by another one) and the timetable has to be adapted to the changes.
- The timetable is finished but violates some rare requirement that was not stated explicitely. Maybe it only became evident when looking at the timetable or the timetabling system did not allow to state it.
- The timetabling system runs forever without producing a (good) solution because the problem is inconsistent or because the algorithm is unsuitable.

Schaerf [Sch96b] stated that "the ability to work interactively is widely recognized as crucial for timetabling systems in the research community".

Being based on chronological backtracking, constraint-programming systems are not very well equipped for interactive problem solving. To move a lesson, all decisions met after placing the lesson have to be recorded, undone, and replayed after moving the lesson. To add or remove a constraint, the whole timetabling process has to be restarted. Unacceptable reaction times may result.

- Support of rare requirements: In Section 2.1, we identified constraints that frequently occur in school-timetabling. If a timetabling system can handle these constraints, it is very likely to be useful. But what about rare requirements? Provided that a language exists to express school-specific requirements, how to propagate them? Generic propagation algorithms (e.g. [BR99]) always have exponential worst case complexity and hence only apply to constraints with few variables that have small domains. In our case, they could apply to job-specific constraints on day level. Alternatively, branch & bound could be used to optimize an objective function that contains all constraints that cannot be dealt with otherwise. However, chronological backtracking is likely to have a very hard time to escape local optima.

The latter issues could be addressed by limiting the role of constraint programming to compute a high-quality initial assignment that serves as a starting point for incremental improvements by interleaving local search with user activity.

Appendix A

A Modular Approach To Proving Confluence

We are interested in investigating the confluence properties of cooperating constraint solvers. If a system of constraint solvers is confluent, then the result of constraint propagation does not depend on how the solvers are scheduled. If it is either known to diverge or if it is neither known to be confluent nor to diverge, then the question arises which scheduling strategy will perform best. This lack of knowledge may be very inconvenient in empirical research and application development as it potentially adds another dimension to the design space.

To establish confluence properties, we suppose that solvers are modeled as reductions that transform constraint networks (cf. Section 1.7), we define the notion of ***insensitivity*** to a superset relation, and show that, if each solver of a given set of solvers is ***insensitive*** to the same terminating superset relation, then any combination of these solvers is confluent.

We apply our approach to the TPP solver which has been presented in Section 3.3. This solver consists of several reductions and we demonstrate its confluence and the confluence of any subset of its reductions with a number of proofs linear in the number of reductions.

This chapter is organized as follows. Section A.1 introduces the concept of insensitivity and relates it to the concept of strong commutation. Section A.2 presents our method for proving confluence. In Section A.3, we investigate the confluence properties of the TPP solver. In Section A.4, we present related work and compare to it. Section A.5 summarizes and closes with perspectives for future work.

A.1 Insensitivity

Intuitively, $\to_1$ is insensitive to $\to_2$, if the inference capabilities of $\to_1$ are preserved under application of $\to_2$.

Definition A.1. Let $(A, \to_1)$ and $(A, \to_2)$ be reduction systems. We say that $\to_1$ ***is insensitive to*** $\to_2$ iff the following requirements are satisfied.

1. If $y \leftarrow_1 x \to_2 z$, $y \neq z$, and $z \to_2 y$, then $z \to_1 y$.

$$\begin{tikzcd} x \arrow[r, "1"] \arrow[dr, "2"'] & y \\ & z \arrow[u, "2"'] \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} x \arrow[r, "1"] \arrow[dr, "2"'] & y \\ & z \arrow[u, "1"'] \end{tikzcd}$$

2. If $y \leftarrow_1 x \to_2 z$, $y \neq z$, $y \not\to_2 z$, and $z \not\to_2 y$, then $u \in A$ exists s.t. $y \to_2 u \leftarrow_1 z$.

$$\begin{tikzcd} x \arrow[r, "1"] \arrow[dr, "2"'] & y \\ & z \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} x \arrow[r, "1"] \arrow[dr, "2"'] & y \arrow[r, "2"] & u \\ & z \arrow[ur, "1"'] & \end{tikzcd}$$

Corollary A.1. *If $\to_1$ and $\to_2$ are insensitive to $\to_3$, then $\to_1 \cup \to_2$ is insensitive to $\to_3$.*

Corollary A.2. *Let $(A, \to)$ be a reduction system. If $\to$ is insensitive to itself, then it is locally confluent.*

Next we study the relationship of insensitivity to the well-known notion of strong commutation. Like insensitivity, strong commutation is a binary relation on reductions.

Definition A.2. Let $(A, \to_1)$ and $(A, \to_2)$ be reduction systems. We say that $\to_1$ and $\to_2$ ***commute strongly*** iff $y \leftarrow_1 x \to_2 z$ implies $\exists u.\ y \to_2^{=} u \leftarrow_1^{*} z$.

Proposition A.1. *Let $(A, \to_1)$ and $(A, \to_2)$ be reduction systems. If $\to_1$ is insensitive to $\to_2$, then $\to_1$ and $\to_2$ commute strongly.*

Proof. We have to show that $y \leftarrow_1 x \to_2 z$ implies $\exists u.\ y \to_2^{=} u \leftarrow_1^{*} z$. If $y = z$, we are done. Otherwise, there are three cases. If $y \to_2 z$, we are done. If $z \to_2 y$, then $z \to_1 y$ because $\to_1$ is insensitive to $\to_2$. If neither $y \to_2 z$ nor $z \to_2 y$, then $u \in A$ exists s.t. $y \to_2 u \leftarrow_1 z$ because $\to_1$ is insensitive to $\to_2$. $\square$

Proposition A.2. *Let $(A, \to_2)$ be a reduction system and let $\to_1 \subseteq \to_2$ be a transitive reduction s.t. $\to_1$ and $\to_2$ commute strongly. If $y \leftarrow_1 x \to_2 z$, $y \neq z$, $y \not\to_2 z$, and $z \not\to_2 y$, then $u \in A$ exists s.t. $y \to_2 u \leftarrow_1 z$.*

Proof. By strong commutation, we know that $u \in A$ exists s.t. $y \to_2^= u \leftarrow_1^* z$. Suppose $u = z$. Then $y \to_2^= z$. Because $y \neq z$, $y \to_2 z$. This contradicts the premise and thus $z \to_1^+ u$. Because $\to_1$ is transitive, $z \to_1 u$. Suppose $u = y$. Then $z \to_1 y$ and thus $z \to_2 y$ because $\to_1 \subseteq \to_2$. This contradicts the premise and thus $y \to_2 u$. □

A.2 Confluence Through Insensitivity

Theorem A.1. *Let $(A, \to_2)$ be a terminating reduction system. If $\to_1 \subseteq \to_2$ is insensitive to $\to_2$, then $\to_1$ is locally confluent.*

Proof. For each pair $(y,z) \in A \times A$ s.t. $\exists x.\ y \leftarrow_1 x \to_2 z$, $\exists x.\ y \leftarrow_2 x \to_1 z$, $y \neq z$, $y \not\to_2 z$, and $z \not\to_2 y$, choose a pair $(\hat{y}, \hat{z}) \in A \times A$ s.t. $y \to_2 \hat{y} \leftarrow_1 z$ and $z \to_2 \hat{z} \leftarrow_1 y$. This is possible because $\to_1$ is insensitive to $\to_2$.

Let $(x_2, y_2) \prec (x_1, y_1)$ iff $x_1 \to_2 x_2$. $\prec$ is well-founded because $\to_2$ is terminating. By well-founded recursion on $\prec$, we define $s(y,z)$ for all (y,z) that satisfy $\exists x.\ y \leftarrow_1 x \to_2 z$ and $\exists x.\ y \leftarrow_2 x \to_1 z$:

$$s(y,z) = \begin{cases} (y,z), & \text{if } y = z,\ y \to_2 z, \text{ or } z \to_2 y \\ (y,z), s(\hat{y},\hat{z}) & \text{otherwise} \end{cases}$$

In the second case, $s(\hat{y},\hat{z})$ is well-defined because $(\hat{y},\hat{z})$ exists, $\hat{y} \leftarrow_1 z \to_2 \hat{z}$, and $\hat{y} \leftarrow_2 y \to_1 \hat{z}$. Since s is defined by well-founded recursion on $\prec$, $s(y,z)$ is finite for all (y,z) that s is defined for.

Let $y_1 \leftarrow_1 x \to_1 z_1$. $s(y_1, z_1)$ is well-defined because $\to_1 \subseteq \to_2$ and thus $y_1 \leftarrow_1 x \to_2 z_1$ and $y_1 \leftarrow_2 x \to_1 z_1$. Let $n > 0$ s.t. $s(y_1, z_1) = (y_1, z_1), \ldots, (y_n, z_n)$. We observe that, for all $1 \le k < n$, $y_k \to_2 y_{k+1} \leftarrow_1 z_k$ and $z_k \to_2 z_{k+1} \leftarrow_1 y_k$, and that $y_n = z_n$, $y_n \to_2 z_n$, or $y_n \leftarrow_2 z_n$. The following figure shows a situation where $n \ge 5$.

$$\begin{tikzcd}
x \arrow[r, "1"] \arrow[dr, "1"'] & y_1 \arrow[r, "2"] \arrow[dr, "1"] & y_2 \arrow[r, "2"] \arrow[dr, "1"] & y_3 \arrow[r, "2", "+"'] & y_{n-1} \arrow[r, "2"] \arrow[dr, "1"] & y_n \arrow[d, bend left, leftrightarrow, "2"] \arrow[d, phantom, shift left=6, "="] \\
& z_1 \arrow[r, "2"'] \arrow[ur, "1"'] & z_2 \arrow[r, "2"'] \arrow[ur, "1"'] & z_3 \arrow[r, "2"', "+"] & z_{n-1} \arrow[r, "2"'] \arrow[ur, "1"'] & z_n
\end{tikzcd}$$

It remains to show that $y_1 \downarrow_1 z_1$. If n is odd, then $y_1 \to_1^* y_n$ and $z_1 \to_1^* z_n$. If n is even, then $y_1 \to_1^* z_n$ and $z_1 \to_1^* y_n$. If $y_n = z_n$, we are done. If $y_n \to_2 z_n$, then $y_n \to_1 z_n$ because $y_n \leftarrow_2 y_{n-1} \to_1 z_n$ and $\to_1$ is insensitive to $\to_2$. If $z_n \to_2 y_n$, then $z_n \to_1 y_n$ because $z_n \leftarrow_2 z_{n-1} \to_1 y_n$ and $\to_1$ is insensitive to $\to_2$. □

The following result is obtained by applying Newman's Lemma. Newman's Lemma states that a terminating reduction is confluent if it is locally confluent.

Corollary A.3. *Let $(A, \to_2)$ be a terminating reduction system. If $\to_1 \subseteq \to_2$ is insensitive to $\to_2$, then $\to_1$ is confluent.*

A.3 Confluence Properties of the `tpp` Solver

We show that each combination of $\rightarrow_{\text{PVS}}$, $\rightarrow_{\text{IPT}}$, $\rightarrow_{\text{FC}}$, and $\rightarrow_{\text{NC}}$ is confluent. We proceed as follows: We show that $\rightarrow_{\text{PVS}}$, $\rightarrow_{\text{IPT}}$, and $\rightarrow_{\text{NC}}$ are insensitive to $\rightarrow_{\text{FD}}$ (cf. Corollaries A.4, A.6, and A.7) and that $\rightarrow_{\text{FC}}$ is insensitive to $\rightarrow_{\text{C}}$ (cf. A.5). It turns out that, if a correct reduction is insensitive to $\rightarrow_{\text{FD}}$, then it is insensitive to $\rightarrow_{\text{C}}$ (cf. Lemma A.1). Thus, each of $\rightarrow_{\text{PVS}}$, $\rightarrow_{\text{IPT}}$, $\rightarrow_{\text{FC}}$, and $\rightarrow_{\text{NC}}$ is insensitive to $\rightarrow_{\text{C}}$ and, by Corollary A.1, each combination of $\rightarrow_{\text{PVS}}$, $\rightarrow_{\text{IPT}}$, $\rightarrow_{\text{FC}}$, and $\rightarrow_{\text{NC}}$ is insensitive to $\rightarrow_{\text{C}}$. Then, by Corollary A.3, each combination of $\rightarrow_{\text{PVS}}$, $\rightarrow_{\text{IPT}}$, $\rightarrow_{\text{FC}}$, and $\rightarrow_{\text{NC}}$ is confluent. The proofs make heavy use of monotonicity and correctness properties.

Lemma A.1. *Let $\rightarrow \subseteq \rightarrow_{\text{FD}}$. If $\rightarrow$ is correct and insensitive to $\rightarrow_{\text{FD}}$, then it is insensitive to $\rightarrow_{\text{C}}$.*

Proof. Let $P_1 \leftarrow P_0 \rightarrow_{\text{C}} P_2$ s.t. $P_2 \rightarrow_{\text{C}} P_1$. $P_2 \rightarrow P_1$ because $\rightarrow_{\text{C}} \subseteq \rightarrow_{\text{FD}}$ and $\rightarrow$ is insensitive to $\rightarrow_{\text{FD}}$.

Let $P_1 \leftarrow P_0 \rightarrow_{\text{C}} P_2$ s.t. $P_1 \neq P_2$, $P_1 \not\rightarrow_{\text{C}} P_2$, and $P_2 \not\rightarrow_{\text{C}} P_1$. We have to show that P_3 exists s.t. $P_1 \rightarrow_{\text{C}} P_3 \leftarrow P_2$. By the definitions of correctness and $\rightarrow_{\text{C}}$, $P_1 \not\rightarrow_{\text{C}} P_2$ iff $P_1 \not\rightarrow_{\text{FD}} P_2$ or $\text{sol}(P_1) \neq \text{sol}(P_2)$. However, $\text{sol}(P_1) = \text{sol}(P_0) = \text{sol}(P_2)$ because $P_1 \leftarrow P_0 \rightarrow_{\text{C}} P_2$ and both $\rightarrow$ and $\rightarrow_{\text{C}}$ are correct. Hence $P_1 \not\rightarrow_{\text{FD}} P_2$. By a symmetric argument, $P_2 \not\rightarrow_{\text{FD}} P_1$. Furthermore, $P_0 \rightarrow_{\text{FD}} P_2$ because $\rightarrow_{\text{C}} \subseteq \rightarrow_{\text{FD}}$. By the insensitivity of $\rightarrow$ to $\rightarrow_{\text{FD}}$, P_3 exists s.t. $P_1 \rightarrow_{\text{FD}} P_3 \leftarrow P_2$. Finally, $\text{sol}(P_3) = \text{sol}(P_2)$ because $P_2 \rightarrow P_3$ and $\rightarrow$ is correct. In consequence, $\text{sol}(P_1) = \text{sol}(P_3)$ and thus $P_1 \rightarrow_{\text{C}} P_3$.

We conclude that $\rightarrow$ is insensitive to $\rightarrow_{\text{C}}$. □

Lemma A.2. *Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs with $P_0 \rightarrow_{\text{FD}} P_1$ and let $Y = \{y_1, \ldots, y_n\} \subseteq X$ s.t. $\delta_1(x) = \delta_0(x)$ for all $x \in X \setminus Y$.*

1. *If $P_0 \rightarrow_{\text{FD}} P_2 \rightarrow_{\text{FD}} P_1$, then*

$$\delta_1(y_1) \times \ldots \times \delta_1(y_n) \subset \delta_2(y_1) \times \ldots \times \delta_2(y_n) \subset \delta_0(y_1) \times \ldots \times \delta_0(y_n).$$

2. *If $P_0 \rightarrow_{\text{FD}} P_2$, $P_1 \not\rightarrow_{\text{FD}} P_2$, $P_2 \not\rightarrow_{\text{FD}} P_1$, and $P_1 \neq P_2$, then*

$$\delta_2(y_1) \times \ldots \times \delta_2(y_n) \not\subseteq \delta_1(y_1) \times \ldots \times \delta_1(y_n).$$

3. *If $P_0 \rightarrow_{\text{C}} P_2$, $P_1 \not\rightarrow_{\text{C}} P_2$, $P_2 \not\rightarrow_{\text{C}} P_1$, and $P_1 \neq P_2$, then*

$$\delta_2(y_1) \times \ldots \times \delta_2(y_n) \not\subseteq \delta_1(y_1) \times \ldots \times \delta_1(y_n).$$

Proof.

1. Let $x \in X \setminus Y$. By the definition of $\rightarrow_{\mathrm{FD}}$, $\delta_1(x) \subseteq \delta_2(x) \subseteq \delta_0(x)$. Considering that $\delta_1(x) = \delta_0(x)$, we obtain $\delta_1(x) = \delta_2(x) = \delta_0(x)$. In consequence, the variables in Y are the only variables the domains of which may be reduced in the course of $P_0 \rightarrow_{\mathrm{FD}} P_2$. Taking into account that some reduction has to take place in the course of $P_0 \rightarrow_{\mathrm{FD}} P_2$, we end up with

$$\delta_2(y_1) \times \ldots \times \delta_2(y_n) \subset \delta_0(y_1) \times \ldots \times \delta_0(y_n).$$

   By a similar argument,

$$\delta_1(y_1) \times \ldots \times \delta_1(y_n) \subset \delta_2(y_1) \times \ldots \times \delta_2(y_n).$$

2. Let $Z = X \setminus Y$. We start by noting that $\delta_2(x) \subseteq \delta_0(x) = \delta_1(x)$ for all $x \in Z$. Now suppose

$$\delta_2(y_1) \times \ldots \times \delta_2(y_n) \subseteq \delta_1(y_1) \times \ldots \times \delta_1(y_n).$$

   Then either $\delta_2(y_i) = \delta_1(y_i)$ for all $1 \leq i \leq n$ or $1 \leq i \leq n$ exists s.t. $\delta_2(y_i) \subset \delta_1(y_i)$.

   Suppose $\delta_2(y_i) = \delta_1(y_i)$ for all $1 \leq i \leq n$. If $\delta_2(x) = \delta_1(x)$ for all $x \in Z$, then $P_2 = P_1$. If $x \in Z$ exists s.t. $\delta_2(x) \subset \delta_1(x)$, then $P_1 \rightarrow_{\mathrm{FD}} P_2$.

   Suppose $1 \leq i \leq n$ exists s.t. $\delta_2(y_i) \subset \delta_1(y_i)$. Then $P_1 \rightarrow_{\mathrm{FD}} P_2$ because $\delta_2(x) \subseteq \delta_1(x)$ for all $x \in Z$.

3. By the definitions of correctness and $\rightarrow_{\mathrm{C}}$, $P_1 \not\rightarrow_{\mathrm{C}} P_2$ iff $P_1 \not\rightarrow_{\mathrm{FD}} P_2$ or $\mathrm{sol}(P_1) \neq \mathrm{sol}(P_2)$. However, $\mathrm{sol}(P_1) = \mathrm{sol}(P_0) = \mathrm{sol}(P_2)$ because $P_1 \leftarrow_{\mathrm{C}} P_0 \rightarrow_{\mathrm{C}} P_2$ and $\rightarrow_{\mathrm{C}}$ is correct. Hence $P_1 \not\rightarrow_{\mathrm{FD}} P_2$. By a symmetric argument, $P_2 \not\rightarrow_{\mathrm{FD}} P_1$. Thus, by (2),

$$\delta_2(y_1) \times \ldots \times \delta_2(y_n) \not\subseteq \delta_1(y_1) \times \ldots \times \delta_1(y_n).$$

□

Lemma A.3. *If* $P_0 \rightarrow_{\mathrm{C}} P_1$ *and* $P_0 \rightarrow^{=}_{\mathrm{FD}} P_2 \rightarrow^{=}_{\mathrm{FD}} P_1$, *then* $\mathrm{sol}(P_0) = \mathrm{sol}(P_2) = \mathrm{sol}(P_1)$.

Proof. By Corollary 1.1, $\mathrm{sol}(P_0) \subseteq \mathrm{sol}(P_2) \subseteq \mathrm{sol}(P_1)$ and, by the correctness of $\rightarrow_{\mathrm{C}}$, $\mathrm{sol}(P_0) = \mathrm{sol}(P_1)$. □

Notation A.1. If $P = (X, \delta, C)$ is a FCSP with $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, and $t = (\mathtt{S}, \mathtt{P}) \in T$, we write $\delta(t)$ instead of $\delta(\mathtt{S}) \times \delta(\mathtt{P})$.

Proposition A.3. *If* $P_0 \rightarrow_{\mathrm{PVS}} P_1$ *and* $P_0 \rightarrow_{\mathrm{FD}} P_2 \rightarrow_{\mathrm{FD}} P_1$, *then* $P_2 \rightarrow_{\mathrm{PVS}} P_1$.

Proof. Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs that satisfy the premise. Let $\texttt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, $t = (\texttt{S}, \texttt{P}) \in T$, and $a \in \mathrm{vs}(t, \delta_0)$ s.t. $a \notin \mathrm{vs}(\mathcal{T}, \delta_0)$ and $\delta_1 = \delta_0$ except for

$$\delta_1(\texttt{S}) = \{s \in \delta_0(\texttt{S}) : \exists p \in \delta_0(\texttt{P}).\ a \notin [s, s+p-1]\}$$

and

$$\delta_1(\texttt{P}) = \{p \in \delta_0(\texttt{P}) : \exists s \in \delta_0(\texttt{S}).\ a \notin [s, s+p-1]\}.$$

By Lemma A.2, $\delta_1(t) \subset \delta_2(t) \subset \delta_0(t)$. Let $\Delta = \delta_2(t) \setminus \delta_1(t)$. We observe that $\emptyset \neq \Delta \subset \delta_0(t) \setminus \delta_1(t)$. It follows that

$$a \in \bigcup_{(s,p) \in \Delta} [s, s+p-1] \subseteq \bigcup_{(s,p) \in \delta_2(t)} [s, s+p-1] = \mathrm{vs}(t, \delta_2).$$

Furthermore, by Lemma 3.1, $a \notin \mathrm{vs}(\mathcal{T}, \delta_2)$. We conclude that $P_2 \rightarrow_{\mathrm{PVS}} P_1$. $\square$

Proposition A.4. *If $P_0 \rightarrow_{\mathrm{PVS}} P_1$, $P_0 \rightarrow_{\mathrm{FD}} P_2$, $P_1 \not\rightarrow_{\mathrm{FD}} P_2$, $P_2 \not\rightarrow_{\mathrm{FD}} P_1$, and $P_1 \neq P_2$, then a FCSP P_3 exists s.t. $P_1 \rightarrow_{\mathrm{FD}} P_3$ and $P_2 \rightarrow_{\mathrm{PVS}} P_3$.*

Proof. Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs that satisfy the premise. Let $\texttt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, $t = (\texttt{S}, \texttt{P}) \in T$, and $a \in \mathrm{vs}(t, \delta_0)$ s.t. $a \notin \mathrm{vs}(\mathcal{T}, \delta_0)$ and $\delta_1 = \delta_0$ except for

$$\delta_1(\texttt{S}) = \{s \in \delta_0(\texttt{S}) : \exists p \in \delta_0(\texttt{P}).\ a \notin [s, s+p-1]\}$$

and

$$\delta_1(\texttt{P}) = \{p \in \delta_0(\texttt{P}) : \exists s \in \delta_0(\texttt{S}).\ a \notin [s, s+p-1]\}.$$

Let $P_3 = (X, \delta_3, C)$ with $\delta_3 = \delta_2$ except for

$$\delta_3(\texttt{S}) = \{s \in \delta_2(\texttt{S}) : \exists p \in \delta_2(\texttt{P}).\ a \notin [s, s+p-1]\}$$

and

$$\delta_3(\texttt{P}) = \{p \in \delta_2(\texttt{P}) : \exists s \in \delta_2(\texttt{S}).\ a \notin [s, s+p-1]\}.$$

$P_2 \rightarrow_{\mathrm{PVS}} P_3$: By Lemma A.2, $\delta_2(t) \not\subseteq \delta_1(t)$, or equivalently, $(s,p) \in \delta_0(t)$ exists s.t. $(s,p) \in \delta_2(t)$ and $(s,p) \notin \delta_1(t)$. If $s \notin \delta_1(\texttt{S})$, then $a \in [s, s+p-1]$ and thus $s \notin \delta_3(\texttt{S})$. If $p \notin \delta_1(\texttt{P})$, then $a \in [s, s+p-1]$ and thus $p \notin \delta_3(\texttt{P})$. In either case, $a \in \mathrm{vs}(t, \delta_2)$ and $P_2 \rightarrow_{\mathrm{FD}} P_3$. Moreover, by Lemma 3.1, $a \notin \mathrm{vs}(\mathcal{T}, \delta_2)$. It follows that $P_2 \rightarrow_{\mathrm{PVS}} P_3$.

$P_1 \rightarrow_{\mathrm{FD}} P_3$: $P_3 \neq P_1$ because otherwise $P_2 \rightarrow_{\mathrm{FD}} P_1$. For all $x \in X \setminus \{\texttt{S}, \texttt{P}\}$, $\delta_1(x) = \delta_0(x) \supseteq \delta_2(x) = \delta_3(x)$ because $P_1 \leftarrow_{\mathrm{PVS}} P_0 \rightarrow_{\mathrm{FD}} P_2 \rightarrow_{\mathrm{PVS}} P_3$. Suppose $\delta_3(t) \not\subseteq \delta_1(t)$, or equivalently, $(s,p) \in \delta_0(t)$ exists s.t. $(s,p) \in \delta_3(t)$ and $(s,p) \notin \delta_1(t)$. If $s \notin \delta_1(\texttt{S})$, then $a \in [s, s+p-1]$ and thus $s \notin \delta_3(\texttt{S})$. If $p \notin \delta_1(\texttt{P})$, then $a \in [s, s+p-1]$ and thus $p \notin \delta_3(\texttt{P})$. In either case, $(s,p) \notin \delta_3(t)$. $\square$

Corollary A.4. $\rightarrow_{\text{PVS}}$ *is insensitive to* $\rightarrow_{\text{FD}}$.

Proposition A.5. *If* $P_0 \rightarrow_{\text{FC}} P_1$ *and* $P_0 \rightarrow_{\text{C}} P_2 \rightarrow_{\text{C}} P_1$*, then* $P_2 \rightarrow_{\text{FC}} P_1$.

Proof. Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs that satisfy the premise. Let $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, $t = (\mathtt{S}, \mathtt{P}) \in T$, and $a \in \text{vc}(\mathcal{T}, \delta_0)$ s.t. $a \notin \text{vc}(T, \delta_0)$, $a \in \text{vs}(t, \delta_0)$, $a \notin \text{vs}(u, \delta_0)$ for all $u \in T$, $u \neq t$, and $\delta_1 = \delta_0$ except for

$$\delta_1(\mathtt{S}) = \{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\ a \in [s, s+p-1]\}$$

and

$$\delta_1(\mathtt{P}) = \{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\ a \in [s, s+p-1]\}.$$

We observe that $a \in \text{vs}(t, \delta_1)$. Hence $a \in \text{vs}(t, \delta_2)$ because $\text{vs}(t, \delta_1) \subseteq \text{vs}(t, \delta_2)$ by Lemma 3.1.

By Lemma A.2, $\delta_1(t) \subset \delta_2(t) \subset \delta_0(t)$. Let $\Delta = \delta_2(t) \setminus \delta_1(t)$. We observe that $\emptyset \neq \Delta \subset \delta_0(t) \setminus \delta_1(t)$. It follows that $(s, p) \in \Delta$ exists with $a \notin [s, s+p-1]$ and thus

$$a \notin \bigcap_{(s,p) \in \Delta} [s, s+p-1] \supseteq \bigcap_{(s,p) \in \delta_2(t)} [s, s+p-1] = \text{vc}(t, \delta_2).$$

Let $u \in T$ with $u \neq t$. By Lemma 3.1, $a \notin \text{vs}(u, \delta_2)$. By Corollary 3.1, $\text{vc}(u, \delta_2) \subseteq \text{vs}(u, \delta_2)$ and hence $a \notin \text{vc}(u, \delta_2)$. With $a \notin \text{vc}(t, \delta_2)$ we conclude that $a \notin \text{vc}(T, \delta_2)$.

Finally, by Lemma 3.1, $a \in \text{vc}(\mathcal{T}, \delta_2)$ and thus $P_2 \rightarrow_{\text{FC}} P_1$. ☐

Proposition A.6. *If* $P_0 \rightarrow_{\text{FC}} P_1$*,* $P_0 \rightarrow_{\text{C}} P_2$*,* $P_1 \not\rightarrow_{\text{C}} P_2$*,* $P_2 \not\rightarrow_{\text{C}} P_1$*, and* $P_1 \neq P_2$*, then a FCSP* P_3 *exists s.t.* $P_1 \rightarrow_{\text{C}} P_3$ *and* $P_2 \rightarrow_{\text{FC}} P_3$.

Proof. Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs that satisfy the premise. Let $\mathtt{tpp}(\mathcal{T}) \in C$, $T \in \mathcal{T}$, $t = (\mathtt{S}, \mathtt{P}) \in T$, and $a \in \text{vc}(\mathcal{T}, \delta_0)$ s.t. $a \notin \text{vc}(T, \delta_0)$, $a \in \text{vs}(t, \delta_0)$, $a \notin \text{vs}(u, \delta_0)$ for all $u \in T$, $u \neq t$, and $\delta_1 = \delta_0$ except for

$$\delta_1(\mathtt{S}) = \{s \in \delta_0(\mathtt{S}) : \exists p \in \delta_0(\mathtt{P}).\ a \in [s, s+p-1]\}$$

and

$$\delta_1(\mathtt{P}) = \{p \in \delta_0(\mathtt{P}) : \exists s \in \delta_0(\mathtt{S}).\ a \in [s, s+p-1]\}.$$

Let $P_3 = (X, \delta_3, C)$ with $\delta_3 = \delta_2$ except for

$$\delta_3(\mathtt{S}) = \{s \in \delta_2(\mathtt{S}) : \exists p \in \delta_2(\mathtt{P}).\ a \in [s, s+p-1]\}$$

and

$$\delta_3(\mathtt{P}) = \{p \in \delta_2(\mathtt{P}) : \exists s \in \delta_2(\mathtt{S}).\ a \in [s, s+p-1]\}.$$

$a \notin \text{vc}(t, \delta_2)$ and $P_2 \rightarrow_{\text{FD}} P_3$: By Lemma A.2, $\delta_2(t) \not\subseteq \delta_1(t)$, or equivalently, $(s, p) \in \delta_0(t)$ exists s.t. $(s, p) \in \delta_2(t)$ and $(s, p) \notin \delta_1(t)$. If $s \notin \delta_1(\mathtt{S})$, then $a \notin$

$[s, s+p-1]$ and thus $s \notin \delta_3(\mathtt{S})$. If $p \notin \delta_1(\mathtt{P})$, then $a \notin [s, s+p-1]$ and thus $p \notin \delta_3(\mathtt{P})$. In either case, $a \notin \mathrm{vc}(t, \delta_2)$ and $P_2 \rightarrow_{\mathrm{FD}} P_3$.

$a \notin \mathrm{vc}(T, \delta_2)$: Let $u \in T$ with $u \neq t$. By Lemma 3.1, $a \notin \mathrm{vs}(u, \delta_2)$. By Corollary 3.1, $\mathrm{vc}(u, \delta_2) \subseteq \mathrm{vs}(u, \delta_2)$ and hence $a \notin \mathrm{vc}(u, \delta_2)$. With $a \notin \mathrm{vc}(t, \delta_2)$ we conclude that $a \notin \mathrm{vc}(T, \delta_2)$.

$a \in \mathrm{vs}(t, \delta_2)$: To show that $a \in \mathrm{vs}(t, \delta_2)$, let $\sigma \in \mathrm{sol}(P_0)$. By Lemma 3.1, $a \in \mathrm{vc}(\mathcal{T}, \sigma)$. By Lemma 3.4, $\mathrm{vc}(T, \sigma) = \mathrm{vc}(\mathcal{T}, \sigma)$. Suppose $a \notin \mathrm{vs}(t, \sigma)$. By Lemma 3.4, $a \notin \mathrm{vc}(t, \sigma)$. Moreover, for $u \in T$ with $u \neq t$, we have $a \notin \mathrm{vs}(u, \delta_0) \supseteq \mathrm{vs}(u, \sigma) = \mathrm{vc}(u, \sigma)$ by Lemma 3.1 and by Lemma 3.4. Now, because no task in T can cover a, $a \notin \mathrm{vc}(T, \sigma)$. Putting it all together, we obtain the contradiction $a \notin \mathrm{vc}(T, \sigma) = \mathrm{vc}(\mathcal{T}, \sigma) \ni a$. $\mathrm{sol}(P_0) = \mathrm{sol}(P_2)$ because $P_0 \rightarrow_{\mathrm{C}} P_2$ and $\rightarrow_{\mathrm{C}}$ is correct. Hence $\sigma \in \mathrm{sol}(P_2)$ and thus $P_2 \rightarrow_{\mathrm{FD}} (X, \sigma, C)$. By Lemma 3.1, it follows that $a \in \mathrm{vs}(t, \sigma) \subseteq \mathrm{vs}(t, \delta_2)$.

$P_2 \rightarrow_{\mathrm{FC}} P_3$: It remains to show that $a \in \mathrm{vc}(\mathcal{T}, \delta_2)$ and that $a \notin \mathrm{vs}(u, \delta_2)$ for all $u \in T$, $u \neq t$. Both facts follow from Lemma 3.1.

$P_1 \rightarrow_{\mathrm{FD}} P_3$: $P_3 \neq P_1$ because otherwise $P_2 \rightarrow_{\mathrm{FD}} P_1$. For all $x \in X \setminus \{\mathtt{S}, \mathtt{P}\}$, $\delta_1(x) = \delta_0(x) \supseteq \delta_2(x) = \delta_3(x)$ because $P_1 \leftarrow_{\mathrm{FC}} P_0 \rightarrow_{\mathrm{FD}} P_2 \rightarrow_{\mathrm{FC}} P_3$. Suppose $\delta_3(t) \not\subseteq \delta_1(t)$, or equivalently, $(s,p) \in \delta_0(t)$ exists s.t. $(s,p) \in \delta_3(t)$ and $(s,p) \notin \delta_1(t)$. If $s \notin \delta_1(\mathtt{S})$, then $a \notin [s, s+p-1]$ and thus $s \notin \delta_3(\mathtt{S})$. If $p \notin \delta_1(\mathtt{P})$, then $a \notin [s, s+p-1]$ and thus $p \notin \delta_3(\mathtt{P})$. In either case, $(s,p) \notin \delta_3(t)$.

$P_1 \rightarrow_{\mathrm{C}} P_3$: $\mathrm{sol}(P_1) = \mathrm{sol}(P_0) = \mathrm{sol}(P_2) = \mathrm{sol}(P_3)$ because $P_1 \leftarrow_{\mathrm{FC}} P_0 \rightarrow_{\mathrm{C}} P_2 \rightarrow_{\mathrm{FC}} P_3$ and both $\rightarrow_{\mathrm{FC}}$ and $\rightarrow_{\mathrm{C}}$ are correct. $\square$

Corollary A.5. *$\rightarrow_{\mathrm{FC}}$ is insensitive to $\rightarrow_{\mathrm{C}}$.*

Proposition A.7. *If $P_0 \rightarrow_{\mathrm{IPT}} P_1$ and $P_0 \rightarrow_{\mathrm{FD}} P_2 \rightarrow_{\mathrm{FD}} P_1$, then $P_2 \rightarrow_{\mathrm{IPT}} P_1$.*

Proof. Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs that satisfy the premise. Let $\mathtt{tpp}(\mathcal{T}) \in C$, $T_0, T_1 \in \mathcal{T}$, and $l, u \geq 0$ s.t., for all $(X, \gamma, C) \in \mathrm{gs}(P_0)$, l is a lower bound on $|\mathrm{vc}(T_0, \gamma)|$, u is an upper bound on $|\mathrm{vc}(T_1, \gamma)|$, and $u < l$. Let $(X, \gamma, C) \in \mathrm{gs}(P_2)$. By Corollary 3.2, $\mathrm{gs}(P_2) \subseteq \mathrm{gs}(P_0)$ and thus l is a lower bound on $|\mathrm{vc}(T_0, \gamma)|$, u is an upper bound on $|\mathrm{vc}(T_1, \gamma)|$, and $u < l$. Furthermore, P_1 is failed because $P_0 \rightarrow_{\mathrm{IPT}} P_1$. We conclude that $P_2 \rightarrow_{\mathrm{IPT}} P_1$. $\square$

Proposition A.8. *If $P_0 \rightarrow_{\mathrm{IPT}} P_1$, $P_0 \rightarrow_{\mathrm{FD}} P_2$, $P_1 \not\rightarrow_{\mathrm{FD}} P_2$, $P_2 \not\rightarrow_{\mathrm{FD}} P_1$, and $P_1 \neq P_2$, then a FCSP P_3 exists s.t. $P_1 \rightarrow_{\mathrm{FD}} P_3$ and $P_2 \rightarrow_{\mathrm{IPT}} P_3$.*

Proof. Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs that satisfy the premise. Let $\mathtt{tpp}(\mathcal{T}) \in C$, $T_0, T_1 \in \mathcal{T}$, and $l, u \geq 0$ s.t., for all $(X, \gamma, C) \in \mathrm{gs}(P_0)$, l is a lower bound on $|\mathrm{vc}(T_0, \gamma)|$, u is an upper bound on $|\mathrm{vc}(T_1, \gamma)|$, and $u < l$. Let $P_3 = (X, \delta_3, C)$ with $\delta_3(x) = \delta_1(x) \cap \delta_2(x)$ for all $x \in X$. P_3 is failed because P_1 is failed.

Suppose $P_1 \not\rightarrow_{\text{FD}} P_3$. Then either $\delta_1 = \delta_3$ or $x \in X$ exists s.t. $\delta_3(x) \not\subseteq \delta_1(x)$. The latter case contradicts the construction of P_3. If $\delta_1 = \delta_3$, then $\delta_1(x) \subseteq \delta_2(x)$ for all $x \in X$ and thus either $P_1 = P_2$ or $P_2 \rightarrow_{\text{FD}} P_1$.

Suppose $P_2 \not\rightarrow_{\text{FD}} P_3$. Then either $\delta_2 = \delta_3$ and thus $P_2 = P_3$ or $x \in X$ exists s.t. $\delta_3(x) \not\subseteq \delta_2(x)$. The former case contradicts $P_1 \not\rightarrow_{\text{FD}} P_2$, the latter case contradicts the construction of P_3.

Finally, let $(X, \gamma, C) \in \text{gs}(P_2)$. By Corollary 3.2, $\text{gs}(P_2) \subseteq \text{gs}(P_0)$ and thus l is a lower bound on $|\text{vc}(T_0, \gamma)|$, u is an upper bound on $|\text{vc}(T_1, \gamma)|$, and $u < l$.

We conclude that $P_2 \rightarrow_{\text{IPT}} P_3$. □

Corollary A.6. $\rightarrow_{\text{IPT}}$ *is insensitive to* $\rightarrow_{\text{FD}}$.

Proposition A.9. *If* $P_0 \rightarrow_{\text{NC}} P_1$ *and* $P_0 \rightarrow_{\text{FD}} P_2 \rightarrow_{\text{FD}} P_1$*, then* $P_2 \rightarrow_{\text{NC}} P_1$.

Proof. Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs that satisfy the premise. Let $\texttt{tpp}(\mathcal{T}) \in C$ and $T \in \mathcal{T}$ s.t. $G_0 = (U_0, V_0, E_0) = \text{vcg}(\mathcal{T}, T, \delta_0)$ does not have a matching M_0 with $|M_0| = |V_0|$. Suppose $G_2 = (U_2, V_2, E_2) = \text{vcg}(\mathcal{T}, T, \delta_2)$ has a matching M_2 with $|M_2| = |V_2|$. Let $M_0 = \left\{(u_i^j, a) \in M_2 : a \in V_0\right\}$. We will show that M_0 is a matching in G_0 and that $|M_0| = |V_0|$.

1. M_0 is a matching because $M_0 \subseteq M_2$.
2. $M_0 \subseteq E_0$: Let $(u_i^j, a) \in M_0$. We know that $u_i^j \in U_2$, $a \in V_0$, and that $s \in \delta_2(\texttt{S}_i)$ exists s.t. $s + j = a$. Let $s \in \delta_2(\texttt{S}_i)$ s.t. $s + j = a$. $u_i^j \in U_0$ and $s \in \delta_0(\texttt{S}_i)$ because $P_0 \rightarrow_{\text{FD}} P_2$.
3. $|M_0| = |V_0|$ because, for all $a \in V_2$, a is matched by M_2 and, by Lemma 3.1, $V_0 = \text{vc}(\mathcal{T}, \delta_0) \subseteq \text{vc}(\mathcal{T}, \delta_2) = V_2$.

Furthermore, P_1 is failed because $P_0 \rightarrow_{\text{NC}} P_1$. We conclude that $P_2 \rightarrow_{\text{NC}} P_1$. □

Proposition A.10. *If* $P_0 \rightarrow_{\text{NC}} P_1$*,* $P_0 \rightarrow_{\text{FD}} P_2$*,* $P_1 \not\rightarrow_{\text{FD}} P_2$*,* $P_2 \not\rightarrow_{\text{FD}} P_1$*, and* $P_1 \neq P_2$*, then a FCSP* P_3 *exists s.t.* $P_1 \rightarrow_{\text{FD}} P_3$ *and* $P_2 \rightarrow_{\text{NC}} P_3$.

Proof. Let $P_0 = (X, \delta_0, C)$, $P_1 = (X, \delta_1, C)$, and $P_2 = (X, \delta_2, C)$ be FCSPs that satisfy the premise. We let our proof follow the pattern that we used to prove Proposition A.8: We choose a constraint $\texttt{tpp}(\mathcal{T}) \in C$ and a track $T \in \mathcal{T}$ s.t. the graph $(U_0, V_0, E_0) = \text{vcg}(\mathcal{T}, T, \delta_0)$ does not have a matching M_0 with $|M_0| = |V_0|$. Then, by reference to the arguments employed in the proof of Proposition A.9, we show that the graph $(U_2, V_2, E_2) = \text{vcg}(\mathcal{T}, T, \delta_2)$ does not have a matching M_2 with $|M_2| = |V_2|$. □

Corollary A.7. $\rightarrow_{\text{NC}}$ *is insensitive to* $\rightarrow_{\text{FD}}$.

Corollary A.8. $\rightarrow_{\mathrm{PVS}}$, $\rightarrow_{\mathrm{IPT}}$, *and* $\rightarrow_{\mathrm{NC}}$ *are insensitive to* $\rightarrow_{\mathrm{C}}$.

Corollary A.9. *If* $R \subseteq \{\rightarrow_{\mathrm{PVS}}, \rightarrow_{\mathrm{FC}}, \rightarrow_{\mathrm{IPT}}, \rightarrow_{\mathrm{NC}}\}$ *and* $\rightarrow_R = \bigcup R$*, then* $\rightarrow_R$ *is terminating, insensitive to* $\rightarrow_{\mathrm{C}}$*, locally confluent, and confluent.*

A.4 Related Work

A.4.1 The Commutative Union Lemma

The Commutative Union Lemma [BN98] (CUL) is a well-known tool to prove confluence. It states that the union of two reductions is confluent if both reductions are confluent and commute. In the following, we compare our method to the method suggested by the CUL. We are interested in how to proceed in different situations: when proving the confluence of a reduction from scratch and when proving the confluence of a reduction that has been obtained by extending or reducing a confluent reduction. Note that our statements wrt. the number of proof obligations arising on top-level are not meant to say anything about the complexity and the difficulties of the proofs that are actually required.

Let $(A, \bigcup_{1 \leq i \leq n} \rightarrow_i)$ be a reduction system. To show that $\bigcup_{1 \leq i \leq n} \rightarrow_i$ is confluent, the CUL suggests to show that each $\rightarrow_i$, $1 \leq i \leq n$, is confluent and that, for each $1 < i \leq n$, $\rightarrow_i$ and $\bigcup_{1 \leq k < i} \rightarrow_k$ commute. Hence n confluence proofs and $n-1$ proofs of commutation are required. Suppose we are showing that $\rightarrow_i$, $1 < i \leq n$, and $\bigcup_{1 \leq k < i} \rightarrow_k$ commute. In general, this proof requires to consider $i-1$ cases, one for each $\rightarrow_k$, $1 \leq k < i$, except for it is possible to characterize $\bigcup_{1 \leq k < i} \rightarrow_k$ in a way that facilitates to deal with all cases in one sweep. In summary, at least $2n+1 \in O(n)$ and at most $n + \frac{(n-1)(n-2)}{2} \in O(n^2)$ proof obligations arise. Our approach requires n proofs of insensitivity to $\bigcup_{1 \leq i \leq n} \rightarrow_i$, one for each $\rightarrow_i$, $1 \leq i \leq n$. In general, each proof of insensitivity has to consider n cases, one for each $\rightarrow_i$, $1 \leq i \leq n$, except for it is possible to characterize $\bigcup_{1 \leq i \leq n} \rightarrow_i$ or some superset relation in a way that facilitates to deal with all cases in one sweep. In summary, at least n and at most n^2 proof obligations arise. In our application to finite-domain constraint solving, we proved insensitivity to either $\rightarrow_{\mathrm{FD}}$ or $\rightarrow_{\mathrm{C}}$. $\rightarrow_{\mathrm{FD}}$ allows for arbitrary domain reductions while $\rightarrow_{\mathrm{C}} \subseteq \rightarrow_{\mathrm{FD}}$ only allows for domain reductions that preserve solutions. This way it was possible to prove the confluence of the TPP solver with a number of proofs linear in the number of reductions.

Let $\rightarrow_{n+1} \in A \times A$. Suppose $\bigcup_{1 \leq i \leq n} \rightarrow_i$ is known to be confluent. To show that $\bigcup_{1 \leq i \leq n+1} \rightarrow_i$ is confluent, the CUL obliges to show that $\rightarrow_{n+1}$ is confluent and that $\rightarrow_{n+1}$ and $\bigcup_{1 \leq i \leq n} \rightarrow_i$ commute. As explained above, the proof of commutation may require to distinguish up to n cases. In the worst case, our approach obliges to show that, for all $1 \leq i \leq n+1$, $\rightarrow_i$ is insensitive to $\bigcup_{1 \leq i \leq n+1} \rightarrow_i$. This amounts to proving confluence from scratch. However, if we can characterize a superset relation that contains all reductions we might wish to investigate, it is possible to reuse all prior proofs of insensitivity: Suppose we know that $\bigcup_{1 \leq i \leq n} \rightarrow_i$ is insensitive to the superset relation, then it is sufficient to show that $\rightarrow_{n+1}$ is insensitive to the superset relation. In our application to finite-domain constraint solving, we used $\rightarrow_{\mathrm{FD}}$ and $\rightarrow_{\mathrm{C}}$ as superset relations. If we wished to extend the TPP solver, it was sufficient to show that the extension is insensitive to either $\rightarrow_{\mathrm{FD}}$ or $\rightarrow_{\mathrm{C}}$.

Suppose $\bigcup_{1 \leq i \leq n} \rightarrow_i$ is known to be confluent. Let $1 \leq k \leq n$. Is $\bigcup_{1 \leq i \leq n, i \neq k} \rightarrow_i$ confluent? If the CUL has been applied, then, for each $k < l \leq n$, it is necessary to prove that $\rightarrow_l$ and $\bigcup_{1 \leq i < l, i \neq k}$ commute. If our approach has been used, no proof obligations arise, independent of whether insensitivity has been proved to $\bigcup_{1 \leq i \leq n} \rightarrow_i$ or to some superset relation.

Suppose $\bigcup_{1 \leq i \leq n} \rightarrow_i$ is known to be confluent. Let $I \subset [1, \ldots, n]$. Is $\bigcup_{i \in I} \rightarrow_i$ confluent? The answers to this question are similar to those to the previous question. In particular, if our approach has been used to prove the confluence of $\bigcup_{1 \leq i \leq n} \rightarrow_i$, no proof obligations arise.

A.4.2 The Critical-Pair Approach

Critical pairs are employed in the static analysis of term-rewriting systems [BN98] and rule-based constraint solvers [AFM99]. Critical pairs are defined in a domain-specific way; basically, a critical pair is a pair of states obtained from the parallel application of two interfering rewrite rules to a common and minimal ancestor state. The importance of critical pairs is due to their relationship to local confluence. For example, a term-rewriting system is locally confluent iff all its critical pairs are joinable. However, the critical-pair approach is not modular. Suppose we are given a term-rewriting system the critical pairs of which are all joinable. After adding a rule, the approach requires to check all the critical pairs the new rule has with itself and with the other rules. After removing a rule, the approach requires to check all proofs of joinability; any proof that is based on the removed rule has be redone.

A.5 Conclusion

We developed a sufficient condition for local confluence and demonstrated that it is well suited to study the behavior of cooperating constraint solvers.

We call our approach modular because removing reductions does not affect confluence and because proving confluence after adding a reduction requires only a single proof that, if a superset relation is used, does not require to reconsider any other reduction. Our approach is suitable for the study of constraint solvers because superset relations are available that comprise all conceivable solvers.

Appendix B

Bibliography

[AAA96] *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*. AAAI Press, 1996.

[AB93] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[AFM99] Slim Abdennadher, Thom Frühwirth, and Holger Meuss. Confluence and semantics of constraint simplification rules. *Constraints*, 4(2):133–165, 1999.

[AM00] Slim Abdennadher and Michael Marte. University course timetabling using Constraint Handling Rules. *Journal of Applied Artificial Intelligence*, 14(4):311–326, 2000.

[App98] Dieter Appelt. *Education in Bavaria*. Bavarian State Ministry of Education and Cultural Affairs, 1998.

[Bap98] Philippe Baptiste. *A theoretical and experimental study of resource constraint propagation*. PhD thesis, Université de Technologie de Compiègne, 1998.

[Bar96] Victor A. Bardadym. Computer-aided school and university timetabling: The new wave. In Burke and Ross [BR96b], pages 22–45.

[BC01] Nicolas Beldiceanu and Mats Carlsson. Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint. In Toby Walsh, editor, *Seventh International Conference on Principles and Practice of Constraint Programming*, LNCS 2239, pages 377–391. Springer, 2001.

[BE01] Edmund Burke and Wilhelm Erben, editors. *Practice and Theory of Automated Timetabling III*. LNCS 2079. Springer, 2001.

[BK01] Peter Brucker and Sigrid Knust. Resource-constrained project scheduling and timetabling. In Burke and Erben [BE01], pages 277–293.

[BN94] P. Brucker and L. Nordmann. The k-track assignment problem. *Computing*, 52(2):97–122, 1994.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[BP97] Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. In Smolka [Smo97], pages 375–389.

[BPN99] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92:305–333, 1999.

[BR96a] Christian Bessière and Jean-Charles Régin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In Freuder [Fre96], pages 61–75.

[BR96b] Edmund Burke and Peter Ross, editors. *Practice and Theory of Automated Timetabling*, LNCS 1153. Springer, 1996.

[BR99] Christian Bessière and Jean-Charles Régin. Enforcing arc consistency on global constraints by solving subproblems on the fly. In Jaffar [Jaf99], pages 103–117.

[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.

[CDM98] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Metaheuristics for high-school timetabling. *Computational Optimization and Applications*, 9(3):277–298, 1998.

[CL98] Michael W. Carter and Gilbert Laporte. Recent developments in practical course timetabling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, LNCS 1408, pages 3–19. Springer, 1998.

[CLW96] B. M. W. Cheng, J. H. M. Lee, and J. C. K. Wu. Speeding up constraint propagation by redundant modeling. In Freuder [Fre96], pages 91–103.

[COC97] Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In *Ninth International Symposium on Programming Languages, Implementations, Logics, and Programs*, LNCS 1292, pages 191–206. Springer, 1997.

[Coh95] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[Con02] Condor Team, University of Wisconsin-Madison. *Condor Version 6.3.3 Manual*, 2002. Available from `http://www.cs.wisc.edu/condor`.

[Cos94] Daniel Costa. A tabu search algorithm for computing an operational timetable. *European Journal of Operational Research*, 76(1):98–110, 1994.

[CP01] Marco P. Carrasco and Margarida V. Pato. A multiobjective genetic algorithm for the class/teacher timetabling problem. In Burke and Erben [BE01], pages 3–17.

[DPPH00] Ulrich Dorndorf, Erwin Pesch, and Toàn Phan-Huy. Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122(1–2):189–240, 2000.

[DS97] Andreas Drexl and Frank Salewski. Distribution requirements and compactness constraints in school timetabling. *European Journal of Operational Research*, 102(1):193–214, 1997.

[dW85] Dominique de Werra. An introduction to timetabling. *European Journal of Operations Research*, 19:151–162, 1985.

[EIS76] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

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

[FCMR99] Carlos Fernandes, Joao Paulo Caldeira, Fernando Melicio, and Agostinho Rosa. High-school weekly timetabling by evolutionary algorithms. In *ACM Symposium on Applied Computing*, pages 344–350, 1999.

[FKN99] U. Faigle, W. Kern, and W. M. Nawijn. A greedy on-line algorithm for the *k*-track assignment problem. *Journal of Algorithms*, 31(1):196–210, 1999.

[FLM99] F. Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In Jaffar [Jaf99], pages 189–203.

[Fre96] Eugene C. Freuder, editor. *Second International Conference on Principles and Practice of Constraint Programming*, LNCS 1118. Springer, 1996.

[Frü98] Thom Frühwirth. Theory and practice of constraint handling rules, special issue on constraint logic programming. *Journal of Logic Programming*, pages 95–138, 1998.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GJBP96] Christelle Gueret, Narendra Jussien, Patrice Boizumault, and Christian Prins. Building university timetables using constraint logic programming. In Burke and Ross [BR96b], pages 130–145.

[GM99] Hans-Joachim Goltz and Dirk Matzke. University timetabling using constraint logic programming. In *Practical Aspects of Declarative Languages*, LNCS 1551, pages 320–334. Springer, 1999.

[HW96] Martin Henz and Jörg Würtz. Using Oz for college timetabling. In Burke and Ross [BR96b], pages 283–296.

[Int02] Intelligent Systems Laboratory, Swedish Institute of Computer Science. *SICStus Prolog User's Manual, Release 3.9.0*, 2002.

[Jaf99] Joxan Jaffar, editor. *Fifth International Conference on Principles and Practice of Constraint Programming*, LNCS 1713. Springer, 1999.

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

[Jun86] Werner Junginger. Timetabling in Germany – A survey. *Interfaces*, 16(4):66–74, 1986.

[KL95] A. W. J. Kolen and J. K. Lenstra. Combinatorics in operations research. In R. L. Graham, M. Grötschel, and L. Lovácz, editors, *Handbook of Combinatorics*, volume 2. Elsevier, 1995.

[Kol97] *Kollegstufe – Die Oberstufe des Gymnasiums in Bayern*. Bavarian State Ministry of Education and Cultural Affairs, 1997.

[KYN99] Kazuya Kaneko, Masazumi Yoshikawa, and Yoichiro Nakakuki. Improving a heuristic repair method for large-scale school timetabling problems. In Jaffar [Jaf99], pages 275–288.

[Mar00] Michael Marte. Towards constraint-based grammar school timetabling. In Edmund Burke and Wilhelm Erben, editors, *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, pages 222–224, 2000. Extended abstract.

[Mar01a] Michael Marte. Basic models for school timetabling. In *Proceedings of the 5th Workshop on Models and Algorithms for Planning and Scheduling Problems*, 2001. Extended abstract.

[Mar01b] Michael Marte. A global constraint for parallelizing the execution of task sets in non-preemptive scheduling. In *Electronic Proceedings of the CP-2001 Doctoral Programme*, 2001. Available from `http://www.math.unipd.it/~frossi/doctoral.html`.

[Mar02] Michael Marte. A modular approach to proving confluence. In Alessandro Armando, editor, *Frontiers of Combining Systems, 4th International Workshop*, LNAI 2309, pages 33–48. Springer, 2002.

[MN99] Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[MS98] Kim Marriott and Peter Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.

[MT00] Kurt Mehlhorn and Sven Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In Rina Dechter, editor, *Sixth International Conference on Principles and Practice of Constraint Programming*, LNCS 1894, pages 306–319. Springer, 2000.

[Nui94] W. P. M. Nuijten. *Time and resource constrained scheduling: A constraint satisfaction approach*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1994.

[PB98] Claude Le Pape and Philippe Baptiste. Resource constraints for preemptive job-shop scheduling. *Constraints*, 3:263–287, 1998.

[PJH99] Simon Peyton Jones and John Hughes, editors. *Haskell 98: A Non-strict, Purely Functional Language*. 1999. Available from `http://www.haskell.org`.

[Pug98] Jean-François Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence and the 10th Innovative Applications of Artificial Intelligence Conference*, pages 359–366. AAAI Press, 1998.

[Rég94] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 362–367. AAAI Press, 1994.

[Rég96] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference* [AAA96], pages 209–215.

[RP97] Jean-Charles Régin and Jean-François Puget. A filtering algorithm for global sequencing constraints. In Smolka [Smo97], pages 32–46.

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

[Sch96a] Andrea Schaerf. Tabu search techniques for large high-school timetabling problems. In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference* [AAA96], pages 363–368.

[Sch96b] Andrea Schaerf. Tabu search techniques for large high-school timetabling problems. Technical Report CS-R9611, CWI - Centrum voor Wiskunde en Informatica, March 30, 1996.

[Sch99] Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

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

[SL94] Majid Sarrafzadeh and D. T. Lee. Restricted track assignment with applications. *International Journal of Computational Geometry and Applications*, 4(1):53–68, 1994.

[Smo97] Gert Smolka, editor. *Third International Conference on Principles and Practice of Constraint Programming*, LNCS 1330. Springer, 1997.

[SW99] Kostas Stergiou and Toby Walsh. The difference all-difference makes. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 414–419. Morgan Kaufmann Publishers, 1999.

[Tho99] Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley, 2 edition, 1999.

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

[vH01] W. J. van Hoeve. The alldifferent constraint: A survey. In *6th Annual Workshop of the ERCIM Working Group on Constraints*, 2001.

[VSD98] Pascal Van Hentenryck, Vijay Saraswat, and Yves Deville. Design, implementation, and evaluation of the constraint language cc(FD). *Journal of Logic Programming*, 37(2):139–164, 1998.

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

[Wil02] Roy Willemen. *School Timetable Construction – Algorithms and Complexity*. PhD thesis, Eindhoven University of Technology, 2002.

[YKYW96] Masazumi Yoshikawa, Kazuya Kaneko, Toru Yamanouchi, and Masanobu Watanabe. A constraint-based high school scheduling system. *IEEE Expert*, 11(1):63–72, 1996.

Appendix C

Index

Appendix D

Lebenslauf

Persönliche Daten

Vorname	Michael
Nachname	Marte
Geburtsdatum	29. Juli 1972
Geburtsort	Salzburg, Österreich
Nationalität	Österreich

Ausbildung

1978 – 1982	Volksschule Salzburg Nonntal
1982 – 1985	Bundesrealgymnasium Salzburg
1985 – 1991	Max-Planck-Gymnasium München
1991	Abitur
1991 – 1992	Studium der Phonetik und der Psycholinguistik an der Universität München
1992 – 1999	Studium der Informatik im Hauptfach an der Universität München mit den Schwerpunkten deklarative Programmierung, automatische Deduktion, und verteiltes Rechnen
1992 – 1999	Studium der Psychologie im Nebenfach an der Universität München mit den Schwerpunkten motorisches Lernen und kognitive Architekturen
1999	Studienabschluß als *Diplom-Informatiker Univ.*
1999 – 2002	Doktorand in der Informatik an der Universität München als Mitglied des Graduiertenkollegs *Sprache, Information und Logik* der Deutschen Forschungsgemeinschaft