Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 137

Optimierung von Schulstundenplänen

Philipp Keck

Studiengang:	Softwaretechnik, B.Sc.
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Prof. Dr. Stefan Funke
Beginn am:	21.05.2014
Beendet am:	10.10.2014
CR-Nummer:	I.2.8, G.2.3, F.2.2, J.1

Zusammenfassung

Die automatische Erstellung von Stundenplänen für Schulen ist seit vielen Jahren Forschungsgegenstand in den Bereichen der Künstlichen Intelligenz und der Unternehmensforschung. Diese Arbeit kombiniert Techniken aus beiden Bereichen: Eine Modellierung als Constraint Satisfaction Problem und eine Modellierung als pseudo-boolesches Optimierungsproblem werden jeweils mit der Relaxierung des zugehörigen linearen Programms kombiniert, um schneller bessere Ergebnisse zu erzielen. Für jede Variante und Kombination dieser Modellierungen wurden Tests mit vier verschiedenen Probleminstanzen durchgeführt. Die Ergebnisse zeigen, dass die vorgestellten Verfahren zumindest für kleinere Schulen, wie beispielsweise deutsche Grundschulen, ähnlich schnell sind wie etablierte Verfahren. Im Unterschied zu diesen liefern die vorgestellten Verfahren jedoch stets vollständig zulässige und beweisbar optimale Lösungen und sind zudem einfacher erweiterbar und anpassbar. In Kombination mit kommerziellen Lösern erzielen die vorgestellten Modellierungen auch für größere Probleminstanzen, wie z. B. die von deutschen Gymnasien, bessere Ergebnisse. Die Modellierungen und Ergebnisse in dieser Arbeit sind spezifisch für das deutsche Schulsystem.

Abstract

For many years, the automated construction of school timetables has been subject of research in Artificial Intelligence and Operations Research. This thesis combines techniques from both areas: Formulations as a constraint satisfaction problem and as a pseudo-boolean optimization problem are each combined with the relaxation of the corresponding linear program, in order to obtain better results more quickly. For each variant and combination of these formulations, tests were carried out using four different instances. The results show that – at least for smaller schools like German primary schools – the proposed methods are as fast as established methods. In contrast to these, the proposed methods always yield entirely feasible and provably optimal solutions. Moreover, they are easier to extend and adjust. When combined with commercial solvers, the proposed problem formulations achieve better results for larger instances like those of German high schools, as well. The formulations and results in this thesis are specific to the German education system.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Das Erstellen von brauchbaren Stundenplänen für Schulen und Universitäten stellt auch im Computer-Zeitalter noch eine Herausforderung dar. Zwar verspricht eine Vielzahl von Programmen, die Aufgabe zu übernehmen oder zu unterstützen, doch neben der Einarbeitung in die jeweilige Software ist oft eine zeitaufwändige Nachbearbeitung der berechneten Stundenpläne notwendig. Insbesondere wenn die algorithmische Stundenplanung daran scheitert, alle Unterrichtsstunden im Plan unterzubringen ohne dabei Konflikte zu erzeugen, muss der Mensch noch seine Erfahrung und Arbeitszeit einbringen, um einen verwendbaren Stundenplan zu erhalten.

Nach einer allgemeinen Definition des Stundenplanproblems für deutsche Schulen in Kapitel 2 gibt diese Arbeit in Kapitel 3 einen Überblick über mögliche Lösungsansätze. Insbesondere geht Abschnitt 3.3 auf einen Ansatz ein, der von Weidler [2012] eingeführt wurde, und der das Problem als ganzzahliges, lineares Optimierungsproblem modelliert, um es dann mit einem Löser für lineare Programmierung zu lösen. Anschließend wird in Kapitel 4 die Software „SchulScheduler“ vorgestellt, die basierend auf diesem Ansatz in einem Studienprojekt entwickelt wurde.

Ziel dieser Arbeit ist es, andere Ansätze zur Lösung des Stundenplanproblems zu untersuchen, die insbesondere nicht von dem kommerziellen Solver Gurobi abhängig sind, der von der Software „SchulScheduler“ derzeit verwendet wird. Außerdem werden Techniken eingesetzt, die es im Unterschied zu vielen bestehenden Softwareprodukten erlauben, beweisbar optimale Lösungen zu berechnen. Kapitel 5 zeigt zwei Ansätze, die mittels Constraint Programming auf unterschiedliche Art das Problem modellieren, und vergleicht die Praxistauglichkeit dieser Ansätze anhand von Implementierungen mit zwei offenen CSP-Solvern. Um die Erfolgsquote zu erhöhen, wird in Kapitel 6 die LP-Relaxierung der Probleminstanzen verwendet. In Kapitel 7 wird ein weiterer Ansatz vorgestellt, der auf pseudo-boolescher Optimierung basiert. Kapitel 8 gibt neben einer Zusammenfassung und einigen Anmerkungen vor allem einen Ausblick auf mögliche Verbesserungen der vorgestellten Ansätze.

Ganz besonders möchte ich Professor Funke für seine hilfreiche und kompetente Betreuung sowie die interessante und praxisnahe Aufgabenstellung danken. Außerdem geht ein herzliches Dankeschön an meine Tante, die mir am lebenden Beispiel einen Einblick in die Stundenplanung an Grundschulen gegeben hat, sowie an meine Kommilitonen Sven Schnaible und Gustav Murawski fürs Korrekturlesen.

Kapitel 2

Problemdefinition

In diesem Kapitel wird das Stundenplanproblem in der Form definiert, die in dieser Arbeit verwendet wird. Abschnitt 2.2 führt die verwendeten fachlichen Begriffe und abkürzende Schreibweisen ein. Nach der formalen Definition der grundlegenden Mengen in Abschnitt 2.3 werden in Abschnitt 2.4 und Abschnitt 2.5 die harten bzw. weichen Bedingungen vorgestellt.

2.1. Abgrenzung

Die in dieser Arbeit betrachtete Variante des Stundenplanproblems orientiert sich an typischen deutschen Schulen. Die Schulsysteme anderer Länder unterscheiden sich zum Teil in wesentlichen Punkten, sodass die Definition und die Lösungsansätze nicht ohne Weiteres übertragbar sind. Insbesondere werden hier keine Sonderfälle wie beispielsweise das Kurssystem an Gymnasien betrachtet.

Für die Erstellung des Stundenplans wird davon ausgegangen, dass die Zuteilung von Klassen, Fächern und Lehrern bereits stattgefunden hat. Die Zuteilung der Fächer zu den Klassen ergibt sich (in Deutschland) ohnehin direkt aus dem Bildungsplan und die Zuteilung von Klassen- und Fach-Lehrern ist eine Entscheidung, die üblicherweise vorab getroffen wird. Auch dafür existieren algorithmische Ansätze [z. B. Tillett, 1975; Breslaw, 1976], die hier aber nicht näher betrachtet werden. Bei der mathematischen Definition folgt diese Arbeit in weiten Teilen der Definition von Weidler [2012, Kap. 4], die jedoch vereinfacht wird, indem zusätzlich der Begriff des Unterrichts eingeführt wird.

2.2. Begriffe und Konventionen

Die grundlegenden Entitäten bei der Stundenplan-Erstellung sind daher **Klassen**, **Fächer** und **Lehrer**. Das Zusammentreffen einer Klasse mit einem Lehrer für ein bestimmtes Fach wird als **Unterricht** bezeichnet. Ein Unterricht nimmt eine gewisse Anzahl an Schulstunden

pro Woche in Anspruch, wodurch sich einzelne **Unterrichtsstunden** ergeben, die es im Zeitraster zu platzieren gilt. Das Zeitraster ist eine Menge von **Zeitslots**, die für jeden Tag und jede Stunde einen Eintrag enthält. Beispiel:

> Der *Deutsch*-Unterricht der Klasse *5a* wird vom Lehrer *Adam* drei Mal wöchentlich gehalten. Er findet in den folgenden Zeitslots statt: *Montag 3. Stunde*, *Montag 4. Stunde* und *Donnerstag 3. Stunde*.

Am gleichen Beispiel lassen sich die in dieser Arbeit verwendeten Konventionen für Abkürzungen verdeutlichen:

> Der Unterricht *5a-D-Ad* findet in den Zeitslots *Mo3*, *Mo4* und *Do3* statt.

2.3. Formalisierung

Mathematisch kann das Stundenplanproblem durch die folgenden jeweils endlichen Mengen dargestellt werden:

- Klassen K
- Fächer F
- Lehrer L
- Unterrichte $U \subseteq K \times F \times L$
- Wochenstunden $W : U \to \mathbb{N}$
- Wochentage $T = \{Mo, Di, Mi, Do, Fr\}$
- Stunden $S = \{1, \ldots, n\}$ mit $n \in \mathbb{N}$
- Verfügbare Zeitslots $Z \subseteq \{z_{ts} \mid t \in T, s \in S\} = T \times S$

Zu bemerken ist hier, dass es sich bei W um eine Multimenge handelt. Wenn ein Unterricht $u = (k, f, l) \in U$ zum Beispiel vier Mal in W enthalten ist (also $W(k, f, l) = 4$), dann bedeutet das, dass der Unterricht vier Mal pro Woche stattfinden soll. Z ist eine Teilmenge der gesamten möglichen Stundenmatrix $T \times S$, weil z. B. am Freitagnachmittag nicht alle Zeitslots zur Planung zur Verfügung stehen.

Das Ziel ist nun, eine Zuteilung $x : U \to \mathcal{P}(Z)$ zu finden, die für jeden Unterricht angibt, in welchen Zeitslots dieser stattfindet. Im Folgenden werden die Kriterien beschrieben, die eine solche Lösung erfüllen muss.

2.3.1. Weitere Mengendefinitionen

Zur Vereinfachung der Formulierung verschiedener Bedingungen werden die folgenden Mengen definiert:

- Der gesamte Unterricht eines Lehrers $l \in L$ ist definiert als $U_l := \{u = (k_u, f_u, l_u) \in U \mid l_u = l\}$
- Der gesamte Unterricht einer Klasse $k \in K$ ist definiert als $U_k := \{u = (k_u, f_u, l_u) \in U \mid k_u = k\}$

2.4. Harte Bedingungen

Die harten Bedingungen orientieren sich ebenfalls an denen von Weidler [2012, Kap. 4.1].

Als *harte Bedingungen* werden die Anforderungen an den Stundenplan bezeichnet, die auf jeden Fall erfüllt sein müssen, damit dieser in der Praxis ausführbar ist. Ein Stundenplan, der alle harten Bedingungen erfüllt, ist *zulässig*.

2.4.1. Korrekte Anzahl Wochenstunden

Einem Unterricht, der i Mal in der Multimenge W vorhanden ist, müssen genau i Zeitslots zugewiesen werden, d. h. es muss gelten:

$$\forall u \in U : \ |x(u)| = W(u)$$

2.4.2. Konfliktfreiheit

Ein Konflikt entsteht, wenn ein Lehrer oder eine Klasse zu einem Zeitpunkt zwei oder mehr Unterrichte hat. Die Forderung nach Konfliktfreiheit bedeutet also, dass zu jedem Zeitpunkt für jeden Lehrer bzw. jede Klasse maximal ein Unterricht stattfinden darf:

$$\forall z \in Z \ \forall l \in L : |\{u \in U_l \mid z \in x(u)\}| \ \leqslant 1$$
$$\forall z \in Z \ \forall k \in K : |\{u \in U_k \mid z \in x(u)\}| \ \leqslant 1$$

Hinweis: Es gibt zwar Fälle, in denen ein Lehrer mehrere Klassen unterrichtet (z. B. im Sport-Unterricht), dieser Spezialfall ist hier aber nicht gemeint. Diese sogenannten Kopplungen werden in Abschnitt 4.3 näher betrachtet.

2.4.3. Kernstunden

Wegen der Aufsichtspflicht ist es an vielen Schulen erforderlich, dass jeder Schüler den ganzen Vormittag Unterricht hat. Es ist also für jeden Vormittags-Zeitslot und für jede Klasse gefordert, dass mindestens einer der möglichen Unterrichte dann stattfindet (wegen der Konfliktfreiheit ist es dann genau einer).

Wenn man mit $Z_k \subseteq Z$ die Menge Zeitslots bezeichnet, die Kernstunden sein sollen, lässt sich die Kernstunden-Bedingung entweder alleine:

$$\forall z \in Z_k \; \forall k \in K : \; |\{u \in U_k \mid z \in x(u)\}| \; \geqslant 1$$

oder als Verschärfung der Konfliktfreiheit darstellen:

$$\forall z \in Z_k \; \forall k \in K : \; |\{u \in U_k \mid z \in x(u)\}| \; = 1$$

2.4.4. Weitere harte Bedingungen

Es gibt eine Reihe von harten Bedingungen, die nicht an jeder Schule eingehalten werden müssen. Ihre Verwendung ist also optional. Diese Bedingungen werden hier nur kurz beschrieben. Für detaillierte mathematische Formulierungen siehe [Weidler, 2012, Kap. 4.1] und Abschnitt 7.2.

- **Doppelstunden**: Viele Schulen verwenden ein Doppelstundenmodell, das mehr oder weniger strikt angewandt wird. Für die strikteste Variante (ausschließlich Doppelstunden) kann man einfach längere/weniger Zeitslots definieren. Für flexiblere Doppelstundenmodelle bietet SchulScheduler (siehe Kapitel 4) die Definition von Doppelstundenpaaren $(s_1, s_2) \in S^2$ an. Dann ist gefordert, dass ein Unterricht genau dann in der einen Stunde stattfindet, wenn er am selben Tag in der anderen Stunde stattfindet. Aus den angegebenen Doppelstundenpaaren wird also eine Menge $D \subset Z^2$ von disjunkten Zeitslotpaaren abgeleitet, von denen ein Unterricht immer entweder beide oder keine Zeitslots eines Paares belegen muss. Eine einzelne Unterrichtsstunde pro Fach kann jedoch von dieser Regel ausgenommen werden, wenn der Unterricht in dem Fach eine ungerade Anzahl Wochenstunden hat.
- **Fach pro Tag**: Es wird gefordert, dass der Unterricht in einem Fach für eine Klasse maximal für zwei Stunden pro Tag stattfindet – und wenn es zwei Stunden sind, müssen diese aufeinanderfolgend sein.
- **Lehrer-Nichtverfügbarkeit**: Wenn ein Lehrer zu bestimmten Zeitpunkten nicht verfügbar ist, kann gefordert werden, dass keiner seiner Unterrichte zu diesen Zeitpunkten stattfinden darf.

- **Freier Tag für Lehrer**: Für einen Lehrer kann gefordert werden (wenn er dies wünscht), dass mindestens ein Wochentag für ihn komplett unterrichtsfrei ist.
- **Fixierte Unterrichtsstunden**: Mitunter ist es notwendig, bestimmte Unterrichtsstunden auf einen bestimmten Zeitslot zu fixieren. Dabei können Unterrichtsstunden im selben Fach übrig bleiben, die nicht fixiert wurden, und dann normal verplant werden.
- **Kopplungen**: Manche Fächer werden von mehreren Lehrern und/oder bei mehreren Klassen gleichzeitig unterrichtet. Diese sogenannten Kopplungen werden in dieser Arbeit nicht als zusätzliche harte Bedingungen behandelt, sondern als eine erweiterte Art von Unterrichten interpretiert und in Abschnitt 4.3 genauer beschrieben.

2.5. Weiche Bedingungen

Die Erfüllung der harten Bedingungen ist entweder unmöglich (dann ist die Probleminstanz nicht lösbar), oder auf viele Arten möglich. Zum Beispiel lässt sich durch Vertauschen von Zeitslots oder Tagen aus einer zulässigen Lösung schnell eine weitere zulässige Lösung ableiten (mit wenigen Ausnahmen).

Daher bietet es sich an, anhand der Präferenzen der Schule aus den zulässigen Lösungen eine besonders gute Lösung auszuwählen. Das geschieht mittels einer *Zielfunktion*, deren Wert maximiert werden soll. In die Zielfunktion gehen verschiedene *weiche Bedingungen* ein, ggf. mit differenzierter Gewichtung. Diejenige zulässige Lösung, die unter allen zulässigen Lösungen den Wert der Zielfunktion maximiert, wird *optimale Lösung* genannt.

An dieser Stelle werden die weichen Bedingungen vorgestellt, die von SchulScheduler (siehe Kapitel 4) unterstützt werden. Für Details zur mathematischen Modellierung siehe [Weidler, 2012, Kap. 4.2].

- **Harte Fächer vormittags**: Als harte Fächer bezeichnet man mental anspruchsvolle Fächer wie Mathematik, Deutsch oder Physik. Zur Optimierung eines Stundenplans können diese bevorzugt vormittags eingeplant werden.
- **Folgen harter Fächer**: Außerdem können aufeinanderfolgende Unterrichtsstunden in harten Fächern vermieden werden.
- **Unterrichtspriorität**: Eine Verfeinerung bzw. Abschwächung der harten Kernstunden-Bedingung ist die bevorzugte Belegung von bestimmten Zeitslots. Zum Beispiel kann Unterricht bevorzugt vormittags stattfinden, wohingegen später Nachmittagsunterricht vermieden wird. Für jede Stunde des Tages wird dazu eine Priorität angegeben.

- **Lehrer-Verfügbarkeit**: Es kann auf die zeitlichen Präferenzen von Lehrern Rücksicht genommen werden, indem diese angeben, wann sie eher unterrichten möchten und wann sie eigentlich keine Zeit haben.
- **Hohlstunden-Vermeidung**: Da für Klassen oft die Einhaltung von Kernstunden gefordert wird (siehe oben), ist die Vermeidung von Hohlstunden als weiche Bedingung vor allem für Lehrer interessant. Hohlstunden sind Zeitslots, die nach dem ersten und vor dem letzten Unterricht des Tages für einen Lehrer liegen, in denen er aber keinen Unterricht gibt.
- **ÖPVN-Abstimmung**: Der Zeitpunkt der letzten Unterrichtsstunde einer Klasse kann beispielsweise an den Busfahrplan angepasst werden.

Es sind viele weitere weiche Bedingungen denkbar. Insbesondere lässt sich jede der harten Bedingungen zu einer weichen Bedingung umformulieren, bei deren Verletzung ein entsprechend großer Betrag von der Zielfunktion abgezogen wird. Dieser Ansatz ist aber nur vorübergehend für die Berechnung sinnvoll, weil ein Stundenplan, der harte Bedingungen verletzt, in der Praxis nicht eingesetzt werden kann, was einer der Hauptkritikpunkte an bestehenden Softwarelösungen ist.

2.6. NP-Vollständigkeit

Even et al. [1975] wiesen nach, dass bereits eine sehr „primitive“ Variante des Stundenplanproblems NP-vollständig ist. Für den Beweis wurde das von Gotlieb [1962] vorgeschlagene Stundenplanproblem so weit vereinfacht, dass alle in der Praxis vorkommenden Varianten mindestens so komplex sind wie die untersuchte. Die Problemdefinition besteht nur aus Zeitslots, Lehrern, Klassen (wie oben beschrieben) und einer Wochenstunden-Matrix, die neben der Konfliktfreiheit eingehalten werden muss. Außerdem sind Nichtverfügbarkeiten nicht nur für Lehrer, sondern auch für Klassen vorgesehen.

Bei der hier vorgestellten Problemvariante kommt es zwar nicht vor, dass Klassen zu bestimmten Zeiten nicht verfügbar sind, doch diese Einschränkung lässt sich leicht nachbilden, indem man ein Platzhalter-Fach pro Klasse mit einem Lehrer einführt, der immer Zeit hat, und dann fixe Stunden in diesem Fach auf die zu sperrenden Zeitslots legt. Damit ist auch das hier vorgestellte Stundenplanproblem NP-vollständig.

Kapitel 3

Lösungsansätze

Dieses Kapitel gibt eine kurze Einführung der Lösungsansätze, auf die sich diese Arbeit später bezieht. Neben dem manuellen Planen in Abschnitt 3.1 und dem Setz-Algorithmus in Abschnitt 3.2 wird insbesondere der auf ganzzahliger, linearer Programmierung aufbauende Ansatz von Weidler [2012] vorgestellt (Abschnitt 3.3).

Für einen ausführlichen Überblick über die Vielzahl an möglichen Ansätzen zur Lösung von verschiedenen Varianten des Stundenplanproblems sei verwiesen auf die Studien von Carter und Laporte [1998], Schaerf [1999], Burke et al. [2004] und Pillay [2013].

3.1. Stundenplanung von Hand

Vor ein paar Jahren wurden Stundenpläne noch überwiegend von Hand erstellt. Auch wenn mittlerweile viele Schulen auf automatisierte Lösungen umgestiegen sind, gibt es immer noch – vor allem kleinere – Schulen, die ihren Stundenplan weiterhin von Hand erstellen.

Die Vorgehensweise variiert dabei stark und hängt von der Erfahrung und den Zielsetzungen des Stundenplaners ab. Ein bekannter Ansatz ist die Verwendung einer großen Magnettafel, auf der jede Unterrichtsstunde mit zwei Plättchen dargestellt wird – eines davon im Stundenplan der Klasse, das andere im Stundenplan des Lehrers. Ausgehend von einer leeren Tafel wird zunächst ein (in sich stimmiger) Stundenplan für eine Klasse erstellt. Das wird dann für die weiteren Klassen wiederholt, wobei Konflikte erkannt werden, indem der jeweilige Unterricht parallel auch in den Lehrerplänen eingetragen wird. Fälle, in denen eine Unterrichtsstunde gar nicht mehr platziert werden kann, können dann nur mit Ausprobieren, Erfahrung oder Neu-Beginnen gelöst werden. (Quelle: eine typische Grundschule aus Süddeutschland)

Dieser Ansatz nimmt zwar auch für Grundschulen einige Zeit in Anspruch, aber in den allermeisten Fällen führt er zu einem zulässigen *und guten* Ergebnis – aufgrund der Erfahrung der Stundenplaner. Deswegen muss er durchaus als Konkurrenz zur automatisierten Stundenplanung betrachtet werden.

Konfliktfreie Stundenpläne lassen sich an Grundschulen vergleichsweise einfach erstellen, weil jede Klasse einen eigenen Klassenlehrer hat, der bis auf wenige Ausnahmen (oft Sport oder Religion) alle Fächer unterrichtet. Dadurch sind je ein Klassen- und Lehrerplan fast identisch und außerdem weitgehend unabhängig von den anderen Plänen. Weil an Grundschulen nur wenige Wahlmöglichkeiten und weniger Abhängigkeiten zu anderen Schulen oder externen Räumen (insbesondere Sporthallen) bestehen, ändern sich die Anforderungen zum neuen Schuljahr nur wenig, sodass oft der alte Stundenplan als Vorlage verwendet werden kann. Für weiterführende Schulen ist der manuelle Ansatz allerdings zunehmend weniger geeignet, weil sie meist größer sind und komplexere und wechselnde Anforderungen haben.

3.2. Setz-Algorithmus

Durch die Übersetzung des manuellen Vorgehens in einen maschinell ausführbaren Algorithmus erhält man einen einfachen Setz-Algorithmus. Die zu verplanenden Unterrichtsstunden werden der Reihe nach auf einen noch freien Zeitslot gelegt, bis alle untergebracht sind. Kann eine Unterrichtsstunde nicht mehr untergebracht werden, wird mittels Backtracking nach anderen Kombinationen gesucht.

Dieser zunächst naive Algorithmus lässt sich durch Heuristiken stark verbessern [siehe z. B. Schmidt und Ströhlein, 1980]. Nachdem ein zulässiger Stundenplan gefunden ist, kann dieser durch Tauschoperationen verbessert werden, bei denen zwei oder sogar mehr Unterrichte vertauscht werden, ohne dabei Konflikte zu erzeugen. Heutzutage in Deutschland etablierte Software-Produkte wie beispielsweise Untis® Express arbeiten nach eigener Aussage mitunter auf diese Weise – wenngleich sicherlich mit einigen Verbesserungen:

> „Während der Setzoptimierung werden die einzelnen Unterrichtsstunden – beginnend mit der schwierigsten – in den zunächst noch leeren und sich langsam füllenden Zeitraster hineingesetzt, während der Tauschoptimierung versucht das Programm durch gezielte Täusche das Ergebnis zu verbessern.“
>
> Gruber & Petters [2014], Untis® Express Benutzerhandbuch

3.3. Ganzzahlige lineare Programmierung

Das in Kapitel 2 vorgestellte Stundenplanproblem kann als lineares Programm formuliert und dann mit entsprechenden Lösern gelöst werden. An dieser Stelle wird nur die Grundidee wiedergegeben. Für Details siehe [Weidler, 2012].

Bei einem linearen Programm (LP) ist eine Zielfunktion

$$\max \sum_{j=1}^{n} c_j x_j$$

gegeben, die unter Einhaltung der ebenfalls gegebenen Nebenbedingungen

$$\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i \qquad\qquad i \in \{1, \ldots, m\}$$

maximiert werden soll.

Hinweis: Nebenbedingungen mit $=$ oder $\geqslant$ lassen sich einfach in die obige Form umformen.

Für die Berechnung von Stundenplänen ist insbesondere die ganzzahlige lineare Optimierung interessant (ILP), bei der zusätzlich gefordert wird, dass $x_j \in \mathbb{N}$.

3.3.1. Modellierung als ganzzahliges lineares Programm

Wir definieren für jeden Unterricht $u \in U$ und jeden möglichen Zeitslot $z \in Z$ eine binäre Variable $x_{uz} \in \{0, 1\}$. Diese Variable hat den Wert 1, wenn der Unterricht (unter anderem) zu diesem Zeitpunkt stattfindet, und sonst den Wert 0.

Damit lässt sich die Forderung nach der korrekten Wochenstundenzahl für jeden Unterricht formulieren als:

$$\forall u \in U : \sum_{z \in Z} x_{uz} = W(u)$$

und die Bedingung der Konfliktfreiheit für Klassen (analog für Lehrer) als:

$$\forall z \in Z \; \forall k \in K : \sum_{u \in U_k} x_{uz} \leqslant 1$$

Analog dazu lautet die Kernstunden-Bedingung:

$$\forall z \in Z_k \; \forall k \in K : \sum_{u \in U_k} x_{uz} \geqslant 1$$

Aus der fertigen Belegung aller x_{uz} erhält man dann die gesuchte Zuteilung $x : U \to \mathcal{P}(Z)$ als:

$$x(u) := \{z \in Z \mid x_{uz} = 1\}$$

Die LP-Formulierung des Problems wird später noch in Kapitel 6 verwendet werden. Außerdem verwendet die Software SchulScheduler (siehe Kapitel 4) dieses ganzzahlige lineare Programm, um Stundenpläne zu berechnen. Formulierungen für einige weitere Bedingungen werden in [Weidler, 2012, Kap. 4] und ergänzend in Abschnitt 7.2 vorgestellt.

3.4. Weitere Ansätze

3.4.1. Reduktion

Analog zur Modellierung als lineares Programm kann das Stundenplanproblem auf andere Problemtypen reduziert und mit einem bestehenden, möglichst effizienten Solver gelöst werden. Die folgenden Kapitel der vorliegenden Arbeit befassen sich hauptsächlich mit diesem Ansatz.

3.4.2. Optimierungs-Frameworks

Eine ähnliche Herangehensweise ist, mit Hilfe eines Frameworks einen evolutionären Algorithmus oder Ähnliches zu implementieren. Die verwendeten Optimierungsverfahren stammen meist aus dem Bereich der künstlichen Intelligenz. Dabei müssen ebenfalls nur die problemspezifischen Teile implementiert werden, es findet aber keine Reduktion statt, d. h. es entsteht keine Probleminstanz eines allgemeineren Problemtyps.

3.4.3. Spezialisierte Algorithmen

Genau wie Untis® Express verwenden auch viele der Löser, die bei der dritten International Timetabling Competition [Post et al., 2013] eingereicht wurden, Algorithmen mit spezifischem Wissen über die Stundenplanung. Einige davon basieren auf der Kingston High School Timetabling Engine [Kingston, 2014], die bereits selbst umfangreiche spezifische Implementierungen bietet.

Da in dieser Arbeit ein anderer Ansatz verfolgt wird, werden an dieser Stelle keine einzelnen Algorithmen genauer erläutert. Abschnitt 8.2 gibt einen kurzen Überblick über aktuelle Techniken und enthält eine kurze Evaluation des Solvers von Kingston [2014].

Kapitel 4

SchulScheduler

In diesem Kapitel wird die Software SchulScheduler vorgestellt, deren Infrastruktur und Datensätze für diese Arbeit verwendet wurden. Abschnitt 4.2 gibt einen Einblick in die Oberfläche der Software. Abschnitt 4.3 behandelt mit den Kopplungen eine in der Praxis wichtige Besonderheit, die in SchulScheduler implementiert ist und in dieser Arbeit ebenfalls ausführlich betrachtet wird. In Abschnitt 4.4 werden die Testdatensätze von SchulScheduler beschrieben. Als Referenz werden in Abschnitt 4.5 die Laufzeiten von SchulScheduler mit dem Solver Gurobi auf diesen Datensätzen aufgeführt.

4.1. Einführung

Basierend auf dem von Weidler [2012] eingeführten und in Unterabschnitt 3.3.1 vorgestellten Ansatz wurde 2013-2014 in einem Studienprojekt an der Universität Stuttgart die Software SchulScheduler entwickelt. Ziel des Projekts war insbesondere die Bereitstellung einer Benutzeroberfläche, die von Lehrern bedient werden kann, die wenig Erfahrung im Umgang mit Computern und mit manueller Stundenplanung haben. Dadurch stehen nun eine Oberfläche sowie ein flexibles Datenmodell und ein Framework für die Implementierung von Algorithmen zur Verfügung. Außerdem sind im Rahmen des Projekts verschiedene Testdatensätze entstanden, die in dieser Arbeit zum Testen der Implementierungen verwendet werden. Der einzige bislang implementierte Algorithmus verwendet den kommerziellen Löser Gurobi von Gurobi Optimization, Inc. [2014] zur Lösung des ganzzahligen linearen Programms.

Die Software ist in Java 8 und Xtend implementiert und genau wie diese Arbeit unter der Apache License 2.0 veröffentlicht. Für die Erstellung einer zeitgemäßen und funktionalen Oberfläche wurde JavaFX 8.0 verwendet.

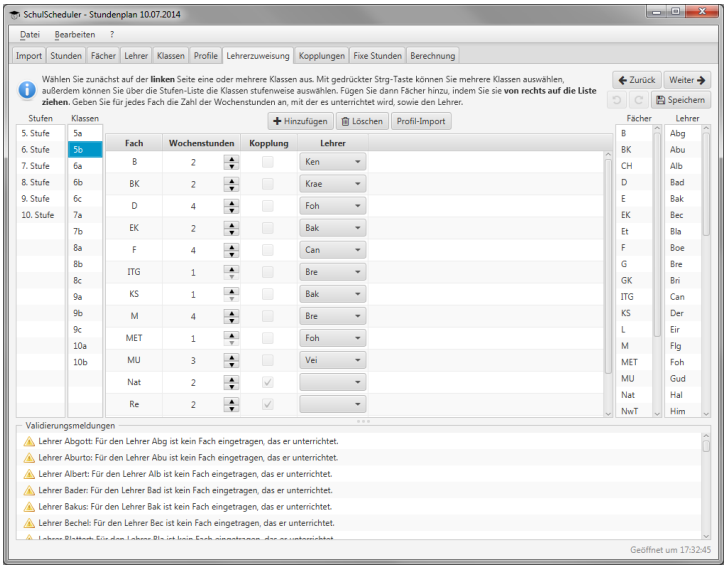

Abbildung 4.1. SchulScheduler Eingabefenster

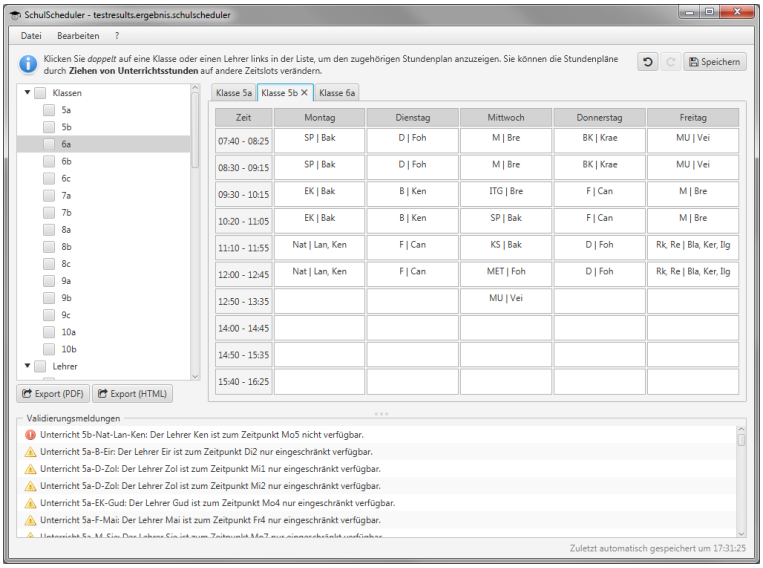

Abbildung 4.2. SchulScheduler Ergebnisfenster

4.2. Oberfläche

In Abbildung 4.1 und Abbildung 4.2 ist die Benutzeroberfläche der Software dargestellt. Insbesondere die schnelle Darstellung, die einfache Veränderbarkeit und die sofortige Validierung der Ergebnisse war für die Erstellung dieser Arbeit eine große Hilfe. Per Drag-and-drop können Unterrichte versuchsweise umgeordnet werden. Alle verletzten Bedingungen werden direkt erkannt und unten im Fenster aufgelistet, sodass sich die Ergebnisse, die mit den hier vorgestellten Ansätzen generiert wurden, leicht überprüfen und beurteilen ließen.

4.3. Kopplungen

Zusätzlich zu den in Kapitel 2 vorgestellten Bestandteilen einer Probleminstanz kennt SchulScheduler noch einige weitere Elemente. Ein wesentliches Konzept sind dabei die Kopplungen. Typischerweise treffen im Sport- oder Religions-Unterricht mehrere Klassen/Lehrer zum Unterricht aufeinander. Beispielsweise könnten drei Klassen zum Sportunterricht bei zwei Lehrern zusammengelegt werden und dann die Mädchen und Jungen getrennt unterrichtet werden. Auch beim Religionsunterricht handelt es sich um eine Kopplung, selbst wenn beispielsweise nur eine Klasse beteiligt ist, die aber nach evangelischer und katholischer Konfession sowie Ethik aufgeteilt und von jeweils einem Lehrer unterrichtet wird.

Während die Definition und Eingabe von Kopplungen relativ kompliziert zu realisieren sind, sind die Auswirkungen auf die eigentliche Berechnung der Lösung nur minimal: An einem Unterricht $u \in U \subseteq K \times F \times L$ können nun mehrere Klassen, Fächer und/oder Lehrer beteiligt sein, d. h. es ist nun $U \subseteq \mathcal{P}(K) \times \mathcal{P}(F) \times \mathcal{P}(L)$. Dementsprechend kann ein einzelner Unterricht $u \in U$ in den Mengen U_k und U_l von mehreren Klassen und Lehrern vorkommen. Alle bisher genannten Bedingungen lassen sich damit auf gleiche oder sehr ähnliche Weise formulieren. Insbesondere ändert sich nichts an der Grundidee der Algorithmen, eine Menge von Unterrichtsstunden, die gewisse Abhängigkeiten haben, in einem Zeitraster einplanen zu müssen. Lediglich die Verknüpfungen zwischen den Unterrichten (z. B. wechselseitiger Ausschluss) nehmen durch den Einsatz von Kopplungen zu, sodass Datensätze mit vielen Kopplungen tendenziell weniger Lösungen haben.

4.4. Testdatensätze

Während der Durchführung des Studienprojekts sind einige Datensätze entstanden, die bereits im passenden Format vorliegen. Da sie in dieser Arbeit zur Evaluation und zum

Vergleich von Algorithmen verwendet wurden, wird im Folgenden eine kurze Charakterisierung dieser Datensätze angegeben. Aus Datenschutzgründen sind die Datensätze nur mit Buchstaben benannt.

- Datensatz **A**: Hierbei handelt es sich um einen sehr kleinen und einfachen Datensatz, der hauptsächlich zum Testen der Oberfläche entworfen wurde. Es gibt nur drei Stunden pro Tag – und noch weniger Unterricht. Die Hälfte der Zeit ist unterrichtsfrei. Die Lösung dieses Datensatzes gelingt selbst von Hand in kürzester Zeit.
- Datensatz **B**: Dieser Datensatz ist eine Entschärfung von Datensatz C: Statt nur sechs stehen hier acht Stunden pro Tag zur Verfügung, was die Anzahl möglicher Lösungen deutlich erhöht. Außerdem wurde die Größe der Probleminstanz (d. h. die Anzahl der Klassen/Lehrer) halbiert.
- Datensatz **C**: Dieser Datensatz gehört zu einer normalen deutschen Grundschule. Da nur sechs Unterrichtsstunden täglich zur Verfügung stehen, die für die höheren Klassen auch vollständig ausgenutzt werden, ist die Zahl der Lösungen stark beschränkt. Dafür gibt es nur wenig Abhängigkeiten zwischen den Klassen, weil an der Grundschule jede Klasse einen festen Klassenlehrer hat, der die meisten Fächer alleine bestreitet.
- Datensatz **D**: Der mit Abstand größte Datensatz stammt ursprünglich von einem typischen Gymnasium und wurde um einige Sonderfälle erweitert, um die Algorithmus-Implementierung ausgiebig testen zu können. Durch seine Größe und insbesondere die große Anzahl an Kopplungen ist dieser Datensatz relativ schwer optimal zu lösen.

Tabelle 4.1. Größenordnungen der verwendeten Datensätze

Datensatz	A	B	C	D
Klassen	3	5	10	15
Fächer	5	9	9	23
Lehrer	4	7	14	45
Unterrichte	15	35	72	235
Kopplungen	4	3	4	21
Unterrichtsstunden	16	114	235	467
Zeitslots	15	40	30	50
Lehrer-Nichtverfügbarkeiten	3	18	23	202
Doppelstunden	0	0	0	3×2
Fixierte Unterrichtsstunden	2	0	0	18

4.5. Performance

Zum späteren Vergleich zeigt Tabelle 4.2 die Rechenzeiten des Gurobi-basierten Lösungsverfahrens für die einzelnen Datensätze. Zusätzlich zur Berechnung des ganzzahligen

Optimums, welches in der Regel von Interesse ist, sind die Zeiten für das Finden einer zulässigen (d. h. nicht unbedingt optimalen) Lösung sowie für das Lösen der LP-Relaxierung (d. h. eine nicht ganzzahlige Lösung) angegeben.

Tabelle 4.2. Rechenzeiten mit Gurobi

Datensatz	A	B	C	D
Ganzzahlig, optimal	12 ms	181 ms	3,2 s	-[†]
Ganzzahlig, zulässig	4 ms	46 ms	150 ms	1,1 s
Relaxiert, optimal	6 ms	91 ms	119 ms	4,0 s
Relaxiert, zulässig	4 ms	69 ms	86 ms	1,4 s

[†] Verbleibende Gap von 0,18 % nach 40 Minuten

Zu diesen Laufzeiten sei angemerkt, dass keiner von vielen getesteten Open-Source-Solvern wie lp_solve, Clp, etc. (außer SCIP) in der Lage war, die Datensätze B bis D in akzeptabler Zeit (unter einem Tag) zu lösen. Im Fall von Datensatz D gelingt das auch mit Gurobi nicht, jedoch ist es in der Praxis problemlos möglich, die Berechnung nach 40 Minuten oder nach einer Stunde abzubrechen. Dann ist der Unterschied zwischen der aktuell besten bekannten Lösung und der besten bekannten Schranke (genannt „Gap“) bereits kleiner als 0,18 %, d. h. das Ergebnis kann sich nicht mehr wesentlich verbessern und ist möglicherweise sogar schon optimal.

Kapitel 5

Lösung mittels Constraint Programming

Dieses Kapitel beschreibt zwei verschiedene Ansätze, das Stundenplanproblem mit Constraint Programming zu lösen. Zunächst wird in Abschnitt 5.1 das Verfahren allgemein vorgestellt. In Abschnitt 5.2 wird eine Möglichkeit gezeigt, das Stundenplanproblem mit ganzzahligen Variablen als CSP zu modellieren. Abschnitt 5.3 geht auf die Implementierung, Rechenergebnisse und einige Möglichkeiten zur Verbesserung ein. Anschließend wird in Abschnitt 5.4 ein alternativer, auf binären Variablen basierender Ansatz vorgestellt.

5.1. Einführung

Nach van Omme et al. [2014] bezeichnet man mit *Constraint Satisfaction Problem* (CSP) ein mathematisches Modell bestehend aus einer Menge von Variablen $V = \{v_1, \ldots, v_n\}$, möglichen Wertemengen $d_1, \ldots, d_n$ für diese Variablen (Domänen) und einer Menge von Constraints C. Die Constraints können dabei eine nahezu beliebige Form annehmen, solange sie mathematisch formulierbar und algorithmisch propagierbar sind. Insbesondere müssen die Constraints nicht linear sein oder ähnliches.
Das Ziel ist zunächst nur, die Constraints zu erfüllen, d. h. eine *zulässige* Lösung zu finden!

Der Begriff *Constraint Programming* bezeichnet Lösungsverfahren für CSPs.

5.1.1. Bedingungen und Constraints

Bei der Definition des Stundenplanproblems, die sich aus der Realwelt ableitet, wird in dieser Arbeit durchgehend der Begriff „Bedingung“ für Einschränkungen und Kriterien verwendet, die ein Stundenplan erfüllen soll. In Bezug auf die Umsetzung der Bedingungen in einem Solver wird der Begriff „Constraint“ verwendet.

Die Unterscheidung ist wichtig, weil es keine 1:1-Beziehung zwischen Bedingungen und Constraints gibt: Für die Umsetzung einer Bedingung kann nur ein Constraint benötigt werden (z. B. für Lehrer-Nichtverfügbarkeiten) oder mehrere (z. B. für die Konfliktfreiheit)

– und manchmal sogar gar keiner. Umgekehrt können aber auch mehrere Bedingungen mit gemeinsamen Constraints abgedeckt werden (z. B. Konfliktfreiheit und Kernstunden). Wiederum andere Bedingungen sind weich, sodass sie nur wenige oder gar keine Constraints benötigen und stattdessen Einfluss auf die Zielfunktion nehmen.

5.1.2. Lösungsverfahren

Grundsätzlich verwenden alle in dieser Arbeit erwähnten Solver die gleiche Vorgehensweise zur Lösung eines CSP:

- Solange noch nicht alle Variablenwerte bekannt sind, werden eine Variable und ein noch möglicher Wert für diese Variable ausgewählt.
- Es wird entschieden, ob dieser Wert angenommen werden soll oder nicht, und ausgehend von dieser Annahme die Suche fortgesetzt.
- Wenn für eine Variable überhaupt kein Wert mehr möglich ist, werden mittels Backtracking andere Lösungsalternativen gesucht.

Dadurch entsteht ein binärer Suchbaum, den alle verwendeten Solver auch zu bestimmten Zeitpunkten oder sogar in Echtzeit visualisieren können.

Um früh bestimmte Werte für eine Variable ausschließen zu können, wird für jede Variable eine Menge der noch möglichen Werte verwaltet. Wenn diese Menge sich ändert (z. B. durch eine Festlegung), hat dies Auswirkungen auf die möglichen Werte für andere Variablen. Beispiel: Wenn der Mathematik-Unterricht der 5c auf den Zeitslot Mo3 gelegt wird, kann der Deutsch-Unterricht der 5c sicher nicht in dieser Stunde stattfinden, d. h. der Wert kann für die betreffende Variable ausgeschlossen werden – lange bevor die Variable überhaupt im Suchbaum selbst betrachtet wird. In dem „Constraint Propagation“ genannten Verfahren wird bei der Änderung der möglichen Werte einer Variablen, die man sich als Knoten in einem Netzwerk vorstellt, die Änderung über die einzelnen Constraints, die man sich als Kanten vorstellt, zu den anderen Variablen propagiert, die davon betroffen sein könnten. Für Details zum Verfahren und insbesondere zu den konkreten Implementierungen siehe [van Omme et al., 2014] und [Schulte und Tack, 2013].

Eine wirkungsvolle Möglichkeit zur Einflussnahme auf die Suche ist die Festlegung der Heuristiken zur Variablen- und Werteauswahl – mehr dazu in Unterabschnitt 5.3.2.

5.1.3. Optimierung mit Constraint Programming

Auch wenn die korrekte Lösung eines CSP nicht notwendigerweise optimal ist, sondern im Regelfall einfach nur irgendeine zulässige Lösung darstellt, kann man mit Constraint Programming Optimierungsprobleme lösen. Dazu benötigt man eine Zielfunktion $z : d_1 \times \cdots \times d_n \to \mathbb{R}$, die mögliche Lösungen bewertet. Auch für diese Zielfunktion gibt es *keine* Beschränkungen wie Linearität oder ähnliches – sie sollte aber mit geringem Aufwand berechenbar sein. Wir gehen ohne Beschränkung der Allgemeinheit davon aus, dass die Zielfunktion zu minimieren ist.

Um eine optimale Lösung zu erhalten, berechnet man zunächst eine zulässige Lösung x_0 und ihren Zielfunktionswert $z(x_0)$. Um die gefundene i-te Lösung zu verbessern, fügt man den Constraint $z(v_0, \ldots, v_n) < z(x_i)$ hinzu und berechnet eine neue, bessere Lösung x_{i+1}. Diesen Vorgang wiederholt man so lange, bis das CSP unlösbar wird. Die unmittelbar davor gefundene Lösung ist nicht nur zulässig, sondern auch optimal. [van Omme et al., 2014]

Die meisten CSP-Solver bieten bereits eine eingebaute Unterstützung für dieses oder ähnliche Optimierungsverfahren an, sodass man die gewünschte Zielfunktion ohne zusätzlichen Implementierungsaufwand direkt angeben kann.

5.2. Erste Modellierung als Constraint Satisfaction Problem

Die Stärke von Constraint Programming liegt darin, dass praktisch beliebige Constraints möglich sind. Insbesondere der `alldifferent`-Constraint ist sehr gut erforscht [siehe López-Ortiz et al., 2003; van Hoeve, 2001] und wurde deshalb für den ersten Modellierungsansatz verwendet. Dieser Ansatz ist naheliegend, weil er das Stundenplanproblem direkt abbildet:

Die Grundidee ist, die einzelnen Unterrichtsstunden jeweils einem Zeitslot zuzuordnen, ohne Konflikte zu erzeugen. Entscheidend ist, dass nicht die Unterrichte (also z. B. Mathematik Klasse 5c) sondern die einzelnen Unterrichtsstunden zugeordnet werden. Für jede Wochenstunde eines Unterrichts $u \in U$ wird eine Variable x_{ui} im CSP erstellt mit $i \in \{1, \ldots, W(u)\}$, sodass es insgesamt $\sum_{u \in U} W(u)$ Variablen gibt. Als Domäne wird für alle Variablen die Menge der verfügbaren Zeitslots Z gewählt. Durch diese Art der Modellierung wird die Forderung nach der korrekten Anzahl Wochenstunden (vgl. Unterabschnitt 2.4.1) automatisch eingehalten.

Die in Abschnitt 2.3 geforderte Zuteilung $x : U \to \mathcal{P}(Z)$ erhält man aus der Lösung des CSP, indem man die zugewiesenen Werte aller Variablen eines Unterrichts zu einer Menge zusammenfasst:

$$x(u) = \{x_{ui} \mid i \in \{1, \ldots, W(u)\}\}$$

5.2.1. Konfliktfreiheit

Die Konfliktfreiheit (vgl. Unterabschnitt 2.4.2) fordert, dass alle Unterrichte einer Klasse sowie alle Unterrichte eines Lehrers jeweils zu unterschiedlichen Zeitpunkten stattfinden. Das lässt sich unmittelbar mit dem `alldifferent`-Constraint modellieren, der fordert, dass allen Variablen einer bestimmten Variablenmenge paarweise verschiedene Werte zugewiesen werden:

$$\forall l \in L: \texttt{alldifferent}(\{x_{ui} \mid u \in U_l, i \in \mathbb{Z}\})$$
$$\forall k \in K: \texttt{alldifferent}(\{x_{ui} \mid u \in U_k, i \in \mathbb{Z}\})$$

5.2.2. Kernstunden

Zur Umsetzung der Kernstunden-Bedingung bieten sich verschiedene Constraints an, die je nach konkreter Solver-Implementierung unterschiedlich unterstützt werden:

Eine für sich genommen ineffiziente Möglichkeit ist, zunächst für jedes Unterricht-Zeitslot-Paar eine binäre Statusvariable zu erstellen, die aussagt, ob der Unterricht zu diesem Zeitpunkt stattfindet. Damit können die Kernstunden analog zum ILP modelliert werden (siehe Unterabschnitt 3.3.1). Dieser Ansatz ist vor allem dann sinnvoll, wenn die Statusvariablen noch für weitere Constraints verwendet werden können, die über den Umfang dieser Arbeit hinausgehen.

Deutlich effizienter ist eine Art der Modellierung, die keine zusätzlichen (Status-)Variablen erfordert, und stattdessen den Constraint $\texttt{count}(v, w, n)$ verwendet, der fordert, dass genau n der Variablen aus der Variablenmenge v den Wert w annehmen sollen. Für eine Kernstunde muss pro Klasse genau eine der zu diesem Zeitpunkt möglichen Unterrichtsstunden auf den Zeitslot gelegt werden:

$$\forall z \in Z_k \; \forall k \in K: \texttt{count}(\{x_{ui} \mid u \in U_k, i \in \mathbb{Z}\}, z, 1)$$

5.2.3. Symmetrische Lösungen

Eine klare Schwäche dieser Form der Modellierung ist die Erzeugung von (exponentiell vielen) symmetrischen Lösungen, die den Lösungsraum und damit den Suchbaum für den Solver unnötig stark vergrößern. Dadurch, dass jeder Unterricht in seine einzelnen Unterrichtsstunden aufgespalten wird (z. B. die beiden Deutsch-Stunden der 5c), gibt es mehrere verschiedene Lösungen, die sich für den Endanwender aber nicht unterscheiden. Im gegebenen Beispiel wäre es egal, ob die erste der beiden Deutsch-Stunden in der ersten Stunde montags stattfindet und die zweite in der zweiten Stunde, oder umgekehrt. Da die einzelnen Unterrichtsstunden in einem Fach völlig austauschbar sind, kann man den Lösungsraum erheblich verkleinern, indem man eine Ordnung der Unterrichtsstunden

erzwingt, d. h. dass die erste Unterrichtsstunde in dem Fach vor der zweiten stattfindet, die zweite vor der dritten, und so weiter:

$$\forall u \in U \; \forall i < j : \quad x_{ui} < x_{uj}$$

Ein solches Verfahren zur Vermeidung von symmetrischen Lösungen wird „Symmetry Breaking“ genannt.

5.3. Implementierung und Verbesserung

Um die Praxistauglichkeit dieses Ansatzes zu überprüfen, wurde diese Modellierung mit den beiden CSP-Solvern Or-Tools [Or-Tools Team, 2010] und Gecode 4.2.1 [Gecode Team, 2006] implementiert und mit den in Abschnitt 4.4 vorgestellten Datensätzen getestet. Beide ausgewählten Solver gehörten in den vergangenen Jahren zu den besten CSP-Solvern [Stuckey et al., 2010]. Bei Or-Tools handelt es sich um einen Solver, der ein Java-Interface anbietet und daher direkt aus der SchulScheduler-Architektur heraus angesteuert werden kann. Auch wenn Gecode direkt aus einer C++-Anwendung heraus gestartet wurde, hat dies keine nennenswerten Auswirkungen auf die Laufzeiten, da Or-Tools intern ebenfalls eine native Bibliothek verwendet und in die Zeitmessung nur die reine Ausführung einbezogen wurde, d. h. nicht das Erstellen der Variablen und Constraints.

Beim Vergleich der Ergebnisse mit den in Abschnitt 4.5 angegebenen Laufzeiten ist es wichtig zu beachten, dass in dieser CSP-Implementierung lediglich eine zulässige Lösung gesucht wurde, während bei der Berechnung mit Gurobi die optimale Lösung gefunden wurde. Zusätzlich zu den Rechenzeiten ist jeweils die Anzahl der Variablen und Constraints angegeben, die durch die jeweilige Art der Modellierung entstanden sind.

5.3.1. Nur Konfliktfreiheit

Zunächst wurden alle Probleminstanzen ausschließlich mit der Bedingung für die Konfliktfreiheit gelöst.

Tabelle 5.1. CSP nur mit Konfliktfreiheits-Bedingung

Datensatz	A	B	C	D
Anzahl Variablen	16	114	236	467
Anzahl Constraints	7	12	23	59
Rechenzeit (Or-Tools)	1 ms	2 ms	4 ms	8 ms
Rechenzeit (Gecode)	0 ms	1 ms	4 ms	14 ms

5.3.2. Suchstrategien

Die in Tabelle 5.1 dargestellten Ergebnisse wurden bereits mit einer gezielten Optimierung bezüglich der Suchstrategie erzielt. Das grundsätzliche Vorgehen der Solver wurde bereits in Unterabschnitt 5.1.2 beschrieben und kann durch geeignete Heuristiken für die Variablen- und Wertewahl gesteuert werden. Hierbei stellen sich die Heuristiken als am effizientesten heraus, die auch dem natürlichen menschlichen Vorgehen beim Setz-Algorithmus (siehe Abschnitt 3.2) entsprechen würden.

Ohne weitere Konfiguration wählen die Solver immer die erste Variable (in der Reihenfolge, in der sie hinzugefügt wurden) und weisen ihr den niedrigsten möglichen Wert zu. Infolgedessen findet viel Unterricht montags statt (auch nachmittags) und wenig am Freitag. Jedoch schlägt das Verfahren beim Datensatz C fehl und es kann auch nach über drei Stunden Rechenzeit keine Lösung gefunden werden. Die Ursache ist vermutlich eine Konfliktsituation, die bereits durch relativ frühe Entscheidungen verursacht wurde und für die „späteren“ Variablen keine mögliche Aufteilung mehr übrig lässt.

Um dem Problem zu begegnen, setzt man bei der manuellen Planung vernünftigerweise zuerst die Unterrichte ins Stundenraster, für die die wenigsten möglichen Plätze übrig bleiben. Der CSP-Solver muss also diejenige Variable auswählen, die die kleinste verbleibende Wertemenge hat. Mit dieser Heuristik, die den Solvern als „Choose Min Size“ bekannt ist, lässt sich auch für den Datensatz C eine Lösung in wenigen Millisekunden finden.

Eine Heuristik zur Auswahl geeigneter Werte bringt in diesem Fall keine weiteren Vorteile. Zwar existieren in der Literatur Heuristiken, die die Werte so auswählen, dass die unmittelbaren negativen Auswirkungen durch die Propagierung möglichst gering sind [Dechter, 2003, Kapitel 5.3.1], aber diese sind weder in Or-Tools noch Gecode implementiert. Stattdessen lässt sich nur anhand von Minimum und Maximum einer der noch möglichen Werte auswählen (z. B. der Mittelwert, das Maximum oder ein zufälliger Wert). Da aber für das Stundenplanproblem in der einfachen, hier betrachteten Art alle Zeitslots (und damit Werte) gleich sind, ist eine Werteheuristik überflüssig.

5.3.3. Ausschließlich lineare Suchbäume

Bemerkenswerterweise terminiert jede Berechnung binnen weniger Millisekunden, wenn eine günstige Heuristik angewendet wird, oder sie benötigt mehrere Stunden oder Tage, abhängig von der Problemgröße. In den Fällen, in denen die Berechnung nach kürzester Zeit erfolgreich abgeschlossen werden kann, ist der Suchbaum stets linear. Das bedeutet, dass *jede* vom Solver getroffene Entscheidung richtig war und demnach kein Backtracking stattgefunden hat. Bei den meisten Datensätzen (bis auf Datensatz C, siehe vorheriger Abschnitt) funktioniert das sogar mit randomisierten Heuristiken anstatt gezieltem Auswählen der günstigsten Variablen. Dabei wird natürlich nicht eine zufällige Lösung erstellt,

die ja mit sehr hoher Wahrscheinlichkeit unzulässig wäre, sondern es wird immer dann zufällig ausgewählt, wenn mehrere noch mögliche Variablen oder Werte übrig sind. Für das erfolgreiche Finden der Lösung ist in diesen Fällen also ausschließlich die Constraint Propagation verantwortlich.

5.3.4. Ausschluss von symmetrischen Lösungen

Wie in Unterabschnitt 5.2.3 beschrieben wurde für die einzelnen Wochenstunden eines Unterrichts eine Ordnung eingeführt, um symmetrische Lösungen zu vermeiden. Während für Or-Tools einzelne $<$-Constraints benötigt werden, stellt Gecode einen Constraint für $x_0 < \cdots < x_n$ direkt zur Verfügung.

Tabelle 5.2. CSP mit Konfliktfreiheit und Symmetry Breaking

Datensatz	A	B	C	D
Anzahl Variablen	16	114	236	467
Anzahl Constraints (Or-Tools)	11	98	203	339
Anzahl Constraints (Gecode)	11	40	79	217
Rechenzeit (Or-Tools)	1 ms	2 ms	6 ms	11 ms
Rechenzeit (Gecode)	0 ms	1 ms	4 ms	15 ms

Auffällig ist, dass die Rechenzeiten im Vergleich zur Berechnung ohne Symmetry Breaking entweder gleich geblieben oder größer geworden sind, obwohl dadurch eigentlich die Berechnung beschleunigt werden sollte. Das ist damit zu begründen, dass die Wirkungsweise des Symmetry Breakings noch gar nicht zur Geltung kommen konnte: Die Idee ist, den Suchbaum zu verkleinern, sodass nach einem Backtracking-Schritt nicht nochmals die (symmetrisch) gleichen Lösungen durchsucht werden. Da aber gar kein Backtracking stattfindet, sondern direkt eine Lösung gefunden wird (egal welche der verschiedenen symmetrischen Lösungen das ist), bewirkt das Symmetry Breaking nur, dass nun eine bestimmte der symmetrischen Lösungen gefunden wird. Dazu ist zunächst ein größerer Rechenaufwand beim Propagieren nötig, weil es mehr Constraints gibt.

Dennoch wurde auch bei allen folgenden Berechnungen das Symmetry Breaking beibehalten, weil sich in Fällen, in denen die Lösung nicht sofort gefunden werden kann, eine signifikante Verbesserung ergibt.

5.3.5. Berechnung mit Kernstunden

Die Kernstunden-Bedingung wurde in beiden Solvern, wie oben beschrieben, mit einem `count`-Constraint implementiert. Durch das Hinzufügen der Bedingung ist es mit keiner der Heuristiken mehr möglich, innerhalb von einer Stunde eine Lösung zu finden (siehe Tabelle 5.3).

Tabelle 5.3. CSP mit Kernstunden

Datensatz	A	B	C	D
Anzahl Variablen	16	114	236	467
Anzahl Constraints (Or-Tools)	11	312	639	1573
Anzahl Constraints (Gecode)	11	140	279	517
Rechenzeit (Or-Tools)	1 ms†	>1 h	>1 h	>1 h
Rechenzeit (Gecode)	0 ms†	>1 h	>1 h	>1 h

† Der Datensatz A enthält keine Kernstunden

Die Kernstunden sind eine Bedingung, die die Constraint Propagation alleine nicht erfüllen kann. Denn es genügt nun nicht mehr, alle Werte konfliktfrei zu belegen, sondern es muss am Ende auch mindestens eine Variable aus einer Teilmenge einen bestimmten Wert haben – obwohl zum Zeitpunkt der Belegung noch viele andere Werte konfliktfrei möglich gewesen wären. Deswegen sind die Solver nun auf Backtracking angewiesen. Doch wie man an den Laufzeiten sieht, scheint das Backtracking nicht effizient genug zu sein. Alle Berechnungen dauerten so lange, dass sie nach einer Stunde abgebrochen werden mussten. Zur Erinnerung: Der ILP-Solver Gurobi berechnet auf diesen Datensätzen eine *optimale* Lösung in weit unter einer Stunde. Versuchsweise wurde die Berechnung mit dem einfachsten Datensatz (B) für längere Zeit durchgeführt, aber es wurde auch nach über acht Stunden keine Lösung gefunden.

Selbst nach einer Stunde beginnt der Suchbaum der Berechnung mit Datensatz B (siehe Abbildung 5.1) noch mit einer linearen Kette, die fast so lang ist wie zu Beginn der Berechnung. Aus der binären Struktur des Baums ist ersichtlich, dass die Berechnung in etwa doppelt so lange laufen müsste, um die lineare Kette im oberen Teil des Baums um nur einen Knoten zu verkürzen. Wenn also die erste, an der Wurzel des Baums gefällte Entscheidung eine Fehlentscheidung war, die zur Unlösbarkeit führt, wären in diesem Fall mehrere Jahre Rechenzeit nötig, um eine Lösung zu finden – und das bei einem vergleichsweise kleinen Datensatz.

Dabei ist die Wahrscheinlichkeit relativ groß, dass die (oberste) Fehlentscheidung, die vom Backtracking gefunden und revidiert werden muss, sich relativ weit oben im Baum befindet: Wenn beispielsweise – wie an vielen Grundschulen üblich – die zweite bis sechste Stunde Kernstunden sind und die jüngeren Schüler in der ersten Klasse auch nur genau so viel Unterricht haben (also 25 Wochenstunden), dann darf der Algorithmus keine einzige Unterrichtsstunde in den Nachmittag oder in die erste Stunde legen.

Mit randomisierten Heuristiken für die Wahl der Werte oder mit der voreingestellten Heuristik, immer die erste Variable und immer den kleinsten Wert zu wählen, geschieht ein solcher Fehler aber fast mit Sicherheit: Die erste Unterrichtsstunde der ersten Klasse (1a) auf

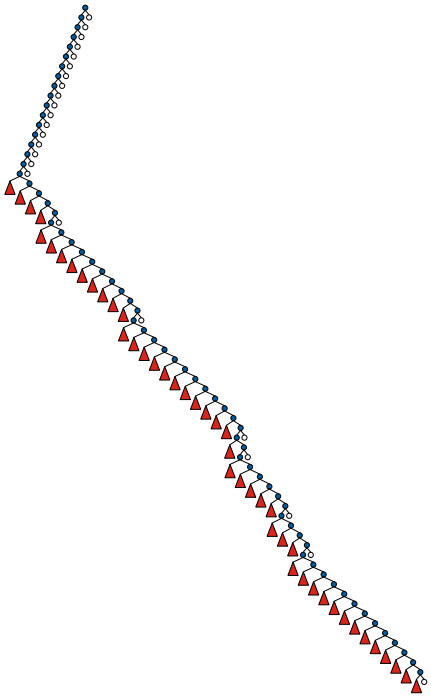

Abbildung 5.1. Suchbaum (Gecode) von Datensatz B mit Kernstunden nach einer Stunde

den ersten verfügbaren Zeitslot (Mo1) zu legen wäre an vielen Grundschulen bereits eine Fehlentscheidung, die zur Unlösbarkeit führen würde, weil Mo1 keine Kernstunde ist und die 1. Klasse nur so viel Unterricht hat, wie es Kernstunden gibt. Danach muss der Solver viel Zeit damit verbringen, die übrigen Unterrichtsstunden konfliktfrei unterzubringen, um dann festzustellen, dass die Kernstunden-Bedingung mangels Unterricht in der ersten Klasse gar nicht mehr erfüllt werden kann. Da aber die letzten Entscheidungen zuerst revidiert werden, probiert der Solver zuerst alle Kombinationen für den restlichen Unterricht durch, bevor er die falsch gesetzte Unterrichtsstunde irgendwann auf eine Kernstunde verschiebt.

5.3.6. Sortierung der Zeitslots

Zwar ist es in der Praxis sicherlich nicht möglich, nur mit Constraint Propagation und ohne Backtracking auszukommen, sobald Bedingungen wie Kernstunden beachtet werden müssen. Und wie Abbildung 5.1 deutlich zeigt, ist selbst bei einem sehr kleinen Datensatz der Suchbaum bereits zu groß, um ihn vollständig mittels Backtracking zu durchsuchen. Trotzdem kann die Backtracking-Methode zielführend sein, wenn es gelingt, die Fehlentscheidungen möglichst weit hinauszuzögern, sodass sie möglichst früh wieder

revidiert werden. Das bedeutet im Umkehrschluss, dass Entscheidungen, die mit hoher Wahrscheinlichkeit richtig sind, zuerst gemacht werden sollten.

Im konkreten Fall der Kernstunden bedeutet das, dass ein Unterricht, der platziert werden soll, bevorzugt auf eine Kernstunde gelegt werden sollte. Die natürliche Ordnung der Zeitslots ist die chronologische Ordnung. Diese Ordnung ist aber für die Lösung des Stundenplanproblems in der bisher vorgestellten Form unbedeutend. Es ist also problemlos möglich, die Zeitslots so zu sortieren, dass Kernstunden am Beginn der Liste stehen. Kombiniert mit der Heuristik „Choose Min Value“ führt das dazu, dass Unterrichtsstunden bevorzugt auf Kernstunden gelegt werden.

Tabelle 5.4. CSP mit Kernstunden und sortierten Zeitslots; unterteilt nach Variablen-Auswahl-Heuristik

Datensatz	A	B	C	D	
Rechenzeit (Or-Tools)	0 ms	3 ms	>1 h	19 ms	Choose First
Failures (Or-Tools)	0	0	-	0	
Rechenzeit (Gecode)	0 ms	2 ms	>1 h	21 ms	
Failures (Gecode)	0	0	-	0	
Rechenzeit (Or-Tools)	0 ms	4 ms	8 ms†	18 ms	Choose Random
Failures (Or-Tools)	0	1‡	9‡	0	
Rechenzeit (Gecode)	0 ms	1 ms	>1 h	14 ms	
Failures (Gecode)	0	0	-	0	

† Nur bei einem Viertel der Versuche, sonst >1 h.
‡ Mittelwert

Obwohl es in manchen Fällen weiterhin zu Konflikten kommen kann, werden die falschen Entscheidungen sehr früh wieder revidiert – auch wenn dies in den Tests nur mit Or-Tools gelang. Beim Datensatz C war außerdem etwas Glück nötig, um eine Lösung zu finden. Wie sich an der Anzahl der Failures, d. h. der Anzahl der erfolglos besuchten Blattknoten zeigt, ist der Suchbaum einer erfolgreichen Suche nun nicht mehr linear – im Vergleich zu allen vorherigen Berechnungen, die entweder gar nicht oder mit 0 Failures terminierten. Durch das Umsortieren der Zeitslots – kombiniert mit einer passenden Heuristik – konnte die Suche also erheblich verbessert werden.

5.3.7. Sortierung der Unterrichte

Zur weiteren Optimierung der Suchreihenfolge könnte man auch die Unterrichte sortieren, sodass nicht nur die Werte sondern auch die Variablen im CSP so sortiert sind, dass die schwierigeren Unterrichte früher platziert werden. Dadurch werden Situationen vermieden, in denen zwei Unterrichte verschiedener Klassen so gelegt werden, dass für eine Kopplung, an der beide Klassen teilnehmen, kein möglicher Zeitslot mehr übrig bleibt. Sinnvoller

ist es, zuerst den Zeitpunkt für die Kopplung festzulegen und danach den individuellen Unterricht zu planen. Um die Unterrichte zu sortieren, kann man eine Ordnung für die Unterrichte definieren, die zum Beispiel von der Anzahl der beteiligten Klassen und Lehrer abhängt, sowie bei Gleichheit von der Anzahl anderer Unterrichte abhängt, an denen die beteiligten Klassen und Lehrer wiederum beteiligt sind.

Dieser Ansatz brachte bei einigen Versuchen allerdings keine weiteren Verbesserungen gegenüber der bloßen Sortierung der Zeitslots, auch nicht in Kombination mit den weiteren noch vorgestellten Bedingungen, sodass an dieser Stelle nicht genauer darauf eingegangen wird.

5.3.8. Modellierung mit Statusvariablen

In Unterabschnitt 5.2.2 wurde eine alternative Art der Modellierung der Kernstunden-Bedingung angesprochen, die für jede Unterricht-Zeitslot-Kombination eine zusätzliche Statusvariable benötigt und dann analog zum linearen Programm mit binären Variablen modelliert werden kann. Tabelle 5.5 zeigt die unterschiedlichen Rechenzeiten, um zu verdeutlichen, dass die konkrete Umsetzung einer Bedingung zwar durchaus mit einem Faktor 2 bis 3 Einfluss auf die Laufzeit nehmen kann (beim größten Datensatz). Dieser ist jedoch verschwindend gering im Vergleich zu den Unterschieden zwischen terminierter Berechnung (unter einer Sekunde) und sehr lang laufender Berechnung (über eine Stunde) – denn auch mit Statusvariablen terminieren die Berechnungen nach einer Stunde nicht, wenn die Zeitslots nicht vorab sortiert werden. Die Wahl der konkreten, effizienten Implementierung eines bestimmten Constraints ist also zunächst zweitrangig. Hinweis: Die Ergebnisse der Berechnungen ohne Zeitslot-Sortierung sind in der Tabelle nicht aufgeführt, weil sie mangels Terminierung keinen Vergleich erlauben.

Tabelle 5.5. Vergleich von Modellierungs-Varianten der Kernstunden-Bedingung

Datensatz	A	B	C	D	
Rechenzeit (Or-Tools)	0 ms	4 ms	8 ms	18 ms	`count`-Constraint
Failures (Or-Tools)	0	1	9	0	
Rechenzeit (Or-Tools)	1 ms	6 ms	10 ms	46 ms	Statusvariablen
Failures (Or-Tools)	0	1	9	0	

5.3.9. Weitere Bedingungen

Unter den in Abschnitt 2.4 vorgestellten Bedingungen sind nur noch wenige, die sich ohne die Verwendung komplexer Constraints umsetzen lassen.

Zum einen sind das die Nichtverfügbarkeiten von Lehrern. Wenn ein Lehrer $l \in L$ in einem Zeitslot $z \in Z$ nicht verfügbar ist, kann keiner seiner Unterrichte zu diesem Zeitpunkt stattfinden, was sich wie folgt formulieren lässt:

$$\forall u \in U_l \; \forall 1 \leqslant i \leqslant W(u) : \quad x_{ui} \neq z$$

Zum anderen ist es die Fixierung von einigen Unterrichtsstunden auf vorgegebene Zeitslots. Für jeden der fixen Zeitslots $\{z_1, \ldots, z_k\} \subset Z$ für einen Unterricht $u \in U$ wird eine der Variablen x_{ui} fixiert:

$$\forall 1 \leqslant i \leqslant k : \quad x_{ui} = z_i$$

Zusätzlich muss beachtet werden, dass die fixierten Zeitslots vom Symmetry Breaking ausgenommen werden müssen, weil sonst der gesamte nicht-fixierte Unterricht nach den fixen Zeitslots stattfinden müsste. Dadurch nimmt die Anzahl der Constraints durch fixierte Unterrichtsstunden um maximal eins zu pro Unterricht. Weil in der Summe aber durch die Vorbelegung einer Variablen mehr Informationen zur Lösung des CSP vorliegen, wird die Probleminstanz durch diese Bedingung eher einfacher.

Die beiden genannten Bedingungen lassen sich als einfache (Un-)Gleichungen implementieren und verbieten lediglich das Setzen von Unterricht an bestimmten Stellen, sodass die Constraint Propagation ausreicht, um die Constraints zu erfüllen. Dementsprechend haben sie keinen negativen Einfluss auf die Laufzeit oder gar den Erfolg der Berechnung:

Tabelle 5.6. CSP mit Kernstunden, Nichtverfügbarkeiten und fixierten Unterrichtsstunden

Datensatz	A	B†	C†	D
Anzahl Variablen	16	114	236	467
Anzahl Constraints (Or-Tools)	24	516	1090	4182
Anzahl Constraints (Gecode)	15	155	299	728
Rechenzeit (Or-Tools)	0 ms	4 ms	>1 h	20 ms
Rechenzeit (Gecode)	0 ms	1 ms	>1 h	21 ms

† Enthält keine fixierten Unterrichtsstunden

Die weiteren Bedingungen wie Doppelstunden, ein freier Tag für ausgewählte Lehrer oder die Begrenzung auf täglich maximal zwei Stunden Unterricht je Fach sind nur mit komplexen Constraints, deren Propagierung mehr Rechenzeit in Anspruch nimmt, und/oder mit einer großen Anzahl von Statusvariablen umsetzbar. Insbesondere für die Erfüllung der weichen Bedingungen, die als gewichteter numerischer Wert in eine Zielfunktion eingehen müssen, sind solche Statusvariablen unverzichtbar.

Beispielhaft wurde die Doppelstunden-Bedingung mit Statusvariablen implementiert. Um einen besseren Vergleich mit dem Suchbaum in Abbildung 5.1 zu erlauben, wurde der dort verwendete Datensatz B um (nur) eine Doppelstunde in der dritten und vierten

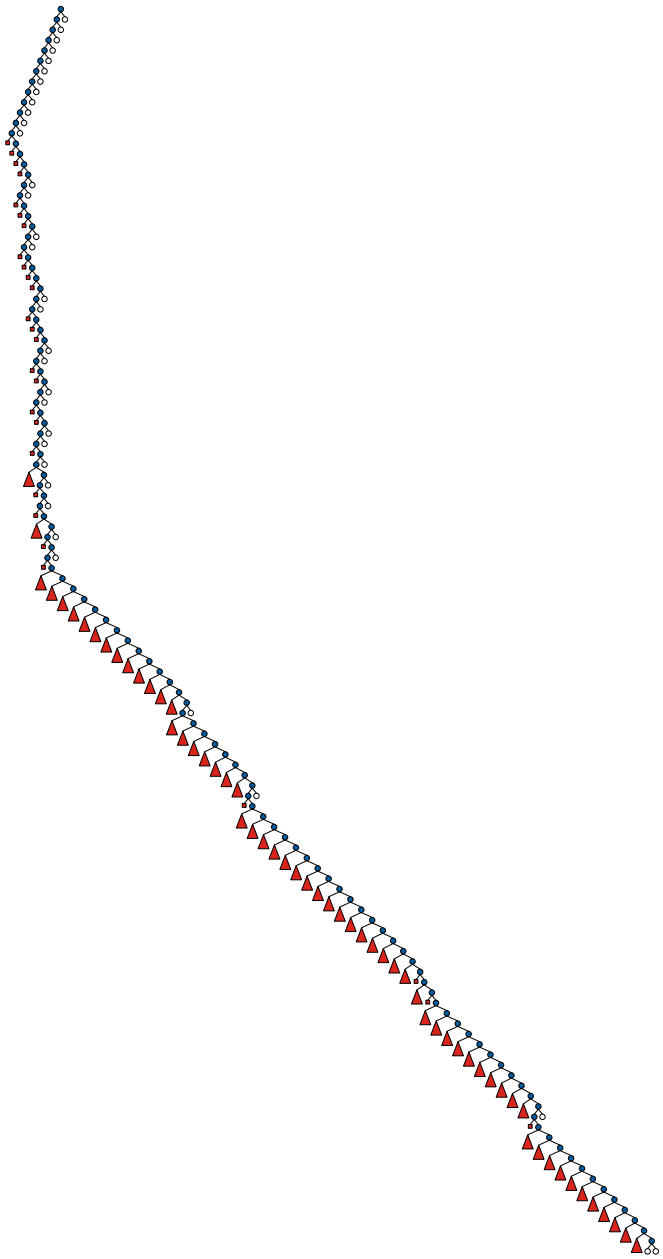

Abbildung 5.2. Suchbaum (Gecode) von Datensatz B mit einer Doppelstunde

Stunde erweitert. Üblicherweise findet das Doppelstundenmodell an Schulen mindestens am Vormittag und oft auch am gesamten Tag Anwendung. Abbildung 5.2 zeigt, dass bereits bei einem einzigen Doppelstundenpaar sehr viele Statusvariablen erstellt werden müssen, die die Tiefe des Suchbaums im Vergleich mit Abbildung 5.1 deutlich erhöhen.

Bei durchgehendem Doppelstundenmodell oder bei der Verwendung anderer (weicher) Bedingungen müsste der Solver also für jede Kombination aus Unterrichtsstunde und Zeitslot eine binäre Statusvariable erstellen, die aussagt, ob die Unterrichtsstunde auf den Zeitslot gelegt wird oder nicht. Es liegt daher nahe, diese binären Variablen direkt zur Grundlage der Modellierung zu machen und die weiteren Constraints darauf aufbauend zu formulieren. Dieser Ansatz wird im nächsten Kapitel vorgestellt.

5.4. Alternative Modellierung

Die oben vorgestellte CSP-Modellierung hat den Vorteil, dass sie intuitiv ist, wenige Variablen verwendet (nur eine pro unterrichteter Wochenstunde), die Bedingung für die korrekte Anzahl Wochenstunden nicht als eigenen Constraint angeben muss und für die Konfliktfreiheit auf den performanten `alldifferent`-Constraint zurückgreifen kann. Dennoch kann es sich lohnen, eine andere Art der Modellierung zu wählen. Zum einen ist nicht nur die Anzahl der Variablen, sondern auch die Größe der Domänen entscheidend für die Zahl der Lösungen, die untersucht werden müssen. Zum anderen kann durch eine Modellierung mit weniger komplexen Constraints dieselbe Probleminstanz parallel als lineares Programm modelliert und dessen Relaxierung zur Verkleinerung der CSP-Instanz und damit des Suchbaums verwendet werden (siehe dazu Kapitel 6). Dementsprechend hat die zweite hier vorgestellte CSP-Modellierung des Stundenplanproblems eine größere Anzahl an Variablen und Constraints, verwendet dafür aber strukturell einfachere Constraints, die sich schneller propagieren und außerdem auch als lineare Gleichungen ausdrücken lassen.

Analog zu der in Unterabschnitt 3.3.1 beschriebenen LP-Formulierung wird nun auch hier pro Unterricht $u \in U$ (*nicht* pro Unterrichtsstunde) und pro Zeitslot $z \in Z$ eine binäre Variable x_{uz} erstellt. Für Details zur Modellierung siehe Unterabschnitt 3.3.1. Der Vorteil dieser Art der Modellierung ist, dass es keine symmetrischen Lösungen mehr in Bezug auf verschiedene Unterrichtsstunden zu einem bestimmten Unterricht gibt, weil die Variablen pro Unterricht und nicht pro Unterrichtsstunde erstellt werden. Dafür muss pro Unterricht ein zusätzlicher Constraint eingefügt werden, der die richtige Anzahl an Wochenstunden erzwingt. Die jeweiligen binären Ungleichungen zu allen Bedingungen werden von den Solvern direkt verstanden und in effiziente Propagatoren umgesetzt, die für binäre Variablen optimiert sind.

5.4.1. Nur Konfliktfreiheit

Genau wie bei der in Abschnitt 5.2 vorgestellten Variante gelingt es auch hier bei allen Datensätzen, innerhalb kürzester Zeit eine Lösung zu finden, solange nur die Konfliktfreiheit gefordert ist. Auffällig ist, dass der schwierigste Datensatz C nur mittels Backtracking gelöst werden kann.

Tabelle 5.7. Binäres CSP nur mit Konfliktfreiheits-Bedingung

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Anzahl Constraints	103	472	677	2901
Rechenzeit (Or-Tools)	0 ms	1 ms	115 ms	8 ms
Rechenzeit (Gecode)	0 ms	7 ms	7,0 s	210 ms
Failures (beide)	0	0	9576	0

Als Heuristik wurde „Choose First Variable“ und „Choose Max Value“ gewählt, was dazu führt, dass die Unterrichte der Reihe nach auf den ersten möglichen Zeitslot fixiert werden, da die erste mögliche Variable auf ihren maximalen Wert (also 1) gesetzt wird.

5.4.2. Mit Kernstunden

Die Forderung nach Kernstunden lässt sich mit binären Variablen besonders einfach ausdrücken, indem der Constraint für die Konfliktfreiheit einer Klasse abgewandelt wird: Anstatt höchstens einer ($\leqslant 1$) Unterrichtsstunde zu einem bestimmten Zeitpunkt ist nun genau eine ($= 1$) gefordert. Dementsprechend bleibt die Anzahl der Constraints gleich, wenngleich die Schwierigkeit der Probleminstanz natürlich stark zunimmt.

Tabelle 5.8. Binäres CSP mit Kernstunden

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Anzahl Constraints	103	472	677	2901
Rechenzeit (Or-Tools)	0 ms[†]	>1 h	144 ms[‡]	>1 h
Rechenzeit (Gecode)	0 ms[†]	6 ms	>1 h	>1 h

[†] Der Datensatz A enthält keine Kernstunden
[‡] Nach 14958 Failures

Im Vergleich zur ersten Modellierungs-Variante (siehe Tabelle 5.3) fällt auf, dass nicht alle Datensätze unlösbar sind, sondern dass Or-Tools und Gecode jeweils einen (Zufalls-)Treffer landen. Das ist möglich, wenn die Variablen so angeordnet sind, dass ganz ohne Backtracking bzw. mit nur wenigen Failures eine Lösung gefunden werden kann. Es sei darauf hingewiesen, dass die CSP-Suche *nicht* randomisiert ist, sondern lediglich die Variablen

und Constraints aus implementierungstechnischen Gründen in leicht unterschiedlichen Reihenfolgen an die Solver übergeben wurden.

Da aber der Großteil der Berechnungen nicht erfolgreich verlief, muss auch hier eine Möglichkeit zur Verkleinerung und/oder Anordnung des Suchbaums gefunden werden, damit die Backtracking-Suche sinnvoll eingesetzt werden kann.

5.4.3. Sortierung der Zeitslots

Analog zu dem in Unterabschnitt 5.3.6 gezeigten Ansatz können auch hier die Zeitslots so sortiert werden, dass die Variablen x_{uz}, die zu einer Kernstunde $z \in Z_k$ gehören, zuerst abgearbeitet werden. In Kombination mit der Werteheuristik „Choose Max Value“ wird der Unterricht dann bevorzugt auf diese Zeitslots gelegt.

Tabelle 5.9. Binäres CSP mit Kernstunden und sortierten Zeitslots

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Anzahl Constraints	103	472	677	2901
Rechenzeit (Or-Tools)	0 ms[†]	>1 h	>1 h	>1 h
Rechenzeit (Gecode)	0 ms[†]	6 ms	>1 h	>1 h

[†] Der Datensatz A enthält keine Kernstunden

Leider verfehlt diese Optimierung bei der binären Modellierungs-Variante ihre Wirkung. Durch die Sortierung der Zeitslots können keine zusätzlichen Datensätze gelöst werden, stattdessen ist nun auch Or-Tools (zufällig) nicht mehr in der Lage, den Datensatz C zu lösen.

Da alle erzielten Lösungen von binären Probleminstanzen mit Kernstunden eher als Zufallstreffer einzustufen sind, bei denen die Variablen zufällig in der richtigen Reihenfolge waren, scheint eine Umsortierung der binären Variablen nicht auszureichen. Stattdessen muss das ursprüngliche Problem, dass der Suchbaum zu groß ist, gelöst werden.

Eine naheliegende Möglichkeit, um den Lösungsraum und damit den Suchbaum zu verkleinern, ist das Fixieren von Variablen auf hoffentlich sinnvolle Werte. Anstatt die zu fixierenden Werte heuristisch auszuwählen, wird ein anderer Ansatz verfolgt, der auch die Motivation dafür war, das Problem überhaupt binär zu modellieren. Im folgenden Kapitel wird eine Technik vorgestellt, mit der die LP-Relaxierung einbezogen werden kann, um Werte vorab zu fixieren.

Kapitel 6

Einbezug der LP-Relaxierung

Der in diesem Kapitel beschriebene Lösungsansatz kombiniert den binären Ansatz aus Abschnitt 5.4 mit der LP-Formulierung aus Abschnitt 3.3. Zunächst wird in Abschnitt 6.1 dargelegt, wieso diese Methode erfolgsversprechend ist. Die Abschnitte 6.3, 6.4 und 6.5 stellen drei verschiedene Möglichkeiten vor, wie die Kombination der beiden Modellierungen sinnvoll vorgenommen werden kann, und analysieren die Ergebnisse. Abschnitt 6.6 zeigt eine Methode, mit der aus dem Konfliktgraphen der Probleminstanz zusätzliche Constraints gewonnen werden können, um die LP-Formulierung zu verschärfen. Abschließend wird in Abschnitt 6.7 untersucht, wie sich die zuvor entwickelten Lösungsansätze verhalten, wenn weitere Bedingungen eingehalten werden sollen.

6.1. Motivation

In Abschnitt 5.4 wurde eine Modellierung des Stundenplanproblems als Constraint Satisfaction Problem vorgestellt, die strukturell der Umsetzung von Weidler [2012] als ganzzahliges lineares Programm (ILP) gleicht. Gibt man die Forderung nach der Ganzzahligkeit in diesem linearen Programm auf, so erhält man eine relaxierte Form der Probleminstanz. Die Bedingung $x_{uz} \in \{0, 1\}$ wird also ersetzt durch $0 \leqslant x_{uz} \leqslant 1$. Im Unterschied zum ursprünglichen, NP-schweren ILP ist das zugehörige relaxierte Problem in Polynomialzeit lösbar.

Ob es überhaupt zielführend ist, die LP-Relaxierung zu verwenden, lässt sich durch einen Vergleich der Zielfunktionswerte der optimalen ganzzahligen und der optimalen reellen Lösungen untersuchen. Außerdem ist es von Interesse, wie ähnlich sich die Belegungen der beiden Lösungen sind. Die Werte wurden dabei mit Gurobi und der SchulScheduler-Modellierung (siehe auch Kapitel 4) ermittelt, wobei die Zielfunktion leicht perturbiert wurde, um stets die gleiche aus mehreren optimalen Lösungen zu erhalten.

Tabelle 6.1. Zielfunktionswerte von LP und ILP (berechnet mit Gurobi)

Datensatz	A	B	C	D
OPT_{INT}	-8,000·10^1	-2,715·10^3	-2,372·10^3	-1,652·10^4 †
OPT_{FRAC}	-8,200·10^1	-2,715·10^3	-2,372·10^3	-1.812·10^4
Anteil gleicher Werte	90,2 %	82,5 %	71,7 %	84,3 %
Anteil ähnlicher Werte	96,5 %	95,9 %	94,8 %	97,5 %
Anteil geänderter Werte	3,5 %	4,1 %	5,2 %	2,5 %
Rechenzeit für Relaxierung	6 ms	91 ms	119 ms	4,0 s

† Nicht optimal. Die Berechnung würde über Tage laufen und wurde deshalb nach 40 Minuten bei einer verbleibenden Gap von 0,19 % abgebrochen.

Tabelle 6.1 zeigt, dass die Zielfunktionswerte stets von der gleichen Größenordnung und in den meisten Fällen sogar bis auf mindestens drei Nachkommastellen gleich sind. Auch inhaltlich sind sich die Lösungen sehr ähnlich: Der Großteil der Variablen (zwischen 70 % und 90 %) ist bereits in der Lösung der LP-Relaxierung ganzzahlig („gleich“ in der Tabelle) und etwa 95 % der Variablen wurden so gerundet wie erwartet (also ab $0,5$ zu 1 aufgerundet; „ähnlich“ in der Tabelle).

Weiterhin lohnt es sich nur, die LP-Relaxierung mit einzubeziehen, wenn diese in akzeptabler Zeit berechnet werden kann. Dem kommerziellen Solver Gurobi, der auch die ganzzahligen Probleminstanzen lösen kann, gelingt das in kürzester Zeit. Ein kurzer Versuch (ohne genaue Messung) mit den offenen LP-Solvern glpk, lp_solve, und SYMPHONY (Clp) zeigte, dass auch diese in unter einer Minute die optimale reelle Lösung bestimmen können.

6.2. Mögliche Vorgehensweisen

Um von einer reellen Lösung auf eine mögliche (und im Fall eines Optimierungsproblems eine möglichst gute) Lösung des ursprünglichen Problems zu schließen, gibt es mehrere verschiedene Ansätze. Eine Möglichkeit ist, gezielt so zu runden, dass dabei zumindest keine Bedingungen verletzt werden und dass der Wert der Zielfunktion sich nicht zu stark verschlechtert. Zum Beispiel sind die Werte der LP-Relaxierung des Vertex-Cover-Problems stets halbzahlig (also 0 oder $0,5$ oder 1) und man erhält eine 2-Approximation, indem man alle Werte von $0,5$ auf 1 rundet. Im Unterschied zum Stundenplanproblem hat das Vertex-Cover-Problem jedoch die Eigenschaft, dass die extreme Lösung, die alle Variablen auf 1 setzt, immer noch zulässig ist. Beim Stundenplanproblem hingegen ist es in der Regel weder möglich, alle Variablen auf 0 zu setzen, noch alle Variablen auf 1 zu setzen. Das

bedeutet, dass bei unvorsichtigem Runden leicht Bedingungen verletzt werden können, sodass die Lösung unzulässig wird.

Der hier vorgestellte Ansatz ist allerdings überhaupt nicht darauf angewiesen, dass jede Variable einen ganzzahligen Wert direkt aus der LP-Relaxierung zugewiesen bekommt. Im Gegenteil ist es sogar gewünscht, dass einige Variablen unbesetzt bleiben, da die LP-Relaxierung nur dazu dienen soll, diejenigen Variablen zu fixieren, deren Werte als vergleichsweise sicher angenommen werden können. Es wäre also ein Vorgehen denkbar, bei dem alle Werte unterhalb von (zum Beispiel) $0,1$ abgerundet, alle Werte oberhalb von $0,9$ aufgerundet und die übrigen vom CSP-Solver bestimmt werden.

6.3. Ganzzahlige Werte fixieren

Noch vorsichtiger ist der Ansatz, überhaupt nicht zu runden, sondern nur Werte, die bereits in der Lösung der LP-Relaxierung ganzzahlig sind (also $0,0$ und $1,0$), zu fixieren. Leider zeigen aber die Ergebnisse, dass selbst dieser Ansatz bereits zu weit geht, sodass einige Datensätze unlösbar werden:

Tabelle 6.2. Binäres CSP mit Fixieren von ganzzahligen Werten aus der LP-Relaxierung

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Anzahl Constraints	103	472	677	2901
Rechenzeit (Relaxierung)	3 ms	14 ms	26 ms	258 ms
Anzahl fixierter 1-Werte	5	114	194	153
Anzahl fixierter 0-Werte	129	922	1234	7750
Gesamtzahl Constraints	237	1508	2105	10804
Rechenzeit (Or-Tools)	0 ms	1 ms	unlösbar	unlösbar

Hinweis: Die Berechnung basiert auf der Modellierung aus Abschnitt 5.4 mit Konfliktfreiheit, Zeitslot-Sortierung und Kernstunden. Für jeden zu fixierenden Wert wurde ein zusätzlicher Constraint der Form $x_{uz} = 1$ bzw. $x_{uz} = 0$ eingefügt.

6.3.1. Unlösbare Probleminstanzen beim Fixieren von Nullen

Zwar sinkt die verbleibende Rechenzeit für Or-Tools bei den Datensätzen A und B deutlich ab, was zeigt, dass durch den Einbezug der LP-Relaxierung die Probleminstanz einfacher geworden ist. Aber bei den anderen beiden Datensätzen wird die Probleminstanz durch das Fixieren der 0- und 1-Werte unlösbar.

Einerseits um sicher zu gehen, andererseits zur Untersuchung der Ursachen, wurde die Unlösbarkeit auch mit Gurobi und einem ganzzahligen linearen Programm der gleichen Struktur nachvollzogen. Neben der bloßen Feststellung der Unlösbarkeit ist Gurobi in der Lage, eine minimale Teilmenge der Constraints zu bestimmen, die zur Unlösbarkeit führt. Durch systematisches Analysieren der Zusammenhänge zwischen diesen Constraints (siehe Abbildung 6.1) und durch iteratives Vereinfachen der Probleminstanzen konnte schließlich eine minimale Probleminstanz gefunden werden, an der das Problem demonstriert werden kann. Das Problem entsteht durch eine ungünstige Kombination aus Kopplungen (siehe Abschnitt 4.3) und Kernstunden.

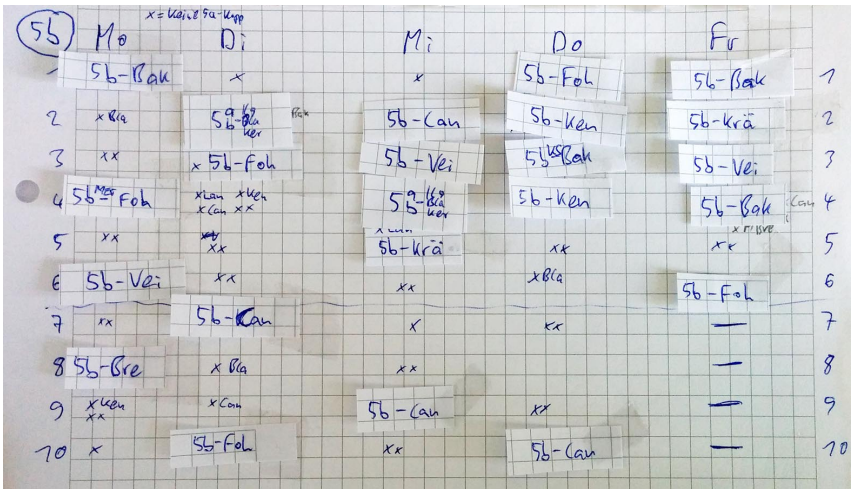

Abbildung 6.1. Mit Papier, Stift und Klebeband verschaffte sich der Autor einen Eindruck davon, wie aufwändig die manuelle Stundenplanung in der Praxis sein muss.

Die minimale Probleminstanz besteht aus zwei Klassen A und B, drei Fächern F, G und H, die von jeweils einem Lehrer unterrichtet werden (diese heißen zur Vereinfachung ebenfalls F, G und H), sowie drei Zeitslots 1, 2, und 3 (insgesamt, nicht pro Tag), die alle drei Kernstunden sind. Jede der Klassen wird in jedem der Fächer genau eine Stunde lang unterrichtet. Dadurch hat jede Klasse genau drei Stunden Unterricht, die Kernstunden-Bedingung lässt sich also erfüllen. Außerdem findet der Unterricht im Fach H gemeinsam, also gekoppelt statt. Während die Lehrer F und G also zwei Stunden pro Woche arbeiten müssen, hält der Lehrer H seinen Unterricht nur ein Mal.

Neben der Konfliktfreiheit, den Kernstunden und natürlich der Erfüllung des geforderten Unterrichts gibt es keine weiteren Bedingungen. Dass die Probleminstanz prinzipiell lösbar ist, lässt sich am einfachsten durch Angabe von zulässigen Stundenplänen zeigen – hier aus Sicht der beiden Klassen:

Tabelle 6.3. Mögliche Lösung für die minimale Probleminstanz

Zeit	Unterricht A	Unterricht B
1	F	G
2	G	F
3	H	H

Wie man leicht nachprüfen kann, muss kein Lehrer zwei Unterrichte gleichzeitig halten, jede Klasse erhält ihren geforderten Unterricht und die Unterrichtsstunde im Fach *H* findet gemeinsam statt.

Interessanterweise liefert Gurobi bereits in der Presolve-Phase diese Lösung. Im Normalfall ist diese Probleminstanz aber in ein größeres Umfeld von weiteren Constraints eingebettet, die dazu führen, dass der LP-Solver unter Umständen eine andere Lösung zurückgibt. Prinzipiell muss die Vorgehensweise zum Einbeziehen der LP-Relaxierung mit jeder zulässigen relaxierten Lösung zurechtkommen. Wenn man im gegebenen Beispiel drei weitere Zeitslots 4, 5 und 6 hinzufügt, die jedoch keine Kernstunden sind, ändert das zunächst nichts an der Zulässigkeit der gegebenen Lösung – die überflüssigen Zeitslots bleiben einfach für alle Klassen unterrichtsfrei. Die zusätzlichen Zeitslots führen jedoch dazu, dass Gurobi eine andere Lösung für die LP-Relaxierung berechnet.

Wir gehen also davon aus, dass aufgrund der Struktur des Problems „um“ diese minimale Probleminstanz herum der LP-Solver eine beliebig ungünstige, aber dennoch zulässige reelle Lösung liefern kann. In diesem Fall sei die Lösung wie folgt aufgebaut:

Tabelle 6.4. Werte für x_{uz} aus der LP-Relaxierung

z \ u	A-F	A-G	B-F	B-G	A-B-H
1	1/2	0	1/2	0	1/2
2	1/2	1/2	1/2	1/2	0
3	0	1/2	0	1/2	1/2

Der Wert x_{uz} gibt an, ob der Unterricht $u \in U$ zum Zeitpunkt $z \in Z$ stattfindet. Beispielsweise bedeutet die 0 unten links in der Tabelle, dass der Unterricht *A-F* sicher nicht zum Zeitpunkt 3 stattfindet.

Wir überprüfen nun, ob alle Bedingungen eingehalten wurden:

- Von jedem Unterricht findet wie gefordert insgesamt eine Stunde statt, da die Spaltensummen 1 sind.
- Es gibt keine Konflikte für Klassen oder Lehrer, da die Summe der zu einer Klasse bzw. einem Lehrer gehörenden Werte in einer Zeile nie größer als 1 ist.
- Die Kernstunden-Bedingung wird eingehalten, weil die Summe der zu einer Klasse gehörenden Werte in einer Zeile immer genau 1 ist.

Das Fixieren betrifft in diesem Fall nur die 0-Werte, weil es keine 1-Stellen gibt. Wir zeigen nun, dass es für die übrigen Werte keine zulässige ganzzahlige Lösung mehr geben kann: Für den gekoppelten Unterricht *H* muss entweder Zeitslot 1 oder Zeitslot 3 gewählt werden. Wenn Zeitslot 1 gewählt wird, gilt für beide Klassen, dass sie in diesem Zeitslot keine Zeit mehr haben. Deswegen *muss* der Unterricht im Fach *F* bei *beiden* Klassen zum Zeitpunkt 2 stattfinden, denn zum Zeitpunkt 3 ist er verboten, weil die LP-Relaxierung hier den Wert 0 zugewiesen hat. Da aber beide Klassen im Fach *F* vom selben Lehrer *F* unterrichtet werden, können diese Unterrichte nicht zeitgleich stattfinden, sodass es keine Lösung gibt. Ein analoger Widerspruch ergibt sich mit Lehrer *G*, wenn man annimmt, dass der Unterricht *H* zum Zeitpunkt 3 stattfindet.

Zwar entsteht dieser Konflikt nur durch die Kombination aus Kopplungen mit Kernstunden und einer ungünstig gewählten reellen Lösung, doch in der Praxis kommt dieser Fall sehr häufig vor. Bei Tests mit verschiedenen LP-Solvern, verschiedenen Algorithmen in Gurobi (Innere-Punkte-Verfahren und Dual-Simplex) und verschiedenen (randomisierten) Reihenfolgen der Variablen und Constraints ergaben sich jeweils unterschiedliche Lösungen für die LP-Relaxierung, von denen etwa zwei Drittel nach dem Fixieren des ganzzahligen Anteils zu unlösbaren Probleminstanzen führten. Auch wenn mit einer Zielfunktion, die sich aus den weichen Bedingungen der Problemstellung und einer leichten Perturbation ergibt, eine eindeutige optimale Lösung für die LP-Relaxierung gefunden werden kann, löst diese das Problem für die Datensätze C und D leider nicht auf.

6.4. Nur Einsen fixieren

Im oben gezeigten Beispiel und auch in allen anderen untersuchten (größeren) Fällen entsteht das Problem durch die Fixierung der 0-Werte, die effektiv die Platzierung eines Unterrichts auf einen bestimmten Zeitslot verbieten. In größeren Probleminstanzen ist stets eine Kopplung mit im Spiel, sowie eine großflächige Zahl von „Verboten“ für einen bestimmten Unterricht. Abbildung 6.2 schlüsselt für einen bestimmten, beispielhaften Fall auf, aus welchen Gründen die zwei verbleibenden Stunden eines Unterrichts nicht platziert

Stunde	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
1	K	K	K	0	L
2	L	K	0	0	K
3	L	K	L	0	K
4	L	L	K	0	K
5	0	L	L	0	0
6	K, L	0	0	L	L
7	0	0	K	L	G
8	0	L	0	0	G
9	0		0	L	G
10	K	0	0	L	G

Abbildung 6.2. Analyse einer unlösbaren Probleminstanz (K = Konflikt mit anderem Unterricht der Klasse, L = Konflikt mit anderem Unterricht des Lehrers, G = Zeitslot gesperrt, 0 = Verboten durch LP-Relaxierung)

werden konnten. Für beide bleibt nur ein einziger Zeitslot (Di9) übrig, weil knapp die Hälfte der übrigen Zeitslots durch die 0-Werte aus der LP-Relaxierung blockiert sind.

Daher liegt die Idee nahe, auf das Fixieren der 0-Werte zu verzichten und nur die Einsen zu betrachten. Die Idee erscheint nicht zuletzt deshalb plausibel, weil das sichere Stattfinden eines Unterrichts eine deutlich stärkere Information ist als das Nicht-Stattfinden. Denn in jeder der vielen Gleichungen der Form $x_1 + \cdots + x_j \leqslant 1$, aus denen das lineare Programm zum größten Teil besteht, kann nur eine einzige Variable den Wert 1 annehmen, während der Wert 0 leicht mehrfach zu vergeben ist. Das zeigt sich auch an den Statistiken in Tabelle 6.2, wo erheblich mehr 0-Werte fixiert wurden als 1-Werte.

Bei der Durchführung der Berechnungen ohne die Fixierung von 0-Werten, d. h. nur mit fixierten 1-Werten, scheint das Problem der Unlösbarkeit zunächst völlig verschwunden zu sein:

Tabelle 6.5. Binäres CSP mit Fixieren von 1-Werten

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Anzahl Constraints	103	472	677	2901
Rechenzeit (Relaxierung)	3 ms	14 ms	26 ms	258 ms
Anzahl fixierter 1-Werte	5	114	194	153
Gesamtzahl Constraints	108	586	871	3054
Rechenzeit (Or-Tools)	0 ms	2 ms	2 ms	>1 h
Rechenzeit (Gecode)	0 ms	0 ms	>1 h	697 ms[†]

[†] Nach 116 Failures

Die in Tabelle 6.5 aufgeführten Messungen basieren auf den von Gurobi (deterministisch) ausgewählten optimalen Lösungen für die LP-Relaxierung. Zwar wird die reelle Lösung jeweils deterministisch ausgewählt, aber es handelt sich dennoch um eine willkürliche Auswahl aus einer Menge möglicher Lösungen. Um zu untersuchen, inwiefern die Ergebnisse auf Zufallstreffer zurückzuführen sind oder nicht, wurden die Berechnungen mit unterschiedlichen Lösungen wiederholt. Zur Generierung dieser zusätzlichen Lösungen wurde eine zufällig ausgewählte Variable geändert und fixiert, um die Lösung zunächst unzulässig zu machen, und anschließend wieder eine optimale Lösung gesucht. Dieses Verfahren wurde für jeden Datensatz insgesamt 100 Mal durchgeführt und anschließend ermittelt, ob die durch das Fixieren zustande gekommene Probleminstanz noch lösbar ist, sowie ob sie dann auch tatsächlich gelöst wird. Die Ergebnisse sind in Tabelle 6.6 dargestellt.

Tabelle 6.6. Einfluss der fixierten 1-Werte auf die Lösbarkeit

Datensatz	A	B	C	D
Anzahl lösbarer Instanzen	100	100	80	100
Anzahl unlösbarer Instanzen	0	0	20	0
Anzahl gelöster Instanzen (Or-Tools)	100	100	48	0
Anzahl gelöster Instanzen (Gecode)	100	100	50	16
Anzahl Timeouts >10 s (Or-Tools)	0	0	32	100
Anzahl Timeouts >10 s (Gecode)	0	0	30	84
Anzahl erkannter Unlösbarkeiten (Or-Tools)	-	-	10	-
Anzahl erkannter Unlösbarkeiten (Gecode)	-	-	8	-
Anzahl Variablen	156	1036	1512	8602
Durchschnittlich fixierte 1-Werte (unlösbar)	-	-	204,6	-
Durchschnittlich fixierte 1-Werte (lösbar)	5,1	112,3	185,5	164,5
Durchschnittlich fixierte 1-Werte (gelöst)	5,1	112,3	192,9	166,7
Durchschnittlich fixierte 1-Werte (Timeout)	-	-	173,8	164,4

Zunächst fällt auf, dass die Probleminstanz selbst dann unlösbar werden kann, wenn ausschließlich die 1-Werte fixiert werden. Das ist allerdings äußerst selten und tritt nur beim Datensatz C auf – die Gründe dafür werden in Unterabschnitt 6.4.1 genauer erläutert.

Von den lösbaren Instanzen konnten dennoch nicht alle gelöst werden, weil dem CSP-Solver zu wenig Information zur Verfügung stand, um schnell eine Lösung zu finden. Da die früheren Messungen gezeigt haben, dass eine Lösung entweder sehr schnell (innerhalb von weniger als einer Sekunde) oder erst nach mehreren Stunden gefunden wird, wurde die Berechnung nach 10 Sekunden abgebrochen und die Berechnung in diesem Fall als erfolglos gezählt (Timeout). Während die Datensätze A und B immer gelöst werden konnten, was im Fall von Datensatz B ohne LP-Relaxierung nicht gelingt, wurde für Datensatz D auch

mit LP-Relaxierung nur äußerst selten eine Lösung gefunden. Außerdem zeigt sich, dass mehr fixierte Werte im Mittel eher dazu beitragen, dass die Instanz gelöst werden kann.

Hierbei bleibt allerdings unklar, wie weit sich die nach dem Fixieren gefundene Lösung vom Optimum entfernt hat. Auf der anderen Seite ist es gerade einer der Vorteile der LP-Relaxierung, dass damit auf vergleichsweise kostengünstige Weise die Zielfunktion mit eingebunden werden kann. Die gerundeten/fixierten Werte gehören zu einer optimalen reellen Lösung und sind daher schon relativ dicht am Ziel. Dadurch entfallen viele der aufwändigen Iterationen, die ein CSP-Solver zum Optimieren durchführen muss (siehe Unterabschnitt 5.1.3) – sofern die Probleminstanz nicht bereits unlösbar geworden ist.

Zur Einschätzung der Anzahl der fixierten Variablen sei angemerkt, dass ein fixierter 1-Wert eine ganze Reihe von Variablen indirekt festlegt. Beispielsweise fallen durch die Fixierung einer Unterrichtsstunde automatisch alle anderen Unterrichte der gleichen Klasse bzw. des gleichen Lehrers zu diesem Zeitpunkt weg. Und wenn durch die LP-Relaxierung alle Stunden eines Unterrichts fixiert werden, kann dieser in allen übrigen Zeitslots nicht mehr stattfinden, sodass fast $|Z|$ Variablen durch eine einzige 1 festgelegt werden. So wurden zum Beispiel im Fall von Datensatz D (siehe Tabelle 6.5) von den insgesamt 8602 Variablen nur 153 auf 1 fixiert. Bei der Berechnung mit Gecode mussten dennoch nur 306 weitere Entscheidungen zur Belegung von Variablen getroffen werden – alle weiteren Werte konnten durch Constraint Propagation hergeleitet werden, die insgesamt 5887 Mal aktiv war, um die verbleibenden 8143 Variablen zu belegen.

6.4.1. Unlösbare Probleminstanzen beim Fixieren von Einsen

Die Ergebnisse in Tabelle 6.6 zeigen, dass es im Fall von Datensatz C in einem Fünftel der Fälle zur Unlösbarkeit kommt, wenn die 1-Werte aus der LP-Relaxierung fixiert werden. Mit einem ähnlichen Vorgehen wie in Unterabschnitt 6.3.1 wurde durch manuelle Analyse der Ursachen eine minimale Probleminstanz ermittelt, an der sich das Problem demonstrieren lässt.

Genau wie im obigen Beispiel gibt es auch hier zwei Klassen A und B, die in den drei Fächern F, G und H jeweils eine Unterrichtsstunde pro Woche haben, wobei das Fach H gemeinsam (also gekoppelt) unterrichtet wird. Für die Fächer G und H gibt es weiterhin jeweils einen Lehrer. Im Fach F werden die beiden Klassen jedoch – im Unterschied zum obigen Beispiel – von jeweils einem eigenen Lehrer unterrichtet, die mit F_A und F_B bezeichnet werden. Es stehen weiterhin drei Zeitslots zur Verfügung.

Da die Arbeitsteilung im Fach F, das nun von zwei Lehrern übernommen wird, keine zusätzliche Einschränkung darstellt, ist die oben angegebene Lösung weiterhin zulässig:

Tabelle 6.7. Mögliche Lösung für die minimale Probleminstanz beim Fixieren von Einsen

Zeit	Unterricht A	Unterricht B
1	F_A	G
2	G	F_B
3	H	H

Analog kann man nun nachprüfen, dass keine Konflikte auftreten und dass die geforderten Unterrichte alle stattfinden. Hinweis: Auch wenn sie es in dieser Lösung nicht tun, *könnten* F_A und F_B zeitgleich stattfinden.

Zwar berechnet Gurobi für die LP-Relaxierung genau diese (ganzzahlige!) Lösung, aber es sind weitere reelle Lösungen denkbar, die zu Problemen führen. Wenn man zu den drei Zeitslots zwei weitere hinzufügt, die überhaupt nicht benötigt werden und nicht belegt werden können, weil sie im Vergleich zu den anderen keine Kernstunden sind, dann liefert Gurobi – obwohl die obige Lösung weiterhin zulässig wäre – die folgende halbzahlige Lösung:

Tabelle 6.8. Werte für x_{uz} aus der LP-Relaxierung beim Fixieren von Einsen

z \ u	A-F_A	A-G	B-F_B	B-G	A-B-H
1	1	0	1	0	0
2	0	$1/2$	0	$1/2$	$1/2$
3	0	$1/2$	0	$1/2$	$1/2$

Zunächst sei angemerkt, dass die beiden Einser im Fach *F* aus Sicht des CSP-Solvers die einzigen beiden vorbelegten Werte sind, die übrigen werden verworfen – auch wenn sie natürlich die Bedingungen zur Konfliktfreiheit und zu den Kernstunden erfüllen müssen. Dass es ausgehend von dieser Situation keine Lösung mehr geben kann, sieht man leicht, indem man zunächst die Kopplung *A-B-H* auf den Zeitslot 2 oder 3 platziert. Da daran beide Klassen beteiligt sind, muss der gesamte restliche Unterricht im jeweils anderen Zeitslot stattfinden. Das ist aber nicht möglich, weil die beiden Klassen im Fach *G* beide vom selben Lehrer, aber getrennt unterrichtet werden.

Im Gegensatz zu dem Beispiel in Unterabschnitt 6.3.1, wo durch eine große Anzahl von 0-Werten jede einzelne Lösung blockiert wurde, die ansonsten denkbar gewesen wäre, genügen hier nur zwei Werte, um die Probleminstanz unlösbar zu machen. Glücklicherweise funktioniert das nur, weil das Zeitraster sehr eng ist, während in Unterabschnitt 6.3.1 auch ein erheblich größeres Zeitraster mit Nullen blockiert worden wäre. Sowohl in diesem

Beispiel als auch in Datensatz C gibt es Klassen, die genau so viel Unterricht haben wie es Zeitslots bzw. Kernstunden gibt, sodass keine Ausweichmöglichkeiten existieren und von Anfang an feststeht, auf welche Zeitslots dieser Unterricht aufgeteilt werden muss.

In der Praxis sind solche Fälle zum Glück eher selten, wie auch die Tabelle 6.6 zeigt, und treten nur auf, wenn eine Schule keinen Nachmittagsunterricht vorsieht. In diesen Fällen könnte man bewusst auf den Einsatz der LP-Relaxierung verzichten und die vergleichsweise kleine Probleminstanz dann nur vom CSP-Solver lösen lassen. Für alle anderen Fälle ist der Einbezug der LP-Relaxierung eine gute Lösung, solange man nur die 1-Werte fixiert.

Die Unlösbarkeit zu erkennen und erst dann auf die LP-Relaxierung zu verzichten ist leider nicht möglich, weil die CSP-Solver die Unlösbarkeit nur in der Hälfte der Fälle früh genug melden (vgl. Tabelle 6.6). Stattdessen kann aber die LP-Relaxierung selbst verbessert werden, sodass das Problem nicht mehr auftritt (siehe dazu Abschnitt 6.6).

6.4.2. Runden statt Fixieren

Eine Überlegung zur weiteren Verbesserung des Verfahrens könnte sein, nicht nur ganzzahlige Variablenwerte zu fixieren, sondern sogar aufzurunden, beispielsweise ab $0,9$ oder $0,8$.

Abbildung 6.3 auf Seite 46 zeigt, dass Werte oberhalb von $0,8$ relativ selten sind, vor allem im Vergleich zur Zahl der Variablen, die genau $1,0$ sind. Versuche mit den beiden großen Datensätzen C und D haben ergeben, dass das Aufrunden keine zusätzlichen Vorteile bringt. Stattdessen kam es in einem Fall vor, dass beim Aufrunden ab $0,8$ die Probleminstanz sogar unlösbar wurde, während sie ohne Aufrunden noch gelöst werden könnte.

6.5. Sortieren statt Fixieren

Das in Abschnitt 6.4 vorgestellte Vorgehen, bei dem nur die Einsen fixiert werden, hat zwei prinzipbedingte Nachteile: Zum einen besteht eine (geringe) Gefahr, dass durch das Fixieren die Probleminstanz unlösbar wird und der CSP-Solver das nicht bemerkt. Zum anderen geht die differenzierte Abstufung der übrigen Variablen $0 < x_{uz} < 1$ verloren, obwohl es sich dabei um etwa die gleiche Anzahl an Variablen handelt wie beim Fixieren der $x_{uz} = 1$.

Beide Probleme lassen sich lösen, indem auf das Fixieren verzichtet und stattdessen lediglich priorisiert wird. Die LP-Relaxierung wird also nicht mehr als Einschränkung des

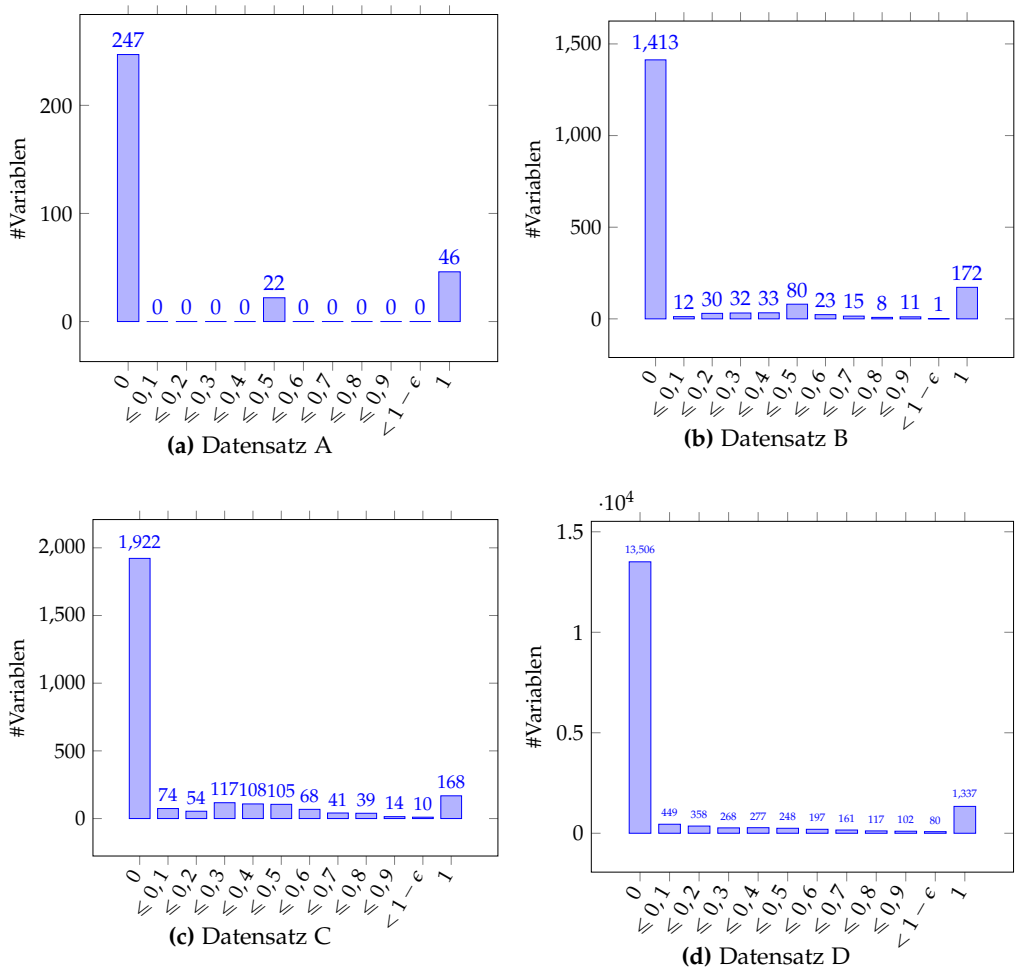

Abbildung 6.3. Histogramme der optimalen relaxierten LP-Lösungen

zu großen Lösungsraums, sondern als Heuristik zur gezielten Suche im weiterhin großen Lösungsraum verwendet.

Dazu wird auf den Variablen der CSP-Instanz eine Ordnung eingeführt und mit der Werteheuristik „Choose Max Value“ kombiniert. Durch diese Ordnung kann vorab festgelegt werden, welche Variablen bevorzugt auf 1 gesetzt werden, sofern dies noch konfliktfrei möglich ist.

6.5.1. Ordnung der Variablen

In erster Linie werden die Variablen nach ihrem Wert in der LP-Relaxierung geordnet. Da aber vor allem die Werte $1,0$ und $0,5$ sowie einige andere Brüche relativ häufig vorkommen, bietet es sich an, die Ordnung noch weiter zu verfeinern. Dazu verwenden wir Überlegungen aus Unterabschnitt 5.3.7 und Unterabschnitt 5.4.3, d. h. wir ziehen Kernstunden vor und bevorzugen Unterrichte mit mehr Abhängigkeiten, was insbesondere größere Kopplungen betrifft.

Zunächst definieren wir den Grad d eines Unterrichts $u \in U$ als die Anzahl aller Unterrichte, an denen beteiligte Lehrer und Klassen teilnehmen:

$$d(u) = \sum_{\substack{u \in U_l \\ l \in L}} |U_l| + \sum_{\substack{u \in U_k \\ k \in K}} |U_k|$$

Damit ergibt sich die Ordnung zweier Variablen $x_{u_1 z_1}$ und $x_{u_2 z_2}$ anhand der folgenden Kriterien, wobei ein Kriterium nur betrachtet wird, wenn alle vorherigen Kriterien keine Unterscheidung ermöglicht haben. $x_{u_1 z_1}$ ist vor $x_{u_2 z_2}$ einzuordnen, wenn:

- $x_{u_1 z_1} > x_{u_2 z_2}$ (Sortierung nach der LP-Relaxierung)
- $z_1 \in Z_k \wedge z_2 \notin Z_k$ (Kernstunden zuerst belegen)
- $|\{k \in K \mid u_1 \in U_k\}| > |\{k \in K \mid u_2 \in U_k\}|$ (mehr beteiligte Klassen)
- $|\{l \in L \mid u_1 \in U_l\}| > |\{l \in L \mid u_2 \in U_l\}|$ (mehr beteiligte Lehrer)
- $d(u_1) > d(u_2)$ (Grad des Unterrichts bei gleicher Klassen- und Lehrer-Zahl)
- $e(u_1) > e(u_2)$ (Einfügereihenfolge)

Dabei ist $e : U \mapsto \mathbb{N}$ die Reihenfolge der Unterrichte im Datensatz, die nur verwendet wird, um bei ansonsten gleichen Unterrichten trotzdem eine deterministische Ordnung zu erhalten.

6.5.2. Ergebnisse

Um Zufallstreffer auszuschließen und die Auswirkungen von unterschiedlichen Lösungen für die LP-Relaxierung zu untersuchen, wurde jede Berechnung wieder 100 Mal durchgeführt:

Tabelle 6.9. Einfluss der Sortierung auf den Erfolg der Berechnung

Datensatz	A	B	C	D
Anzahl gelöster Instanzen (Or-Tools)	100	100	46	4
Anzahl gelöster Instanzen (Gecode)	100	100	43	4
Anzahl Timeouts >10 s (Or-Tools)	0	0	54	96
Anzahl Timeouts >10 s (Gecode)	0	0	57	96
Anzahl Variablen	156	1036	1512	8602
Durchschnittlich Anzahl 1-Werte (gelöst)	5,2	113,0	186,5	174,5
Durchschnittlich Anzahl 1-Werte (Timeout)	-	-	184,5	164,4

Die Zahl der erfolgreichen Berechnungen ist etwas niedriger als beim Fixieren (vgl. Tabelle 6.5), dafür kommt es selbstverständlich nicht mehr vor, dass eine Probleminstanz gänzlich unlösbar wird.

Es sei nochmals darauf hingewiesen, dass in den Fällen, in denen keine Lösung gefunden wurde, bereits nach 10 Sekunden abgebrochen wurde. Weil es jedoch eines der Ziele des neuen Ansatzes ist, die Variablen so zu sortieren, dass die erste zulässige Lösung bereits sehr früh im Suchbaum auftritt, wurden einige längere Berechnungen mit den Datensätzen C und D durchgeführt, die zwischen einer und fünf Stunden dauerten. Diese waren alle erfolglos, d. h. es wäre noch mehr Zeit nötig, um eine zulässige Lösung zu bestimmen.

6.5.3. Analyse mit kleiner Probleminstanz

Die in Unterabschnitt 6.4.1 vorgestellte minimale Probleminstanz, die beim Fixieren der 1-Werte unlösbar wird, bleibt durch die bloße Sortierung der Variablen natürlich lösbar. Doch da grundsätzlich die Variablen mit hohen Werten in der LP-Relaxierung zuerst mit 1 belegt werden, wird der Fehler bei dieser Probleminstanz direkt zu Beginn begangen.

Bei der in Abbildung 6.4 dargestellten Suche werden mit den ersten beiden Entscheidungen ($AF = 1$ und $BF = 1$) bereits zwei Unterrichte so gelegt, dass es keine zulässige Lösung für die übrigen mehr geben kann. Weil aber jeder der anderen Unterrichte (hier nur AX) alleine noch platziert werden könnte, wird dieser Fehler vom Algorithmus nicht sofort erkannt. Stattdessen versucht der Solver, einen weiteren Unterricht (AX) zu platzieren, wodurch sich ein Widerspruch ergibt – egal auf welchen der beiden verbleibenden Zeitslots der Unterricht gelegt wird. Erst nachdem der komplette Teilbaum durchsucht ist, wird die Entscheidung per Backtracking revidiert und dann sofort eine Lösung gefunden.

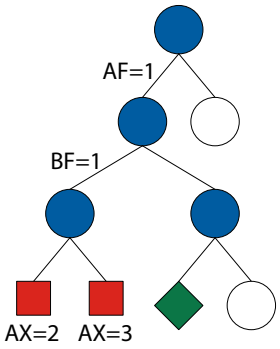

Abbildung 6.4. Suchbaum einer kleinen Probleminstanz mit Variablen-Sortierung nach LP-Relaxierung

Im gegebenen Beispiel wird die Lösung trotzdem in wenigen Mikrosekunden gefunden und es mussten davor nur zwei unzulässige Blattknoten untersucht werden. Das liegt ausschließlich daran, dass der Suchbaum mit einer Tiefe von drei Entscheidungen (Kanten) insgesamt sehr klein ist und in weniger als einer Millisekunde vollständig durchsucht werden könnte. Bei größeren Probleminstanzen liegen jedoch viele weitere Entscheidungen zwischen den mit 1 belegten Variablen, für die die Fehlentscheidung getroffen wurde, und den Variablen, bei deren Belegung der Fehler entdeckt wird. Der Teilbaum unterhalb der Fehlentscheidung, der sicher keine Lösung enthält, kann also sehr groß ausfallen, was die Berechnung enorm verlangsamt. Die Messergebnisse aus Tabelle 6.9 bestätigen diese Überlegungen.

6.5.4. Position der ersten Fehlentscheidung

Um genauer einschätzen zu können, wie weit oben oder unten im Baum das vorgestellte Verfahren die erste Fehlentscheidung trifft, wurde diese Position in mehreren Testläufen (100 pro Datensatz) genauer bestimmt.

Der CSP-Solver belegt alle Variablen zunächst mit 1, bis das nicht mehr konfliktfrei möglich ist. Wenn ein Konflikt auftritt, wird stets die letzte gemachte Entscheidung revidiert. Also befindet sich eine lange Kette von 1-Belegungen am Anfang des Suchbaums, solange dieser erst zu einem kleinen Teil durchsucht wurde. Die erste solche 1, die zusammen mit allen vorherigen die Probleminstanz unlösbar macht, ist die erste Fehlentscheidung.

Mit Hilfe von Gurobi können die Probleminstanzen als ganzzahliges lineares Programm modelliert und in kurzer Zeit auf Lösbarkeit überprüft werden. Es wurden die Variablen in

der gleichen Reihenfolge auf 1 fixiert, in der auch der CSP-Solver sie bearbeiten würde, und jeweils die Position der Variablen festgehalten, nach deren Fixierung die Probleminstanz unlösbar wurde. Die Position lässt sich dabei durch zwei Kennzahlen charakterisieren: zum einen durch den relativen Index im Vergleich zu allen Variablen (z. B. wäre die 400. von insgesamt 8000 Variablen bei 5 %) und zum anderen durch ihren Wert in der LP-Relaxierung (was der Position im Histogramm entspricht, vgl. Abbildung 6.3).

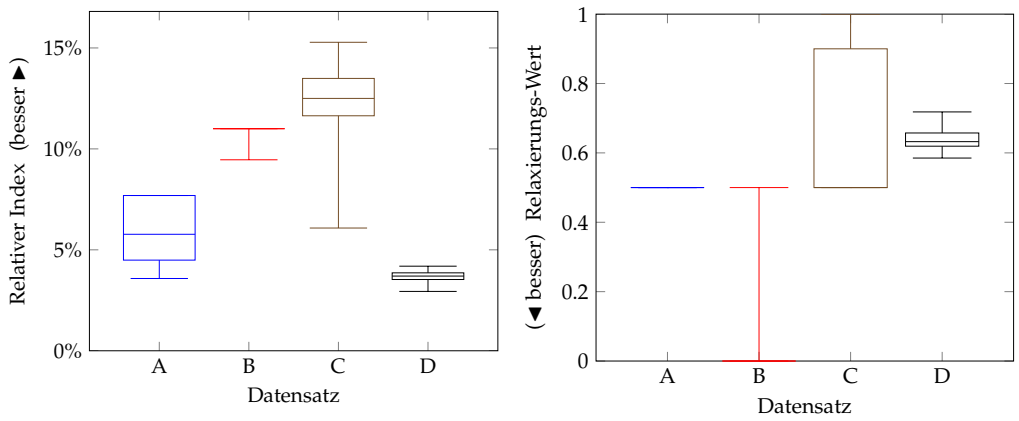

Abbildung 6.5. Position der ersten Fehlentscheidung im CSP-Suchbaum

Die Diagramme in Abbildung 6.5 machen deutlich, dass zum einen alle ersten Fehler bereits sehr früh auftreten (im ersten Sechstel) und dass zum anderen im Fall von Datensatz C bereits das Auswählen einer 1 aus der LP-Relaxierung ein Fehler sein kann. Dieser Umstand ist bereits aus Unterabschnitt 6.4.1 bekannt und führt dazu, dass es nicht möglich ist, alle Einsen zu fixieren ohne die Lösbarkeit zu beeinträchtigen.

6.5.5. Sortierung verbessern

Eine naheliegende Idee zur Verbesserung der Suche ist es daher, nach besseren Heuristiken für die Sortierung zu suchen, um die erste Fehlentscheidung möglichst weit hinauszuzögern.

Die Analyse von einigen problematischen Beispielen (z. B. Unterabschnitt 6.4.1) zeigt, dass stets Kopplungen beteiligt sind, die nicht mehr platziert werden können, nachdem anderer Unterricht auf ungünstige Zeitslots gelegt wurde. Um das Problem zu umgehen, könnte man versuchen, die Kopplungen möglichst früh zu platzieren. Das erreicht man beispielsweise, indem man die betreffenden Variablen möglichst früh einsortiert – etwa

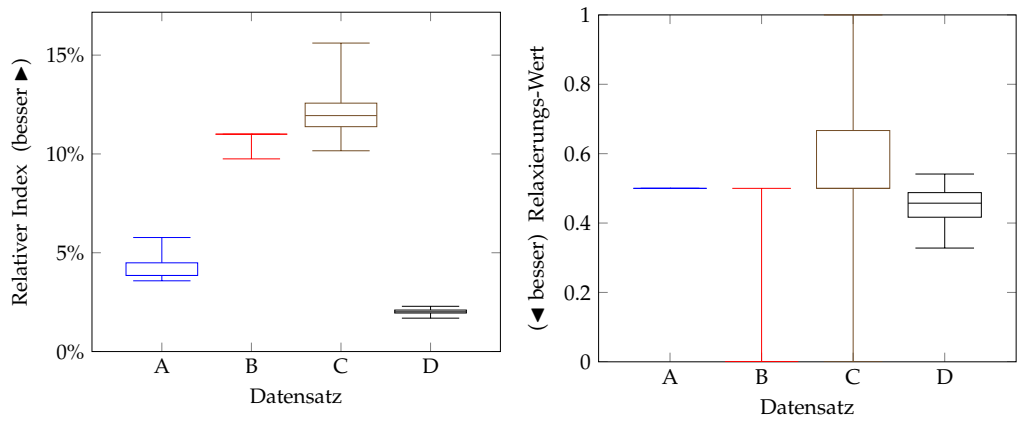

Abbildung 6.6. Position der ersten Fehlentscheidung im CSP-Suchbaum (Kopplungen priorisiert)

direkt hinter alle Variablen mit dem Wert $1,0$ in der LP-Relaxierung. Diese alternative Sortierung führt zu keinen wesentlich besseren Ergebnissen (siehe Abbildung 6.6).

Es wurden einige solche Abwandlungen der Variablensortierung getestet, von denen jedoch keine zu einer nennenswerten Verbesserung führte.

6.6. Verschärfung der LP-Relaxierung

Bisher wurde versucht, durch Fixieren oder Sortieren der Variablen anhand der LP-Relaxierung und einiger weiterer Kriterien den Ablauf der Suche zu verbessern. Anstatt dabei nur die Kriterien zu verändern, kann man auch versuchen, die LP-Relaxierung selbst zu verbessern. Dazu muss die LP-Instanz des Problems gezielt um Constraints erweitert werden, die für eine ILP-Instanz eigentlich redundant sind. Das bedeutet, dass dadurch keine ganzzahligen Lösungen verloren gehen, wohl aber reelle Lösungen, sodass die optimale relaxierte Lösung näher bei der gesuchten ganzzahligen Lösung liegt.

In anderen Problembereichen bringt diese Methode große Vorteile. So lässt sich beispielsweise die LP-Relaxierung bei der Suche nach unabhängigen Mengen in Graphen erheblich verbessern, indem man für jeden ungeraden Zyklus im Graphen einen zusätzlichen Constraint hinzufügt [Grötschel et al., 1988, Kapitel 9.1].

6.6.1. Idee

Ein analoger Ansatz für das Stundenplanproblem lässt sich aus der schwierigen Probleminstanz ableiten, die in Unterabschnitt 6.4.1 vorgestellt wurde. Die Werteverteilung

Tabelle 6.10. Schlechte LP-Relaxierung bei ungünstiger Probleminstanz

z \ u	$A\text{-}F_A$	A-G	$B\text{-}F_B$	B-G	A-B-H
1	1	0	1	0	0
2	0	1/2	0	1/2	1/2
3	0	1/2	0	1/2	1/2

der LP-Relaxierung bei dieser Probleminstanz ist in Tabelle 6.10 dargestellt. Nach dem Setzen der beiden 1-Werte gibt es für die übrigen Unterrichte keine zulässige Verteilung mehr. Der Grund dafür ist, dass nur noch zwei Zeitslots zur Verfügung stehen, aber drei Unterrichte platziert werden müssen, was prinzipiell nur möglich ist, wenn zwei davon gleichzeitig stattfinden. Im konkreten Fall gibt es aber keine zwei Unterrichte, die gleichzeitig stattfinden könnten: Die Kopplung hat mit den beiden anderen Unterrichten je eine Klasse gemeinsam, und die beiden Unterrichte im Fach G haben denselben Lehrer.

Das Kernproblem lässt sich noch etwas weiter reduzieren, indem nur ein einziger Zeitslot (egal ob 2 oder 3) betrachtet wird: Obwohl von den drei Unterrichten nur einer stattfinden kann, haben alle drei in der LP-Relaxierung den Wert $1/2$, in der Summe also mehr als 1. Dies lässt sich verhindern, indem pro Zeitslot ein zusätzlicher Constraint hinzugefügt wird, der die Summe auf 1 beschränkt.

6.6.2. Verallgemeinerung

Hinweis: Da nun auch Kopplungen betrachtet werden, wird die Definition aus Abschnitt 2.3 wie in Abschnitt 4.3 beschrieben erweitert, sodass pro Unterricht mehrere Klassen, Lehrer und Fächer möglich sind:

$$U \subseteq \mathcal{P}(K) \times \mathcal{P}(F) \times \mathcal{P}(L)$$

Wir betrachten den Konfliktgraphen $G = (U, E)$ der zu lösenden Probleminstanz, in dem die Unterrichte als Knoten dargestellt sind und zwischen zwei Unterrichten genau dann eine Kante existiert, wenn sie nicht gleichzeitig stattfinden können, also:

$$E = \left\{ \{(k_1, f_1, l_2), (k_2, f_2, l_2)\} \in \binom{U}{2} \,\middle|\, k_1 \cap k_2 \neq \varnothing \vee l_1 \cap l_2 \neq \varnothing \right\}$$

Die drei Unterrichte aus dem Beispiel oben haben die Eigenschaft, dass sich jeder mit jedem anderen ausschließt. Sie bilden also eine (kleine) Clique im Konfliktgraphen (siehe Abbildung 6.7).

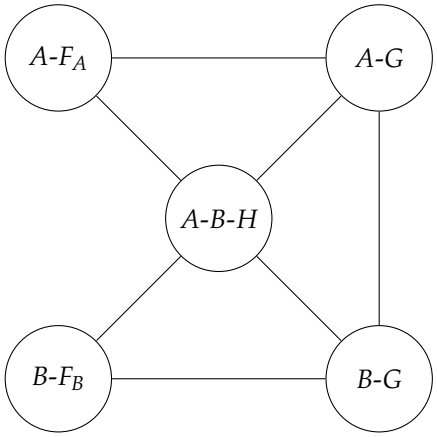

Abbildung 6.7. Konfliktgraph der minimalen Probleminstanz

6.6.3. Maximale Cliquen

In einem Graphen $G = (U, E)$ ist eine *Clique* definiert als Teilmenge $C \subseteq U$ der Knoten, wobei zwischen je zwei Knoten eine Kante existieren muss, also $\binom{C}{2} \subseteq E$. Eine Clique C heißt *maximal* genau dann, wenn sie in keiner anderen Clique echt enthalten ist, also $\nexists D \supsetneq C : \binom{D}{2} \subseteq E$. Während die Suche nach Cliquen einer gegebenen Größe k und damit auch nach der größten Clique NP-vollständig ist [Schöning, 2003], können maximale Cliquen zumindest mit Aufwand $O(nm)$ pro gefundener Clique (mit $n = |U|, m = |E|$) aufgezählt werden [Tsukiyama et al., 1977]. In einem Graph können jedoch exponentiell viele Cliquen existieren.

In der Praxis erweist sich eine Abwandlung des Algorithmus von Bron und Kerbosch [1973] als praktikabel, bei der durch die Wahl eines Pivot-Knotens der Suchbaum signifikant verkleinert wird [Tomita et al., 2006]. Der Bron-Kerbosch-Algorithmus ohne Pivotieren ist im Folgenden als Pseudo-Code dargestellt:

Algorithm 1 Bron-Kerbosch-Algorithmus (findet alle maximalen Cliquen)

Require: $C, M, X \subseteq U$ sind Knotenmengen
1: **function** FIND-MAX-CLIQUES(C, M, X)
2: **if** $M = \varnothing \wedge X = \varnothing$ **then**
3: Speichere maximale Clique C
4: **end if**
5: **for all** $u \in M$ **do**
6: FIND-MAX-CLIQUES($C \cup \{u\}$, $M \cap N(u)$, $X \cap N(u)$)
7: $M \leftarrow M \setminus \{u\}$
8: $X \leftarrow X \cup \{u\}$
9: **end for**
10: **end function**

Dabei ist $N(v) := \{u \in U \mid \{u, v\} \in E\}$ die Menge der direkten Nachbarn von v. In der Menge C ist stets die aktuelle Clique gespeichert, in M sind weitere mögliche Knoten, die in die Clique aufgenommen werden könnten, und X enthält die Knoten, die nicht mehr in die Clique aufgenommen werden sollen. Der Algorithmus wird mit dem initialen Aufruf FIND-MAX-CLIQUES($\varnothing$, U, $\varnothing$) gestartet.

Durch die Wahl eines Pivot-Knotens $p \in M \cup X$ lässt sich die Suche – je nach Struktur des Graphen – erheblich beschleunigen [Tomita et al., 2006]. Die Idee ist, dass in allen maximalen Cliquen, die noch gefunden werden können, entweder p selbst enthalten ist, oder wenn nicht, dann auch keiner von den Nachbarn von p. Denn alle Knoten in $m \in M \cup X$ haben die Eigenschaft, dass sie mit allen in der aktuellen Clique C verbunden sind, sodass $C \cup \{m\}$ wieder eine (größere) Clique wäre. Ein Nachbar von p kann C aber nur dann zu einer maximalen Clique ergänzen, wenn auch p in der Clique ist, denn ohne p wäre die Clique nicht maximal, weil sie sich mit p einfach vergrößern lassen würde. Da alle Nachbarn von p in dem rekursiven Aufruf betrachtet werden, in dem zunächst p selbst ausgewählt wurde, müssen sie nicht ein weiteres Mal betrachtet werden, wenn p nicht in C enthalten ist.

Es lassen sich also alle Nachbarn $N(p)$ *eines* beliebigen Knotens in jeder Runde von der Suche ausschließen, ohne dass dadurch die Vollständigkeit der Suche verloren geht. Um einen möglichst großen Effekt zu erzielen, wählt man als Pivot-Knoten denjenigen Knoten $p \in M \cup X$ mit den meisten Nachbarn in M oder alternativ einfach mit den meisten Nachbarn überhaupt, d. h. mit dem höchsten Grad.

Algorithm 2 Bron-Kerbosch-Algorithmus mit Pivotieren

Require: $C, M, X \subseteq U$ sind Knotenmengen
1: **function** FIND-MAX-CLIQUES(C, M, X)
2: **if** $M = \varnothing \wedge X = \varnothing$ **then**
3: Speichere maximale Clique C
4: **end if**
5: Wähle $p \in M \cup X$ mit $|N(p)|$ maximal (oder $|N(p) \cap M|$ maximal)
6: **for all** $u \in M \setminus N(p)$ **do**
7: FIND-MAX-CLIQUES($C \cup \{u\}$, $M \cap N(u)$, $X \cap N(u)$)
8: $M \leftarrow M \setminus \{u\}$
9: $X \leftarrow X \cup \{u\}$
10: **end for**
11: **end function**

6.6.4. Interessante Cliquen

Cliquen im Konfliktgraphen sind Mengen von Unterrichten, die sich paarweise gegenseitig ausschließen. Naheliegende Cliquen sind insbesondere die Mengen aller Unterrichte eines bestimmten Lehrers oder einer bestimmten Klasse – und meistens sind diese auch maximal. Da diese ohnehin über die Konfliktfreiheits-Bedingung abgedeckt sind und einen Großteil der Cliquen ausmachen, lohnt es sich nicht, diese mit dem Bron-Kerbosch-Algorithmus zu finden. Im Fall des großen Testdatensatzes D sind 54 der insgesamt 96 Cliquen uninteressant, weil sie bereits über die Konfliktfreiheits-Bedingung abgedeckt sind. Bei der kleinen Beispielinstanz (siehe Abbildung 6.7) betrifft das die obere und die untere Clique.

Um die Cliquensuche weiter zu verkürzen, hilft die Überlegung, dass Cliquen immer uninteressant sind, wenn sie nur aus normalen Unterrichten bestehen, also wenn keine Kopplungen enthalten sind. Normale Unterrichte sind solche, bei denen genau ein Lehrer genau eine Klasse in genau einem Fach unterrichtet. Die Aussage, dass bei solchen Cliquen entweder alle Lehrer gleich sind, oder alle Klassen, oder beides, lässt sich per Induktion über die Größe n der Clique beweisen:

- $n = 1$: Die Aussage gilt, es sind sowohl alle Lehrer als auch alle Klassen gleich, weil nur ein Unterricht betrachtet wird.
- $n = 2$: Da beide Unterrichte eine Clique bilden, sind sie durch eine Kante verbunden, weil zwischen beiden ein Konflikt besteht. Der kann nur dadurch verursacht werden, dass beide denselben Lehrer und/oder dieselbe Klasse haben.
- $n > 2$: Da jede Teilmenge der Clique wieder eine Clique ist, kann ein Unterricht u aus der Clique entfernt werden. Für die übrige Teilmenge gilt per Induktionsvoraussetzung, dass alle Unterrichte denselben Lehrer und/oder dieselbe Klasse haben.

    1. Fall: Sie haben dieselbe Klasse und denselben Lehrer. Da u zu allen anderen Unterrichten eine Kante hat, hat er entweder auch denselben Lehrer oder auch dieselbe Klasse.
    2. Fall: Sie haben dieselbe Klasse, aber es gibt mindestens zwei mit unterschiedlichem Lehrer. Da u zu diesen beiden eine Kante hat, muss er dieselbe Klasse haben. Denn hätte er sie nicht, dann müsste er mit den beiden anderen Unterrichten einen Lehrerkonflikt haben, was aber nicht möglich ist, weil diese verschiedene Lehrer haben.
    3. Fall: Sie haben denselben Lehrer, aber es gibt mindestens zwei mit unterschiedlicher Klasse: analog zu Fall 2.

Wenn in einer Clique alle Lehrer oder alle Klassen gleich sind, existiert für die Clique bereits ein Constraint aus der Konfliktfreiheits-Bedingung, sodass die Clique uninteressant ist.

Demnach sind nur die Cliquen potenziell interessant, in denen mindestens eine Kopplung vorkommt. In Abbildung 6.7 ist es also nur die Clique auf der rechten Seite. Um solche Cliquen schneller zu ermitteln, kann die Suche direkt ausgehend von den Kopplungen gestartet werden:

Algorithm 3 Cliquensuche ausgehend von Kopplungen

Require: U ist die Menge aller Unterrichte

```
1: function FIND-MAX-CLIQUES(U)
2:     X ← ∅
3:     for all u ∈ U do
4:         if u ist Kopplung then
5:             FIND-MAX-CLIQUES({u}, N(u) \ X, X ∩ N(u))
6:         end if
7:     end for
8: end function
```

6.6.5. Ergebnisse der Cliquensuche

Tabelle 6.11. Statistiken zur Cliquensuche

Datensatz	A	B	C	D
Anzahl Kopplungen	4	3	4	21
Anzahl maximaler Cliquen	7	8	19	96
Davon interessante Cliquen	4	1	3	42
Durchsuchte Cliquen (Bron-Kerbosch)	7	8	19	96
Laufzeit (Bron-Kerbosch ohne Pivot)	60 ms	72 ms	112 ms	36,1 s
Laufzeit (Bron-Kerbosch mit Pivot)	32 ms	44 ms	72 ms	920 ms
Durchsuchte Cliquen (Start in Kopplungen)	7	6	13	81
Laufzeit (Start in Kopplungen mit Pivot)	32 ms	34 ms	54 ms	476 ms

Die Ergebnisse in Tabelle 6.11 zeigen, dass vor allem das Pivotieren, aber auch das gezielte Starten in den Kopplungen die Effizienz der Cliquensuche verbessern.

6.6.6. Auswirkungen auf die erste Fehlerposition

Zum Vergleich mit Abbildung 6.5 wurde wieder für 100 zufällige Lösungen der LP-Relaxierung ermittelt, wie früh die erste Fehlentscheidung passieren würde, wenn der CSP-Solver die Variablen in der Reihenfolge der reellen Lösung auf 1 setzt. Die Ergebnisse in Abbildung 6.8 sind durchgehend besser als ohne die Cliquen-Constraints. Insbesondere

gibt es – wie beabsichtigt – auch bei Datensatz C keinen Fall mehr, in dem das Fixieren einer ganzzahligen 1 eine Fehlentscheidung wäre.

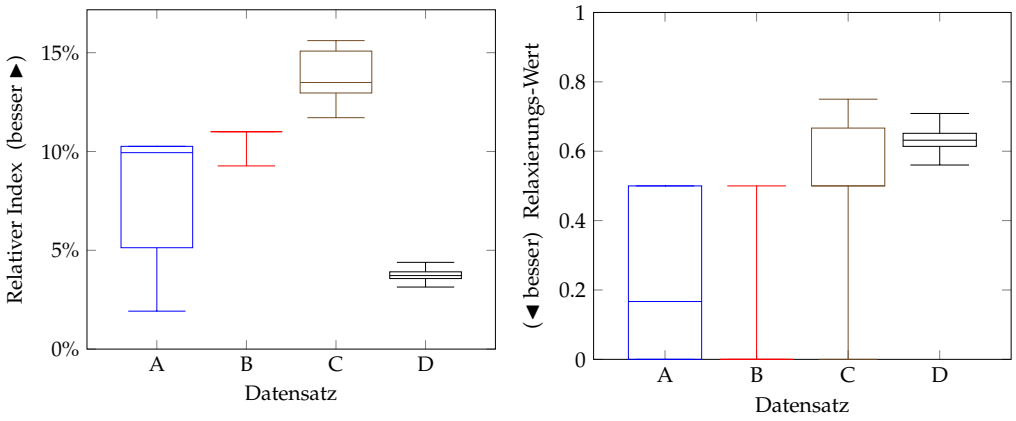

Abbildung 6.8. Position der ersten Fehlentscheidung im CSP-Suchbaum (mit Cliquen-Constraints)

6.6.7. CSP-Berechnungen mit Cliquen-Constraints

Um die Berechnung mit dem CSP-Solver zu beschleunigen, könnte man in die CSP-Instanz nicht nur die Ergebnisse der verbesserten LP-Relaxierung einfließen lassen, sondern zusätzlich die Cliquen-Constraints direkt zur CSP-Instanz hinzufügen. Aus theoretischer Sicht bringt das aber keinen Mehrwert, weil der CSP-Solver nur mit diskreten Werten aus den Domänen der Variablen arbeitet. Wenn beispielsweise mittels Constraint Propagation darauf geschlossen werden kann, dass ein Unterricht zu einem bestimmten Zeitpunkt nicht mehr stattfinden kann, dann deswegen, weil ein Nachbar-Unterricht (im Konfliktgraphen) bereits zu diesem Zeitpunkt stattfindet. Diese Information kann jedoch alleine dadurch propagiert werden, dass eine Kante im Konfliktgraphen vorliegt. Die zusätzliche Information, dass aus einer Menge von n Unterrichten nur m gleichzeitig stattfinden können (mit $m < n$), hilft dem CSP-Solver nicht, weil er erst einen Unterricht setzen muss, bevor ein Constraint propagiert werden kann – und dann ist ohnehin schon sicher, dass keiner der anderen $n - 1$ Unterrichte überhaupt noch zu diesem Zeitpunkt stattfinden kann. Probeweise durchgeführte Berechnungen bestätigen diese Überlegungen.

Deswegen wurden die Cliquen-Constraints im Folgenden nur zur relaxierten LP-Instanz hinzugefügt. Deren Lösung wurde wiederum 100 Mal zufällig ausgewählt und dann wie in Abschnitt 6.5 beschrieben in die CSP-Instanz eingearbeitet. Die Ergebnisse in Tabelle 6.12 müssen also mit Tabelle 6.9 verglichen werden.

Tabelle 6.12. Einfluss der Cliquen-Constraints auf den Erfolg der Berechnung

Datensatz	A	B	C	D
Anzahl gelöster Instanzen (Or-Tools)	100	100	97	18
Anzahl gelöster Instanzen (Gecode)	100	100	97	16
Anzahl Timeouts >10 s (Or-Tools)	0	0	3	82
Anzahl Timeouts >10 s (Gecode)	0	0	3	84
Anzahl Variablen	156	1036	1512	8602
Durchschnittlich Anzahl 1-Werte (gelöst)	10,58	112,7	197,9	168,4
Durchschnittlich Anzahl 1-Werte (Timeout)	-	-	165,7	160,8

Obwohl sich die Anzahl der gelösten Instanzen im Vergleich zur Berechnung ohne Cliquen-Constraints bereits stark gesteigert hat, bleiben vor allem beim großen Datensatz D die meisten Berechnungen erfolglos. Selbst mit mehreren Stunden Berechnungszeit können diese Instanzen nicht gelöst werden.

6.6.8. Verbesserte Ordnung der Variablen

Die mit Cliquen-Constraints ermittelte Lösung der LP-Relaxierung ist sehr präzise in Bezug auf die 1-Werte. In allen weiteren Experimenten wurde kein Fall mehr beobachtet, in dem ein Wert von $1,0$ aus der Relaxierung nicht auch in der endgültigen Lösung korrekt gewesen wäre. Anstatt die Werte aber zu fixieren (wie in Abschnitt 6.4), wurde die heuristische Variablen-Ordnung aus Unterabschnitt 6.5.1 so angepasst, dass 1-Werte deutlich bevorzugt werden, wohingegen die feine Abstufung der reellen Variablen eher in den Hintergrund gerückt ist, weil die anderen Kriterien wichtiger sind (siehe Überlegungen in Unterabschnitt 5.3.7 und Unterabschnitt 5.4.3).

In der neuen Ordnung wird $x_{u_1 z_1}$ vor $x_{u_2 z_2}$ eingeordnet, wenn:

- $x_{u_1 z_1} = 1 \wedge x_{u_2 z_2} < 1$ (Präferiere Einsen aus der LP-Relaxierung)
- $z_1 \in Z_k \wedge z_2 \notin Z_k$ (Kernstunden zuerst belegen)
- $|\{k \in K \mid u_1 \in U_k\}| > |\{k \in K \mid u_2 \in U_k\}|$ (mehr beteiligte Klassen)
- $|\{l \in L \mid u_1 \in U_l\}| > |\{l \in L \mid u_2 \in U_l\}|$ (mehr beteiligte Lehrer)
- $x_{u_1 z_1} > x_{u_2 z_2}$ (Sortierung nach der LP-Relaxierung)
- $d(u_1) > d(u_2)$ (Grad des Unterrichts bei gleicher Klassen- und Lehrer-Zahl)
- $e(u_1) > e(u_2)$ (Einfügereihenfolge)

Mit dieser neuen Ordnung können die Probleminstanzen fast immer gelöst werden:

Tabelle 6.13. Berechnung mit Cliquen-Constraints und verbesserter Variablen-Ordnung

Datensatz	A	B	C	D
Anzahl gelöster Instanzen (Or-Tools)	100	100	98	99
Anzahl gelöster Instanzen (Gecode)	100	100	97	98
Anzahl Timeouts >10 s (Or-Tools)	0	0	2	1
Anzahl Timeouts >10 s (Gecode)	0	0	3	2
Anzahl Variablen	156	1036	1512	8602
Durchschnittlich Anzahl 1-Werte (gelöst)	10,26	113,2	199,1	161,5
Durchschnittlich Anzahl 1-Werte (Timeout)	-	-	176,6	169,0

6.7. Weitere Bedingungen

Mit der im vorherigen Abschnitt vorgestellten Methode lassen sich alle getesteten Datensätze fast immer lösen. Dabei wurde aber jeweils nur eine zulässige Lösung berechnet, die die Bedingungen „Wochenstunden“, „Konfliktfreiheit“ und „Kernstunden“ erfüllt. Es wurden also nur vergleichsweise wenige Bedingungen eingehalten und insbesondere keine weichen Bedingungen verwendet, sodass keine Optimierung des fertigen Ergebnisses nötig war.

Im Folgenden werden zunächst weitere harte Bedingungen hinzugefügt und die Auswirkungen auf die Berechnung untersucht. Anschließend werden Berechnungen mit weichen Bedingungen durchgeführt. Für die Definition der Bedingungen siehe Kapitel 2. Auf mögliche Varianten der Modellierung wird in diesem Kapitel nicht weiter eingegangen. Alle Bedingungen wurden so implementiert wie in Abschnitt 7.2 oder von Weidler [2012] beschrieben.

6.7.1. Harte Bedingungen

Alle Ergebnisse sind in Tabelle 6.14 dargestellt. Die Bedingungen „Fixe Stunden“ und „Fach-Pro-Tag-Begrenzung“ lassen sich hinzufügen, ohne dass bei der Berechnung Probleme auftreten. Dazu ist anzumerken, dass die Datensätze B und C überhaupt keine fixen Stunden enthalten.

Außerdem zeigt sich beim Datensatz C bereits, dass die LP-Relaxierung und die Constraint Propagation zusammen nicht mehr ausreichen, um die Lösung zu finden. Die CSP-Solver müssen per Backtracking nach Lösungen suchen. Je nach gewählter LP-Relaxierung gelingt das relativ schnell nach einigen Failures, oder es dauert so lange, dass die Berechnung abgebrochen werden muss. Mit der Bedingung „Lehrer-Nichtverfügbarkeiten“ terminiert

keine Berechnung mehr für Datensatz C. Zum Vergleich: Alle Instanzen können von Gurobi innerhalb weniger Sekunden gelöst werden.

Tabelle 6.14. CSP mit zusätzlichen harten Bedingungen

Datensatz	A	B	C	D	
Anzahl Variablen	156	1036	1512	8602	
Anzahl Constraints	103	472	677	2901	Nur Kernstunden
Rechenzeit (Or-Tools)	0 ms	1 ms	2 ms	11 ms	
Rechenzeit (Gecode)	0 ms	4 ms	10 ms	106 ms	
Anzahl Constraints	105	472	677	2919	+ Fixe Stunden
Rechenzeit (Or-Tools)	1 ms	2 ms	3 ms	18 ms	
Rechenzeit (Gecode)	0 ms	4 ms	10 ms	87 ms	
Anzahl Constraints	190	697	1407	5579	+ Fach-Pro-Tag
Rechenzeit (Or-Tools)	0 ms	3 ms	13 ms†	22 ms	
Rechenzeit (Gecode)	0 ms	4 ms	245 ms†	230 ms	
Anzahl Constraints	190	742	1487	5668	+ Nichtverf.
Rechenzeit (Or-Tools)	1 ms	1 ms	>1 h	17 ms	
Rechenzeit (Gecode)	0 ms	4 ms	>1 h	103 ms	
Anzahl Variablen	-‡	-‡	-‡	10192	+ Doppelstunden
Anzahl Constraints	-‡	-‡	-‡	11846	
Rechenzeit (Or-Tools)	-‡	-‡	-‡	>1 h	
Rechenzeit (Gecode)	-‡	-‡	-‡	>1 h	

† Nach 775 Failures
‡ Enthält keine Doppelstunden

6.7.2. Weiche Bedingungen

Durch das Hinzufügen von weichen Bedingungen werden Optimierungsschritte nötig (siehe Unterabschnitt 5.1.3). Insbesondere muss der Solver, *nachdem* er bereits eine optimale Lösung gefunden hat, den gesamten restlichen Suchbaum durchsuchen, um überhaupt feststellen zu können, dass die gefundene Lösung optimal ist.

Für den ersten Test wurden nur die harten Bedingungen „Wochenstunden“, „Konfliktfreiheit“ und „Kernstunden“ verwendet. Als einzige weiche Bedingung wurde die „Unterrichtspriorität“ verwendet (siehe Abschnitt 2.5), die den Unterrichten zu jedem Zeitslot, der keine Kernstunde ist, eine Priorität zuweist, sodass der Unterricht insgesamt eher vormittags stattfindet.

Mit Gurobi wurde der Zielfunktionswert der optimalen Lösung ermittelt. Beide CSP-Solver hatten je eine Stunde Zeit zur Berechnung und in Tabelle 6.15 ist die jeweils letzte/beste gefundene Lösung angegeben.

Tabelle 6.15. CSP mit Zielfunktion

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Größe der Zielfunktion	156	476	392	4862
Anzahl Constraints	104	473	678	2902
Optimum (Minimum)	-78	-56	-136	-362
Beste Lösung (Or-Tools)	-78	-56	-136	-309
Gefunden nach (Or-Tools)	2 ms	2 ms	7 ms	52 min
Terminiert nach (Or-Tools)	>1 h	>1 h	>1 h	>1 h
Beste Lösung (Gecode)	-78	-56	-136	-309
Gefunden nach (Gecode)	5 ms	4 ms	181 ms	55 min
Terminiert nach (Gecode)	>1 h	>1 h	>1 h	>1 h

Für die Datensätze A bis C kann innerhalb weniger Millisekunden eine optimale Lösung gefunden werden. Im Fall der Datensätze B und C ist bereits die erste gefundene Lösung optimal, was auf eine gute Lösung der LP-Relaxierung schließen lässt. Aber auch für Datensatz A wird nach nur 4 Failures bereits eine optimale Lösung gefunden.

Auch wenn die optimale Lösung gefunden ist, kann die Suche noch nicht terminieren, weil der Solver noch nicht weiß, dass die Lösung optimal ist. Die weitere Suche, die den Optimalitätsbeweis liefern würde, dauert länger als eine Stunde und wurde daher abgebrochen.

Der Datensatz D ist deutlich größer als die anderen, sodass in der gegebenen Zeit keine optimale Lösung gefunden werden kann. Die gefundenen zulässigen Lösungen nähern sich binnen weniger Minuten dem Wert -307 oder -320 (je nach ausgewählter Lösung der LP-Relaxierung). Danach verbessern sich die Lösungen nur noch sehr langsam und auch nach fünf Stunden ist der Wert höchstens um -5 besser und damit noch weit vom Optimum entfernt.

Für den zweiten Test wurden die harten Bedingungen „Fixe Stunden“, „Lehrer-Nichtverfügbarkeiten“ und „Fach-pro-Tag“ sowie die weiche Bedingung „Lehrer-Verfügbarkeiten“ hinzugefügt.

Tabelle 6.16. CSP mit Zielfunktion und zusätzlichen harten Bedingungen

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Größe der Zielfunktion	140	480	397	5510
Anzahl Constraints	231	883	1489	6604
Optimum (Minimum)	-74	-54	-134	-346
Beste Lösung (Or-Tools)	-74	-52	-134	-214
Gefunden nach (Or-Tools)	0 ms	3,9 s	264 ms	51 min
Terminiert nach (Or-Tools)	20,1 s	>1 h	>2 h	>2 h
Beste Lösung (Gecode)	-74	-52	-134	-215
Gefunden nach (Gecode)	0 ms	15,5 s	609 ms	38 min
Terminiert nach (Gecode)	12,7 s	>1 h	>1 h	>1 h

Während beim kleinen Datensatz A die zusätzliche Einschränkung des Suchbaums dazu führt, dass die optimale Lösung binnen weniger Sekunden bestätigt werden kann, sind die übrigen Datensätze weiterhin zu groß um in akzeptabler Zeit eine Lösung zu erhalten. Stattdessen zeigt sich, dass die Suche nach besseren Lösungen aufgrund der größeren Zahl von Constraints nun länger dauert. Mit der zusätzlichen harten Bedingung „Doppelstunden“ wird für Datensatz D in einer Stunde überhaupt keine Lösung mehr gefunden.

Auch hier hat die von Gurobi ausgewählte Lösung der LP-Relaxierung erheblichen Einfluss auf die weitere Berechnung. Obwohl die LP-Relaxierung ebenfalls mit Zielfunktion gelöst wird, was die Menge der möglichen Lösungen stark einschränkt, gibt es weiterhin verschiedene optimale Lösungen, die sich vor allem in der Größe des ganzzahligen Anteils stark unterscheiden können. Dadurch kann z. B. für den Datensatz C binnen Millisekunden eine optimale Lösung gefunden werden, während es bei Datensatz B, der durch Vereinfachung von Datensatz C entstanden ist, deutlich länger für eine suboptimale Lösung dauert.

6.7.3. Auswertung

Die Suche mittels Constraint Propagation und Backtracking ist nur dann in der Lage, eine Lösung zu finden, wenn diese entweder direkt am Anfang des Suchbaums liegt, was durch eine geschickte Sortierung der Variablen erreicht werden kann, oder wenn die Lösungen allgemein dicht gestreut sind. Letzteres war bei den meisten Tests der vorherigen Kapitel vor allem deswegen der Fall, weil einige Bedingungen weggelassen wurden. Die Tests mit zusätzlichen Bedingungen zeigen, dass insbesondere der Datensatz C schwer zu lösen ist, was sich damit erklären lässt, dass die Klassen in diesem Datensatz fast oder genau so viel

Unterricht haben wie es Wochenstunden gibt. Für die Verteilung des Unterrichts gibt es also nur sehr wenige Freiräume. Durch die in der Realität natürlich ebenfalls geforderten Doppelstunden wird aber auch der Datensatz D schwerer lösbar.

Jede zusätzliche Bedingung führt also dazu, dass zulässige Lösungen im Suchbaum seltener werden. Um trotzdem noch eine zulässige Lösung finden zu können, müssten die Such-Heuristik und/oder die Reihenfolge der Variablen entsprechend angepasst werden, so wie es in den vorherigen Kapiteln für die Kernstunden und die Konfliktfreiheit beschrieben wurde. Dieser Ansatz wird jedoch mit zunehmender Anzahl an verschiedenen Bedingungen schwieriger, weil diese sich zum Teil widersprechen und es keine Sortierung oder Heuristik gibt, die allen gerecht wird.

Selbst wenn es gelingen würde, eine solche Heuristik zu finden, die es für jeden Datensatz und beliebige Bedingungen ermöglicht, die zulässigen Lösungen schnell auffindbar zu machen, wäre die Größe des Suchbaums weiterhin ein Problem, sobald weiche Bedingungen beachtet werden sollen. Denn dafür muss der Suchbaum, der mit Einschränkung auf den optimalen Zielfunktionswert übrig bleibt, komplett durchsucht werden – weil es in diesem letzten Suchbaum per Definition keine zulässige Lösung mehr gibt. Dafür sind die Probleminstanzen, die in der Praxis der Stundenplanung vorkommen, schlicht zu groß.

Aus diesem Grund betrachten wir im nächsten Kapitel einen Ansatz, der nicht auf variablenbasierter Suche aufbaut, sondern mit einer eher „globalen“ Perspektive nach Lösungen sucht und mittels Resolution den Optimalitätsbeweis führen kann.

Kapitel 7

Lösung mittels pseudo-boolescher Optimierung

Dieses Kapitel beschreibt einen Ansatz, das Stundenplanproblem mit pseudo-boolescher Optimierung zu lösen. Zunächst wird in Abschnitt 7.1 das Verfahren allgemein vorgestellt. Abschnitt 7.2 gibt für alle benötigten Bedingungen des Stundenplanproblems eine binäre Formulierung an. Abschnitt 7.3 geht auf die Implementierung und Rechenergebnisse ein und in Abschnitt 7.4 wird das Verfahren mit der LP-Relaxierung kombiniert.

7.1. Einführung

Im Allgemeinen ist eine pseudo-boolesche Funktion definiert als $f : \mathbb{B}^n \mapsto \mathbb{R}$ [Boros und Hammer, 2002]. Funktionsterme können in Literalschreibweise ausgedrückt werden mit $\overline{x_i} := (1 - x_i)$.

Für das Stundenplanproblem in der hier vorgestellten Form gelingt es sogar, alle Bedingungen als lineare Constraints zu formulieren, sodass sie der Definition von Chai und Kuehlmann [2005] entsprechen:

$$\sum a_i \cdot l_i \geqslant k \quad \text{mit } a_i, k \in \mathbb{R},\ l_i \in \{x_i, \overline{x_i}\},\ x_i \in \mathbb{B}$$

Hinweis: Constraints mit $=$ oder $\leqslant$ lassen sich entsprechend umformen.

Vervollständigt wird die Probleminstanz durch eine (lineare) pseudo-boolesche Zielfunktion, deren Minimum bzw. Maximum gesucht ist.

7.1.1. Lösungsverfahren

Für die pseudo-boolesche Optimierung (PBO) gibt es mehrere grundsätzlich verschiedene Lösungsansätze. Einige Solver verfolgen den Ansatz, PBO-Instanzen auf ganzzahlige lineare Programmierung (ILP) oder Constraint Programming (CSP) zu reduzieren. Da diese

Techniken bereits in den vorherigen Kapiteln ausführlich durch direkte Modellierungen betrachtet wurden, wurden diese Solver hier ausgeschlossen.

Von besonderem Interesse sind deshalb die Solver, die auf SAT-Verfahren basieren, indem zum Beispiel die pseudo-booleschen Constraints in aussagenlogische Klauseln übersetzt werden [Eén und Sörensson, 2006] oder indem ein SAT-Solver so erweitert wird, dass er direkt mit den pseudo-booleschen Constraints umgehen kann [Le Berre und Parrain, 2010]. Um damit auch Optimierungsprobleme lösen zu können, bietet sich ein ähnlicher Ansatz wie beim Constraint Programming (siehe Unterabschnitt 5.1.3) an, bei dem eine zulässige Lösung durch Hinzufügen von Constraints solange verbessert wird, bis es keine bessere Lösung mehr gibt. Darüber hinaus gibt es fortgeschrittenere Ansätze, die zum Beispiel Cutting Planes verwenden [Manquinho und Marques-Silva, 2006].

7.2. Vollständige Binärisierung

Selbst wenn man als Grundlage der Modellierung binäre Variablen wählt, wie es der von Weidler [2012] eingeführte und in Abschnitt 3.3 vorgestellte LP-Ansatz tut, können zur Modellierung komplexerer Bedingungen zusätzliche Variablen nötig sein, die nicht unbedingt binär sind [z. B. Weidler, 2012, Kap. 4.2.3, 4.2.4]. Deswegen muss für alle Bedingungen eine binäre Formulierung gefunden werden, insbesondere für solche Bedingungen, die in dieser Arbeit bislang noch nicht genauer betrachtet wurden. Denn wenn eine solche Formulierung nicht oder nur sehr ineffizient möglich wäre, wäre der gesamte pseudo-boolesche Ansatz für die Praxis untauglich.

Wie beim LP-Ansatz (siehe Unterabschnitt 3.3.1) wählen wir als Grundlage der Modellierung die Variablen x_{uz} mit $u \in U$ und $z \in Z$. Da die Solver lineare Ungleichungen direkt als Constraints akzeptieren, lassen sich die Konfliktfreiheits- und die Kernstunden-Bedingung analog formulieren. Für alle anderen Bedingungen wird im Folgenden eine mögliche Formulierung mit binären Variablen und pseudo-booleschen Constraints angegeben. Zur Definition der Bedingungen selbst siehe Abschnitt 2.4 und Abschnitt 2.5.

7.2.1. Fixe Stunden und Nichtverfügbarkeiten

Die Bedingung, dass der Unterricht $u \in U$ zum Zeitpunkt $z \in Z$ sicher stattfinden soll, lässt sich durch den folgenden Constraint ausdrücken:

$$x_{uz} = 1$$

Ebenso einfach lässt sich verhindern, dass ein Lehrer $l \in L$ zu einem Zeitpunkt $z \in Z$ unterrichten muss, an dem er nicht verfügbar ist:

$$\forall u \in U_l : \; x_{uz} = 0$$

7.2.2. Doppelstunden

Es gibt eine Variante des Doppelstundensystems, die sehr einfach umzusetzen ist: Wenn der gesamte Stundenplan nur aus Doppelstunden besteht, halbiert man einfach die Wochenstundenzahl jedes Unterrichts und verdoppelt die Länge jeder Unterrichtsstunde.

Wenn das Doppelstundensystem nicht den ganzen Tag über oder nicht für alle Unterrichte gelten soll, kann man für jeden Unterricht $u \in U$ und jedes Zeitslot-Paar $(z_1, z_2) \in D$ erzwingen ($D \subset Z^2$ ist die Menge der Zeitslot-Paare, für die die Doppelstunden-Bedingung gelten soll), dass der Unterricht stets paarweise stattfindet [siehe Weidler, 2012, Kap. 4.1.6]:

$$x_{uz_1} = x_{uz_2}$$

Für Unterrichte mit ungerader Wochenstundenzahl bedeutet das, dass mindestens eine Unterrichtsstunde außerhalb des Doppelstundenbereichs stattfinden muss. Es gibt aber in der Praxis viele Fälle, in denen das Doppelstundensystem grundsätzlich den ganzen Tag über angewendet wird, obwohl es Unterrichte mit ungerader Wochenstundenzahl gibt. Diese werden entweder 14-tägig unterrichtet, was man leicht modellieren kann, indem man die Wochenstundenzahl auf die nächste gerade Zahl rundet und im Ergebnis eine Doppelstunde als 14-tägig markiert. Auch wenn der 14-tägige Unterricht im Wechsel mit einem anderen Fach/Lehrer oder mit einer anderen Klasse stattfindet, lässt sich das mit einer Kopplung vergleichsweise einfach modellieren. Oder die „übrig gebliebenen“ einzelnen Unterrichtsstunden müssen im Doppelstundenraster untergebracht werden, indem jeweils zwei (von verschiedenen Fächern) einen Doppelstunden-Slot belegen – oder auch nur eine, wodurch eine Freistunde entsteht. Natürlich darf maximal eine Unterrichtsstunde pro Unterricht alleine vorkommen, weil ansonsten das gesamte Doppelstundensystem aufgelöst würde.

Ein CSP-Solver kann sich mit einer Statusvariable leicht merken, welche Unterrichtsstunde die alleinstehende ist. Für die Modellierung mit pseudo-booleschen Constraints sind jedoch zusätzliche (binäre) Variablen nötig: Wir definieren für jedes Zeitslot-Paar $(z_1, z_2) \in D$ und jeden Unterricht $u \in U$ mit ungerader Wochenstundenzahl $W(u)$ die Variablen

$$i \in \{1,2\} : \; e_{uz_i} = \begin{cases} 1 & \text{Die zugehörige Unterrichtsstunde } x_{uz_i} \text{ ist eine Einzelstunde} \\ 0 & \text{Sonst} \end{cases}$$

und ersetzen den bisherigen Constraint $x_{uz_1} = x_{uz_2}$ durch

$$x_{uz_1} \leqslant x_{uz_2} + e_{uz_1}$$
$$x_{uz_2} \leqslant x_{uz_1} + e_{uz_2}$$

Die aussagenlogische Schreibweise dieser Constraints verdeutlicht die Idee:

$$x_{uz_1} \rightarrow x_{uz_2} \vee e_{uz_1}$$
$$x_{uz_2} \rightarrow x_{uz_1} \vee e_{uz_2}$$

Wenn der Unterricht zum einen Zeitpunkt stattfindet, dann findet er auch zum anderen statt, oder es handelt sich um eine Einzelstunde.

Zusätzlich muss sichergestellt werden, dass maximal eine Einzelstunde stattfindet:

$$\sum_{z \in Z} e_{uz} \leqslant 1$$

und dass Einzelstunden nur stattfinden, wenn überhaupt Unterricht stattfindet:

$$\forall z \in Z : \; e_{uz} \leqslant x_{uz}$$

Hinweis: Die Entscheidung zur Einführung von zusätzlichen Variablen pro Unterricht und Zeitslot bedeutet nicht, dass sich die Gesamtzahl der Variablen in der Probleminstanz verdoppelt. Diese zusätzlichen Variablen sind nur für Zeitslots nötig, die zu Doppelstunden gehören, und vor allem nur für Unterrichte, die eine ungerade Stundenzahl haben. Unterrichte mit gerader Stundenzahl sind in der Praxis deutlich häufiger und lassen sich weiterhin ohne zusätzliche Variablen mit diesem Constraint zu Doppelstunden zusammenfassen:

$$x_{uz_1} = x_{uz_2}$$

Dieser Constraint lässt sich leicht in Klauselschreibweise umformen:

$$x_{uz_1} \leftrightarrow x_{uz_2} \Leftrightarrow (\overline{x_{uz_1}} \vee x_{uz_2}) \wedge (x_{uz_1} \vee \overline{x_{uz_2}})$$

7.2.3. Hohlstunden-Vermeidung

Für einen gegebenen Lehrer $l \in L$ soll die Anzahl an Hohlstunden minimiert werden. Es soll also möglichst wenige Zeitslots geben, die zwischen belegten Zeitslots desselben Tages liegen, aber für den Lehrer unterrichtsfrei sind. Weidler [2012] verwendet dafür ganzzahlige Variablen, die die Nummer der ersten und letzten Unterrichtsstunde an einem Tag markieren, und minimiert dann die Differenz.

Ein guter Ansatz, um auf nicht-binäre Variablen verzichten zu können, ist die Umkehrung des Geforderten: Wenige Hohlstunden sind gleichbedeutend mit vielen Stunden vor der ersten oder nach der letzten Unterrichtsstunde an einem Tag.

Wir definieren neue Variablen $\alpha_{lz}, \omega_{lz} \in \mathbb{B}$ für jeden Zeitslot $z \in Z$. Die Variablen α_{lz} sind vor der ersten Unterrichtsstunde des Lehrers wahr, also wenn er noch ausschlafen kann. Die Variablen ω_{lz} sind nach der letzten Unterrichtsstunde wahr, also am Feierabend.

Für jede Stunde des Tages gibt es auf den einzelnen Lehrer bezogen vier Möglichkeiten:

- Er hat noch frei (schläft noch), dann ist α_{lz} wahr.
- Er unterrichtet, dann $\exists u \in U_l : \ x_{uz} = 1$.
- Er hat schon frei (Feierabend), dann ist ω_{lz} wahr.
- Er hat eine Hohlstunde, dann ist keine der Variablen wahr.

Weil sich diese Möglichkeiten gegenseitig ausschließen, muss gelten:

$$\forall z \in Z : \ \alpha_{lz} + \omega_{lz} + \sum_{u \in U_l} x_{uz} \leqslant 1$$

Außerdem müssen für eine freie Morgenstunde auch alle Stunden davor frei sein, analog müssen nach einer Feierabendstunde alle folgenden Stunden frei sein:

$$\forall t \in T, \ 2 \leqslant i \leqslant |S| : \ \alpha_{lz_{ti}} \leqslant \alpha_{lz_{ti-1}}$$
$$\forall t \in T, \ 1 \leqslant i \leqslant |S| - 1 : \ \omega_{lz_{ti}} \leqslant \omega_{lz_{ti+1}}$$

Das Ziel ist die Vermeidung von Hohlstunden, was mit einer Maximierung der Zahl der freien Randstunden gleichzusetzen ist. Es gilt also, diesen Term zu maximieren:

$$\sum_{z \in Z} (\alpha_{lz} + \omega_{lz})$$

7.2.4. Weitere Bedingungen

Für die weiteren Bedingungen „Fach pro Tag“, „Harte Fächer vormittags“, „Folgen harter Fächer“ und „Lehrer-Nichtverfügbarkeiten“ gibt Weidler [2012] bereits Modellierungen in binärer Form an, die direkt als pseudo-boolesche Constraints formuliert werden können.

7.3. Implementierung und Optimierung

Für die Lösung der PB-Instanzen wurden Solver ausgewählt, die auf SAT basieren. Sowohl Sat4J Pseudo 2.3.5 [Le Berre und Parrain, 2010] als auch die Solver WBO 2.0 [Manquinho et al., 2009] und PWBO 2.2 [Martins et al., 2011] gehörten in den letzten Jahren zu den besten PB-Solvern [Pseudo Boolean Competition, 2012]. Da PWBO nur Probleminstanzen mit Zielfunktion unterstützt, wurde für die Berechnung mit ausschließlich harten Bedingungen der Solver WBO verwendet, und für die weiteren Berechnungen stets PWBO. Zum Vergleich wurde der LP-basierte Solver SCIP 3.1.0 mit SoPlex 2.0.0 herangezogen [Achterberg, 2009].

7.3.1. Ausschließlich harte Bedingungen

Tabelle 7.1 zeigt die Ergebnisse der Berechnungen mit harten Bedingungen.

Tabelle 7.1. Laufzeit der Pseudo-Boolean-Solver mit harten Bedingungen

Datensatz	A	B	C	D	
Anzahl Variablen	156	1036	1512	8602	
Anzahl Constraints	115	500	733	3088	Konfliktfreiheit
Rechenzeit (Sat4J)	7 ms	16 ms	418 ms	43 ms	
Rechenzeit (WBO)	<10 ms	20 ms	80 ms	140 ms	
Rechenzeit (SCIP)	<10 ms	50 ms	560 ms	220 ms	
Anzahl Constraints	115†	600	933	3388	+ Kernstunden
Rechenzeit (Sat4J)	7 ms	51 ms	134 ms	68 ms	
Rechenzeit (WBO)	<10 ms	40 ms	60 ms	160 ms	
Rechenzeit (SCIP)	<5 ms	30 ms	100 ms	210 ms	
Anzahl Constraints	127†	645	1013	4394	+ Fixe Stunden
Rechenzeit (Sat4J)	7 ms	61 ms	37 ms	48 ms	
Rechenzeit (WBO)	<10 ms	50 ms	100 ms	130 ms	
Rechenzeit (SCIP)	<5 ms	30 ms	60 ms	220 ms	

† Der Datensatz A enthält keine Kernstunden

Hinweis: Die Rechenzeiten von (P)WBO und SCIP können lediglich auf 10 Millisekunden genau angegeben werden.

Alle Solver sind in der Lage, für alle Probleminstanzen innerhalb kurzer Zeit eine zulässige Lösung zu finden. Der Vergleich mit den CSP-Implementierungen in Abschnitt 5.3 und Abschnitt 5.4 zeigt, dass die PB-Solver trotz der schwierigeren Kernstunden-Bedingungen alle Datensätze lösen können, was den CSP-Solvern nur mit zusätzlichen Hilfestellungen wie einer gezielten Sortierung der Variablen (Unterabschnitt 5.4.3) oder mit Unterstützung durch die LP-Relaxierung (Kapitel 6) gelingt.

7.3.2. Optimierung mit weichen Bedingungen

Tabelle 7.2 zeigt die Ergebnisse der Berechnungen mit den oben genannten harten Bedingungen sowie einer Zielfunktion aus den weichen Bedingungen „Unterrichtspriorität“ und „Lehrer-Verfügbarkeiten“. Hinweis: Beim Vergleich mit den CSP-Berechnungen (siehe Unterabschnitt 6.7.2) ist zu beachten, dass die dort verwendeten Bedingungen leicht von den hier verwendeten abweichen, sodass auch der optimale Zielfunktionswert nicht immer gleich ist.

Tabelle 7.2. Laufzeit der Pseudo-Boolean-Solver mit harten und weichen Bedingungen

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Größe der Zielfunktion	140	480	397	5510
Anzahl Constraints	127	645	1013	4394
Optimum (Minimum)	-74	-54	-134	-348
Beste Lösung (Sat4J)	-74	-35	-128	-267
Gefunden nach (Sat4J)	5,6 s	206 s	59 min	335 ms
Terminiert nach (Sat4J)	11,4 s	>1 h	>1 h	52 min[†]
Beste Lösung (PWBO)	-74	-30	-122	-46
Gefunden nach (PWBO)	0,9 s	80 ms	170 ms	2,2 s
Terminiert nach (PWBO)	2,5 s	>1 h	>1 h	>1 h
Beste untere Schranke (PWBO)	-	-558	-528	-1766
Beste Lösung (SCIP)	-74	-54	-134	-348
Gefunden nach (SCIP)	<40 ms	<40 ms	100 ms	5,2 s
Terminiert nach (SCIP)	40 ms	40 ms	370 ms	116,0 s

[†] Abbruch wegen Speicherüberlauf

Nur der vergleichsweise kleine Datensatz A kann von allen Solvern optimal gelöst werden. Bei den anderen Datensätzen zeigt sich ein deutlicher Unterschied zwischen den SAT-basierten PB-Solvern, die keinen einzigen Datensatz lösen können, und dem LP-basierten Solver SCIP, der alle in akzeptabler Zeit löst.

7.4. Einbezug der LP-Relaxierung

Die Beobachtung, dass der LP-basierte Solver wesentlich erfolgreicher ist als die reinen PB-Solver, legt nahe, die LP-Relaxierung einzubeziehen – analog zu dem in Kapitel 6 vorgestellten CSP-Ansatz. Da die Variablen in der PB-Modellierung keine Reihenfolge haben, in der sie abgearbeitet werden, kommt nur die Fixierung von ganzzahligen Werten in Frage. Weil die Fixierung von 0-Werten eine Probleminstanz unlösbar machen kann (siehe Unterabschnitt 6.3.1), werden hier nur die 1-Werte fixiert, d. h. sie werden als Axiome in die Probleminstanz aufgenommen.

Die Berechnungen wurden mit den harten Bedingungen „Kernstunden“, „Fixe Stunden“ und „Lehrer-Nichtverfügbarkeiten“ sowie den weichen Bedingungen „Unterrichtspriorität“ und „Lehrer-Verfügbarkeiten“ durchgeführt und die Ergebnisse sind in Tabelle 7.3 dargestellt. Zu den Ergebnissen bei Datensatz C sei angemerkt, dass der Erfolg bei beiden Solvern von der gewählten Lösung der LP-Relaxierung abhängt.

Tabelle 7.3. Laufzeit der Pseudo-Boolean-Solver mit LP-Relaxierung

Datensatz	A	B	C	D
Anzahl Variablen	156	1036	1512	8602
Anzahl fixierter Variablen	6	114	194	97
Größe der Zielfunktion	140	480	397	5510
Anzahl Constraints	131	759	1207	4476
Optimum (Minimum)	-74	-54	-134	-348
Beste Lösung (Sat4J)	-74	-54	-134	-300
Gefunden nach (Sat4J)	14 ms	22 ms	55 ms	429 ms
Terminiert nach (Sat4J)	160 ms	27 ms	>1 h	>1 h
Beste Lösung (PWBO)	-74	-54	-134	-326
Gefunden nach (PWBO)	230 ms	30 ms	770 ms	337,4 s
Terminiert nach (PWBO)	360 ms	30 ms	52,6 s	>1 h
Beste untere Schranke (PWBO)	-	-	-	-1166

Generell ist die Berechnung durch die Fixierung der Einsen aus der LP-Relaxierung deutlich schneller und erfolgreicher. Im Vergleich mit den CSP-Berechnungen (siehe Unterabschnitt 6.7.2) zeigt sich außerdem, dass die PB-Solver schneller terminieren, weil sie den Optimalitätsbeweis mit Resolution (oder mit anderen Techniken) führen können und nicht den gesamten verbleibenden Suchbaum durchsuchen müssen. Für Datensatz D, der nicht innerhalb der gegebenen Zeit gelöst werden kann, finden die PB-Solver in kürzerer Zeit bessere Lösungen als die CSP-Solver. Ohne Optimierung, dafür aber mit der Doppelstunden-Bedingung, kann für Datensatz D in weniger als einer Sekunde eine zulässige Lösung gefunden werden, was den CSP-Solvern auch nach einer Stunde nicht gelungen ist. Auch mit zufällig ausgewählten (optimalen) Lösungen der LP-Relaxierung bleiben diese Unterschiede bestehen, sodass ausgeschlossen werden kann, dass die oben verwendete deterministisch gewählte Lösung per Zufall den PB-Solvern eine bessere Ausgangssituation bietet. Das zeigt, dass der PB-Ansatz nicht so stark auf wachsende Probleminstanzen reagiert wie der CSP-Ansatz.

Genau wie die CSP-Solver profitieren auch die PB-Solver nicht (messbar) von den zusätzlichen Cliquen-Constraints. Diese werden daher nur zur Verbesserung der LP-Relaxierung eingesetzt.

7.5. Auswertung

Auch wenn die PB-Implementierung in fast allen Fällen erfolgreicher ist als die CSP-Implementierung, bleibt sie noch deutlich hinter den Ergebnissen zurück, die der kommerzielle Solver Gurobi mit der ILP-Modellierung erzielt. Die Berechnung mit dem großen Datensatz D benötigt zu viel Zeit. Zwar gelingt es PWBO, innerhalb von einer Stunde eine

knapp akzeptable Lösung zu finden, doch die obigen Berechnungen wurden nicht mit allen Bedingungen durchgeführt. Zusätzliche Bedingungen führen zwar nicht zu einer exponentiell längeren Laufzeit, doch selbst wenn sich die Laufzeit nur um einen Faktor vergrößert, wäre die resultierende Laufzeit mit dem hier vorgestellten Ansatz für die Praxis zu groß.

Mit den zusätzlichen Bedingungen „Fach-pro-Tag“ und „Doppelstunden“ dauert die Berechnung für Datensatz D bereits erheblich länger. Nach einer Stunde ist ein Zielfunktionswert von -109 erreicht, das Optimum läge bei -270. Während Gurobi binnen zwei Minuten (anstatt wenigen Sekunden) in der Lage ist, das Optimum trotz dieser zusätzlichen Bedingungen zu berechnen, steigt die Laufzeit bei den (freien, kostenlosen) PB-Solvern von Stunden auf Tage an. Dabei handelt es sich nicht notwendigerweise um einen grundsätzlichen Mangel des PB-Ansatzes oder der PB-Solver, aber es fehlt noch an weiteren Verbesserungen, um auch bei größeren Datensätzen mit allen gewünschten Bedingungen praktikable Laufzeiten zu erhalten.

Für kleinere (Grund-)Schulen könnte das in dieser Arbeit vorgestellte Verfahren bereits eingesetzt werden. Um zu lange Laufzeiten und insbesondere den zeitintensiven Optimalitätsbeweis am Ende zu vermeiden, könnte man die Berechnung abbrechen, sobald mehr als eine Stunde lang keine bessere Lösung gefunden wurde. Die bis dahin gefundenen Lösungen sind bereits gut genug (im Beispiel von Datensatz A bis C sogar schon optimal), um verwendet werden zu können. Weitere Verbesserungen, die in der Praxis auch bemerkbar wären, können eher durch eine Verbesserung der Modellierung und insbesondere der Zielfunktion erreicht werden, als durch eine weitere Annäherung an das theoretische Optimum um wenige Prozent.

Kapitel 8

Zusammenfassung, Anmerkungen und Ausblick

Dieses Kapitel gibt in Abschnitt 8.1 eine Zusammenfassung über die in dieser Arbeit vorgestellten Lösungsansätze und Ergebnisse. Die Abschnitte 8.2 und 8.3 enthalten Vergleiche mit anderen Ansätzen und Solvern. Abschnitt 8.4 dokumentiert das Verfahren für die Zeitmessungen in dieser Arbeit. In den Abschnitten 8.5 bis 8.8 wird ein Ausblick auf mögliche zukünftige Entwicklungen gegeben.

8.1. Zusammenfassung

Es wurden drei Ansätze zur Lösung des Stundenplanproblems untersucht: der erste mit ganzzahligen Variablen in einem CSP, der zweite mit binären Variablen und der dritte mit pseudo-boolescher Optimierung. Die letzteren beiden Ansätze wurden mit Hilfe der LP-Relaxierung verbessert.

Der erste Lösungsansatz konnte vom `alldifferent`-Constraint und von der geringen Variablenzahl profitieren. Solange nur die Konfliktfreiheits-Bedingung erfüllt werden musste, gelang es damit in wenigen Millisekunden, zulässige Lösungen zu finden. Mit Hilfe einer gezielten Sortierung der Zeitslots und der Unterrichte konnte auch die Kernstunden-Bedingung erfüllt werden. Die Modellierung der meisten anderen Bedingungen und insbesondere das Aufstellen einer Zielfunktion, die jede Unterricht-Zeitslot-Belegung unterschiedlich gewichten muss, erfordern jedoch eine große Menge an binären Statusvariablen, die den Ansatz an seine Grenzen bringen.

Deswegen wurden die binären Variablen im zweiten Ansatz direkt als Hauptvariablen des CSP gewählt. Diese Art der Modellierung erzielte zunächst ähnliche Ergebnisse wie die vorherige. Als zusätzlicher Vorteil konnte die LP-Relaxierung einbezogen werden, um die Suche weiter zu beschleunigen. Es wurde untersucht, wie unterschiedliche Varianten des Rundens oder des Fixierens von Werten die Lösbarkeit der Probleminstanz beeinträchtigen

und wie sie die Berechnung beschleunigen. Dabei stellte sich heraus, dass insbesondere die 1-Werte aus der LP-Relaxierung als zuverlässig angesehen werden können, nicht jedoch die (deutlich häufigeren) 0-Werte. Um auch von den Zwischenwerten zu profitieren, ohne sie runden zu müssen, wurde die LP-Relaxierung als Sortierreihenfolge der CSP-Variablen eingesetzt. In Kombination mit einer heuristischen Verfeinerung der Sortierung der Variablen und einer Verschärfung der LP-Relaxierung durch Cliquen-Constraints konnten so Ergebnisse erzielt werden, die vergleichbar mit der ersten Modellierung sind. Anders als bei jener können aber bei der binären Modellierung mit LP-Unterstützung weitere Bedingungen eingefügt werden, ohne dass die Laufzeit extrem ansteigt.

Komplexere Bedingungen wie die Doppelstunden, vor allem aber der eng gelagerte Datensatz C bringen auch diesen CSP-Ansatz an seine Grenzen. Da für die meisten schwierigeren Bedingungen eine Heuristik zur gezielteren Suche gefunden werden muss, lassen sich mehrere Bedingungen nur schwer kombinieren. Außerdem ist die Suche nach dem Optimum mit dieser Methode relativ langsam, weil zum Schluss der gesamte verbleibende Suchbaum durchsucht werden muss, um sicher zu sein, dass es keine bessere Lösung gibt.

Aufgrund dessen wurde für den dritten Ansatz ein auf SAT basierendes Verfahren gewählt, sodass der Optimalitätsbeweis mit effizienteren Methoden (wie Resolution) geführt werden kann. Ein weiterer Vorteil ist, dass bei der pseudo-booleschen Optimierung nur binäre Variablen und pseudo-boolesche Constraints möglich sind und daher auch nur diese vom Solver unterstützt werden müssen, was für das Stundenplanproblem ausreicht. Mit diesem Ansatz konnten in vergleichbarer Zeit zulässige Lösungen gefunden werden und die nach einer Stunde erreichten Zielfunktionswerte waren durchgehend besser als beim CSP-Ansatz. Vor allem aber terminierten die PB-Solver nach dem Finden der optimalen Lösung erheblich schneller als die CSP-Solver, d. h. sie konnten wie erwartet den Optimalitätsbeweis schneller durchführen.

Insgesamt ist es dennoch mit keiner Methode in Kombination mit den genannten Solvern gelungen, den Datensatz D optimal zu lösen. Dem freien (nicht kommerziell nutzbaren) Solver SCIP gelingt das zwar sowohl mit der ILP- als auch mit der PB-Modellierung des Problems, doch wenn weitere Bedingungen – insbesondere die Doppelstunden – hinzugefügt werden, benötigt auch SCIP relativ lange für die Berechnung.

8.2. Vergleich mit spezialisierten Algorithmen

Wie bereits in Kapitel 3 erwähnt, gibt es viele Algorithmen und Herangehensweisen, die speziell für das Stundenplanproblem entwickelt oder angepasst wurden. Viele der neueren bauen auf der Problemdefinition von Post et al. [2012] auf, die als sehr allgemein gehaltenes XML-Format gegeben ist.

Manche dieser Algorithmen verwenden einen Setz-Algorithmus (siehe Abschnitt 3.2), um zunächst eine zulässige Lösung zu erhalten. Andere bauen diese aus kleineren Teillösungen zusammen (z. B. KHE) oder verwenden ein randomisiertes Verfahren. Mitunter werden sogar harte Bedingungen relaxiert, damit überhaupt eine erste Lösung gefunden werden kann. Diese wird dann mit unterschiedlichen Methoden verbessert bzw. überhaupt erst in eine zulässige Lösung überführt.

Die naheliegende Idee, mittels lokaler Suche zwei oder drei Unterrichte zu vertauschen, um neue Lösungen zu finden, ist dabei oft nicht zielführend. Kingston [2013] nennt dafür zwei Gründe: Zum einen sind die Probleminstanzen üblicherweise so groß, dass die zu reparierenden Problemstellen zu weit auseinander liegen, als dass sie mit einer lokalen Operation behoben werden könnten. Zum anderen ist schnell ein Zustand erreicht, in dem alle möglichen lokalen Operationen bereits erfolglos durchprobiert wurden, weil jede direkte Nachbar-Lösung nicht besser als die aktuelle Lösung ist.

Um entfernte, scheinbar unabhängige Problemstellen einer Lösung gemeinsam beheben zu können, sind also andere Methoden nötig. Ein möglicher Ansatz ist, mittels Simulated Annealing dennoch einige Nachbarlösungen zu explorieren, auch wenn diese nicht unmittelbar besser sind [Fonseca et al., 2012]. Meyers und Orlin [2007] betrachten hingegen eine größere Nachbarschaft beim Verändern der Lösungen. Mit GELATO [Cipriano et al., 2013], einem Tool das auf Gecode aufbaut, ließe sich das Stundenplanproblem mit einer Kombination aus Constraint Programming und Large Neighborhood Search lösen.

Ähnlich ist der Ansatz von Kingston [2013], der mehrere lokale Operationen so aneinander reiht, dass sich am Ende der Kette eine Verbesserung ergibt. Diese Idee wurde – neben vielen weiteren Optimierungen – in KHE implementiert [Kingston, 2014]. Probeweise wurden die SchulScheduler-Instanzen in das XML-Format von Post et al. [2012] konvertiert, wobei nur die Konfliktfreiheit und die Kernstunden als Bedingungen verwendet wurden. Der aktuelle KHE-Solver (vom Mai 2014) terminiert für den größten Datensatz D in weniger als einer Minute, gibt jedoch eine unzulässige Lösung aus, bei der die Kernstunden-Bedingung an einer Stelle verletzt wird.

Im Vergleich mit den speziell für das Stundenplanproblem entwickelten Algorithmen, die oft (leicht) unzulässige Lösungen tolerieren und mit lokaler Suche arbeiten, haben die in dieser Arbeit vorgestellten Ansätze (lineare Programmierung, Constraint Programming und pseudo-boolesche Optimierung) den Vorteil, dass die gefundenen Lösungen stets zulässig und außerdem beweisbar optimal sind. Ein weiterer Vorteil ist der geringere Entwicklungs- und Wartungsaufwand: Die Modellierung als ILP oder CSP nimmt erheblich weniger Zeit in Anspruch und erlaubt es, Änderungswünsche von Schulen schnell umzusetzen.

Ein klarer Nachteil sind die längeren Rechenzeiten. Im Vergleich mit KHE benötigen die für diese Arbeit verwendeten Solver Or-Tools, Gecode, Sat4J und (P)WBO deutlich länger

für die gleiche Probleminstanz, und in einigen Fällen terminieren sie überhaupt nicht in akzeptabler Zeit. In der Praxis relativiert sich dieser Nachteil jedoch, da durch den Einsatz von kommerziellen Solvern wie Gurobi oder SCIP die Rechenzeit vergleichbar gut ist. Hingegen hat die Tatsache, dass die gefundenen Lösungen dann ausnahmslos zulässig und optimal sind, in der Praxis eine große Relevanz.

8.3. Weitere getestete Solver

Neben den hauptsächlich verwendeten Solvern Or-Tools, Gecode, Sat4J und (P)WBO wurden einige weitere Solver kurz getestet, die hier nur kurz erwähnt werden, weil sie entweder nicht besser als die verwendeten Solver abschnitten, oder weil sie kostenpflichtig sind. Alle Solver (außer KHE) wurden mit der binären Problemformulierung getestet.

- Gurobi 5.6.2 [Gurobi Optimization, Inc., 2014] löste alle verwendeten Instanzen in (oft deutlich) unter einer Stunde optimal und erzielte auch bei den vollständigen Instanzen aus SchulScheduler gute Ergebnisse (siehe Abschnitt 4.5).
- KHE14 (siehe oben) rechnete schnell, lieferte aber teilweise unzulässige Lösungen.
- LocalSolver 4.5 [Benoist et al., 2011] war durchweg deutlich langsamer als Gurobi, SCIP und auch als der in Abschnitt 7.4 vorgestellte Ansatz mit pseudo-boolescher Optimierung und LP-Relaxierung.
- Mistral 2.0 [Hebrard, 2014] berechnete zulässige Lösungen so schnell wie die anderen Solver, benötigte aber bereits für den kleinen Datensatz A länger als eine Stunde für den Optimalitätsbeweis der gefundenen optimalen Lösung. Bei den größeren Datensätzen waren die nach einer Stunde gefundenen Lösungen noch weit vom Optimum entfernt.
- Opturion CPX 1.0.2 [Opturion, 2014] rechnete erheblich langsamer als PBSugar und Sat4J.
- Sat4J Maxsat 2.3.5 [Le Berre und Parrain, 2010] war durchgehend etwas langsamer als Sat4J Pseudo.
- SCIP 3.1.0 [Achterberg, 2009] rechnete ähnlich gut wie Gurobi, benötigte jedoch ein Mehrfaches der Zeit. Da es sich bei den kleineren Datensätzen aber nur um Sekunden handelt, ist die Laufzeit für diese Instanzen trotzdem gut.
- Sugar 2.2.1 [Tamura et al., 2009] und PBSugar 1.1.1 [Tamura et al., 2013] mussten zur iterativen Optimierung für jede Iteration vollständig neu starten und rechneten etwas langsamer als Sat4J. Insbesondere die letzte Iteration (Optimalitätsbeweis) dauerte sehr lang.

Die hier genannten Ergebnisse erheben keinesfalls den Anspruch, repräsentativ für die Leistungsfähigkeit dieser Solver zu sein. Es handelt sich lediglich um kurze, wenig optimierte Tests mit bereits bestehenden Instanzen des Stundenplanproblems.

8.4. Messverfahren

Alle Berechnungen wurden auf einem gewöhnlichen Desktop-Rechner (Intel Core i7-920 mit 2,67 GHz und 6 GB RAM) unter Windows 7 durchgeführt. Für Solver, die lediglich Linux unterstützen, wurde Ubuntu 14.04 in einer virtuellen Maschine mit 2 Gigabyte Arbeitsspeicher verwendet. Das betrifft KHE, Mistral, PWBO und WBO.

Generell wurden lang laufende Berechnungen nach einer Stunde abgebrochen und die Rechenzeit mit >1 h angegeben, wenn eigentlich Berechnungsergebnisse im Bereich von Sekunden üblich waren. Bei sehr kurzen Rechenzeiten im Millisekundenbereich wurden die Berechnungen fünf bis zehn Mal ausgeführt und der Median aller Messungen angegeben.

Aufgrund der Vielzahl der verwendeten Solver mussten die Probleminstanzen aus dem SchulScheduler-Format in viele andere Formate konvertiert und für viele andere Schnittstellen aufbereitet werden. Da die dafür verwendeten Skripte auf Korrektheit und nicht auf Laufzeit optimiert wurden und Laufzeiten von mehreren Sekunden aufwiesen, wurde für alle Solver einheitlich nur die reine Berechnungszeit gemessen und nicht das Einlesen und Konvertieren der Probleminstanz. Auch Vorberechnungen wie die Suche aller Cliquen und das Ermitteln des fertigen Stundenplans aus den Berechnungsergebnissen sind von den Rechenzeiten stets ausgenommen. Die berechneten Stundenpläne wurden stichprobenartig auf Korrektheit überprüft.

8.5. Softwaretechnische Anmerkungen

Aus verschiedenen Gründen kann es notwendig sein, dieselbe Probleminstanz in unterschiedlichen Formen an verschiedene Solver zu übergeben: Einerseits lassen sich dadurch Solver vergleichen oder parallel ausführen. Vor allem aber ist es für die Berechnung der LP-Relaxierung nötig, deren Ergebnis dann in die weitere Berechnung einfließt.

Um den Code zur Generierung der Variablen und Constraints für die verschiedenen Solver nicht mehrfach implementieren und warten zu müssen, könnte in Zukunft auf eine allgemeinere Softwarearchitektur zurückgegriffen werden. Diese könnte entweder die SchulScheduler-Probleminstanz zuerst in ein eigenes Zwischenformat aus Variablen, Constraints und Zielfunktion übersetzen und anschließend an die verschiedenen Solver weitergeben. Oder es könnte eine verallgemeinerte Schnittstelle zur Kapselung der Solver

entworfen werden, sodass jede Bedingung nur ein Mal implementiert werden muss und trotzdem alle Solver-Typen ansteuern kann.

Eine Lösung oder zumindest eine gute Referenz dafür könnte die relativ neue Java Constraint Programming API (JSR-331) sein, die im Rahmen dieser Arbeit aber noch nicht verwendet wurde.

8.6. Umgang mit nicht lösbaren Probleminstanzen

Im Normalfall sind die in der Praxis vorkommenden Probleminstanzen lösbar. Vor allem wenn ein ähnlicher Stundenplan wie im Vorjahr erstellt werden soll, mit nur leichten Abweichungen bei Klassen und Lehrern, existiert ziemlich sicher eine Lösung. Dennoch kann es vorkommen, dass die Anforderungen nicht erfüllbar sind und die Schule beispielsweise ihre Lehrerzuweisung oder deren Verfügbarkeiten anpassen muss. Außerdem kann es durch einfache Tippfehler oder Verständnisprobleme in Bezug auf die Konzepte der Software dazu kommen, dass eine unlösbare Probleminstanz zum Solver gelangt. Viele Konflikte und Probleme können vorab durch sorgfältige Validierung der Daten in polynomieller Zeit und meist schon in Echtzeit während der Eingabe erkannt werden.

Für alle übrigen Fälle ist es wichtig, dass die Solver mit unlösbaren Probleminstanzen umgehen können. CSP- und PB-Solver können das von Haus aus, weil sie bei der Optimierung die Optimalität der letzten Lösung dadurch beweisen, dass die gleiche Probleminstanz mit Forderung nach einer besseren Lösung unlösbar ist. Jedoch ist zu diesem Zeitpunkt die Probleminstanz bereits relativ weit eingeschränkt und der Solver hat bereits einige Constraints verarbeiten können und so etwas über die Probleminstanz gelernt.

Die getesteten CSP- und PB-Solver melden einige unlösbare Probleminstanzen innerhalb von wenigen Millisekunden – vor allem wenn es sich um direkte Widersprüche handelt, wie etwa Klassen, die mehr Unterricht haben als es Zeitslots gibt. Andere unlösbare Probleminstanzen werden selbst nach mehreren Stunden nicht erkannt. Letzteres ist problematisch, weil der Nutzer glaubt, es würde eine Lösung berechnet, während beispielsweise ein CSP-Solver den kompletten Suchbaum durchsuchen muss, obwohl es keine Lösung gibt. Das dauert aber mit Sicherheit länger als das gesetzte Zeitlimit, sodass der Nutzer die Rückmeldung bekommt, dass seine Probleminstanz zu groß oder zu schwierig sei.

Es müssten also weitere Techniken entwickelt werden, um unlösbare Probleminstanzen möglichst schnell zu erkennen. Dafür kommen unter anderem (weitere) geschickte Validierungen, Heuristiken oder auch die LP-Relaxierung in Frage, deren Unlösbarkeit vergleichsweise schnell festgestellt werden kann. Einen besonderen Vorteil bieten Solver wie Gurobi, die eine minimale Menge von Constraints berechnen können, die zur Unlösbarkeit

führt. Die Software kann diese dann in menschenlesbare Bedingungen übersetzen und gruppieren, sodass dem Nutzer Hinweise gegeben werden können, durch welche seiner Eingaben die Probleminstanz unlösbar wird. In SchulScheduler ist eine solche Funktion bereits implementiert.

8.7. Lokale Suche und Large Neighborhood Search

Wie bereits in Abschnitt 8.2 angesprochen, ist die lokale Suche in einer *größeren* Nachbarschaft ein geeigneter Ansatz zur Verbesserung von Lösungen des Stundenplanproblems, der aber den Nachteil hat, dass die Exploration der Nachbarschaft meist explizit programmiert werden muss. Um die Vorteile der lokalen Suche mit der Flexibilität von Ansätzen, die lediglich eine einfache Modellierung des Problems erfordern, zu kombinieren, könnten (neue) Solver evaluiert werden, die wie LocalSolver eine generische und heuristische Implementierung für lokale Suche anbieten.

8.8. Runden der LP-Relaxierung

Für andere Problemtypen finden sich in der Literatur viele Ansätze, um die Lösung des relaxierten linearen Programms so zu runden, dass die damit gefundene ganzzahlige Lösung zulässig ist und nur wenig vom Optimum abweicht [siehe Vazirani, 2001]. In Unterabschnitt 6.3.1 wurde gezeigt, dass das beim Stundenplanproblem nur schwer möglich ist, weil es reelle Lösungen gibt, deren ganzzahliger Teil die Probleminstanz bereits unlösbar macht. Selbst mit den in Abschnitt 6.6 vorgestellten Cliquen-Constraints wäre das Problem durch normales Runden nicht immer lösbar: Durch das Fixieren aller ganzzahligen Werte wird der Datensatz C in 20 % und der Datensatz D sogar in 59 % der Fälle unlösbar.

Dennoch ist die LP-Relaxierung für das Stundenplanproblem äußerst interessant. Ein Vergleich der Zielfunktionswerte von relaxierter und ganzzahliger Lösung (siehe Tabelle 6.1) ergibt, dass diese nur wenig bis gar nicht voneinander abweichen. Zudem kam es bei den Berechnungen im Rahmen dieser Arbeit nie vor, dass eine der Einsen aus der LP-Relaxierung falsch gewesen wäre, wenn diese mit Cliquen-Constraints berechnet wurde.

Möglicherweise könnte ein Verfahren gefunden werden, das die LP-Relaxierung geschickt rundet oder mehrfach rundet. Ebenfalls denkbar wären weitere Verschärfungen der LP-Relaxierung, die verhindern, dass die Fixierung von Nullen die Probleminstanz unlösbar macht. Ein weiterer Ansatzpunkt zur Verbesserung der LP-Relaxierung ist ihre Zielfunktion. Diese könnte gezielt so perturbiert werden, dass die gefundene reelle Lösung die genannten Probleme nicht mehr aufweist.

8.9. Fazit

Lösungsverfahren, die das Stundenplanproblem nicht direkt lösen, sondern es auf CSP, ILP oder ähnliches reduzieren, sind durch die Verfügbarkeit von schnellen Solvern und Computern als ebenbürtige Konkurrenz zu spezialisierten Algorithmen anzusehen und sind zudem besser wartbar und flexibler anpassbar. In dieser Arbeit wurde das Stundenplanproblem in der von SchulScheduler verwendeten Form als CSP, als pseudo-boolesches Optimierungsproblem und als binäres lineares Programm modelliert. Wenn offene und kostenlose Solver verwendet werden, sind die vorgestellten Ansätze zwar nur für kleinere Schulen ausreichend performant, doch für den praktischen Einsatz an größeren Schulen könnte auch auf kommerzielle Solver zurückgegriffen werden. Mit weiteren Verbesserungen der Modellierung und der Solver ist es mit hoher Wahrscheinlichkeit möglich, auch größere Datensätze schnell genug zu lösen. Dabei ist der pseudo-boolesche Ansatz aufgrund der Arbeitsweise der Solver erfolgversprechender als die CSP-Modellierung.

Die Ergebnisse der Arbeit zeigen, dass vor allem die LP-Relaxierung der gewählten binären LP-Modellierung sehr aussagekräftig und damit zielführend ist. Sie kann außerdem mit verschiedensten Ansätzen kombiniert werden. Für die weitere Forschung in diesem Bereich sind daher vor allem die LP-Relaxierung und außerdem die lokale Suche vielversprechend.

Literaturverzeichnis

[Achterberg 2009] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, Juli 2009.

[Benoist et al. 2011] T. Benoist, B. Estellon, F. Gardi, R. Megel, und K. Nouioua. LocalSolver 1.x: A black-box local-search solver for 0-1 programming. *4OR*, 9(3):299–316, 2011. URL http://www.localsolver.com/.

[Boros und Hammer 2002] E. Boros und P. L. Hammer. Pseudo-boolean optimization. *Discrete Appl. Math.*, 123(1-3):155–225, Nov. 2002.

[Breslaw 1976] J. A. Breslaw. A linear programming solution to the faculty assignment problem. *Socio-Economic Planning Sciences*, 10(6):227–230, 1976.

[Bron und Kerbosch 1973] C. Bron und J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, Sept. 1973.

[Burke et al. 2004] E. K. Burke, G. Kendall, M. Mısır, und E. Özcan. Applications to timetabling. In *Handbook of Graph Theory, chapter 5.6*, Seiten 445–474. Chapman Hall/CRC Press, 2004.

[Carter und Laporte 1998] M. W. Carter und G. Laporte. Recent developments in practical course timetabling. In *Selected Papers from the Second International Conference on Practice and Theory of Automated Timetabling II*, PATAT '97, Seiten 3–19, London, UK, 1998. Springer-Verlag.

[Chai und Kuehlmann 2005] D. Chai und A. Kuehlmann. A fast pseudo-boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3): 305–317, März 2005.

[Cipriano et al. 2013] R. Cipriano, L. Di Gaspero, und A. Dovier. A multi-paradigm tool for large neighborhood search. In E.-G. Talbi, Hrsg., *Hybrid Metaheuristics*, Band 434 von *Studies in Computational Intelligence*, Seiten 389–414. Springer Berlin Heidelberg, 2013.

[Dechter 2003] R. Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[Even et al. 1975] S. Even, A. Itai, und A. Shamir. On the complexity of time table and multi-commodity flow problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, Seiten 184–193, 1975.

[Eén und Sörensson 2006] N. Eén und N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[Fonseca et al. 2012] G. Fonseca, S. Brito, und H. Santos. A simulated annealing based approach to the high school timetabling problem. In H. Yin, J. Costa, und G. Barreto, Hrsg., *Intelligent Data Engineering and Automated Learning - IDEAL 2012*, Band 7435 von *Lecture Notes in Computer Science*, Seiten 540–549. Springer Berlin Heidelberg, 2012.

[Gecode Team 2006] Gecode Team. Gecode: Generic constraint development environment, 2006. URL http://www.gecode.org/.

[Gotlieb 1962] C. C. Gotlieb. The construction of class-teacher time-tables. In *IFIP Congress*, Seiten 73–77, 1962.

[Grötschel et al. 1988] M. Grötschel, L. Lovász, und A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, Band 2 von *Algorithms and Combinatorics*. Springer, 1988.

[Gruber & Petters 2014] Gruber & Petters. Untis Express Benutzerhandbuch, 2014. URL http://downloads.grupet.at/downloads/int/HelpManuals/DE/Light.pdf.

[Gurobi Optimization, Inc. 2014] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2014. URL http://www.gurobi.com/.

[Hebrard 2014] E. Hebrard. Mistral 2.0, 2014. URL http://homepages.laas.fr/ehebrard/mistral.html.

[Kingston 2013] J. H. Kingston. Repairing high school timetables with polymorphic ejection chains. *Annals of Operations Research*, Seiten 1–16, 2013.

[Kingston 2014] J. H. Kingston. The KHE High School Timetabling Engine, 2014. URL http://sydney.edu.au/engineering/it/~jeff/khe/.

[Le Berre und Parrain 2010] D. Le Berre und A. Parrain. The Sat4J library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[López-Ortiz et al. 2003] A. López-Ortiz, C.-G. Quimper, J. Tromp, und P. Van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *IJCAI*, Band 3, Seiten 245–250, 2003.

[Manquinho und Marques-Silva 2006] V. Manquinho und J. P. Marques-Silva. On using cutting planes in pseudo-boolean optimization. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:209–219, 2006.

[Manquinho et al. 2009] V. Manquinho, J. Marques-silva, und J. Planes. Algorithms for weighted boolean optimization. In *In SAT'09*, Seiten 495–508, 2009.

[Martins et al. 2011] R. Martins, V. Manquinho, und I. Lynce. Parallel Search for Boolean Optimization. In *RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, 2011.

[Meyers und Orlin 2007] C. Meyers und J. Orlin. Very large-scale neighborhood search techniques in timetabling problems. In E. Burke und H. Rudová, Hrsg., *Practice and Theory of Automated Timetabling VI*, Band 3867 von *Lecture Notes in Computer Science*, Seiten 24–39. Springer Berlin Heidelberg, 2007.

[Opturion 2014] Opturion. CPX discrete optimizer, 2014. URL http://www.opturion.com/cpx.html.

[Or-Tools Team 2010] Or-Tools Team. or-tools: Operations research tools developed at Google, 2010. URL https://code.google.com/p/or-tools/.

[Pillay 2013] N. Pillay. A survey of school timetabling research. *Annals of Operations Research*, Seiten 1–33, 2013.

[Post et al. 2012] G. Post, S. Ahmadi, S. Daskalaki, J. Kingston, J. Kyngas, C. Nurmi, und D. Ranson. An XML format for benchmarks in high school timetabling. *Annals of Operations Research*, 194(1):385–397, 2012.

[Post et al. 2013] G. Post, L. Di Gaspero, J. Kingston, B. McCollum, und A. Schaerf. The third international timetabling competition. *Annals of Operations Research*, Seiten 1–7, 2013.

[Pseudo Boolean Competition 2012] Pseudo Boolean Competition, 2012. URL `http://www.cril.univ-artois.fr/PB12/`.

[Schaerf 1999] A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

[Schmidt und Ströhlein 1980] G. Schmidt und T. Ströhlein. Timetable construction – an annotated bibliography. *The Computer Journal*, 23(4):307–316, 1980.

[Schulte und Tack 2013] C. Schulte und G. Tack. Programming propagators. In C. Schulte, G. Tack, und M. Z. Lagerkvist, Hrsg., *Modeling and Programming with Gecode*. 2013. Bezieht sich auf Gecode 4.2.1.

[Schöning 2003] U. Schöning. *Theoretische Informatik - kurzgefasst*. Spektrum Akademischer Verlag, 4. Auflage, 2003.

[Stuckey et al. 2010] P. J. Stuckey, R. Becket, und J. Fischer. Philosophy of the MiniZinc challenge. *Constraints - An International Journal*, 15(3):307–316, Juli 2010.

[Tamura et al. 2009] N. Tamura, A. Taga, S. Kitagawa, und M. Banbara. Compiling finite linear CSP into SAT. *Constraints - An International Journal*, 14(2):254–272, 2009.

[Tamura et al. 2013] N. Tamura, M. Banbara, und T. Soh. Compiling pseudo-boolean constraints to SAT with order encoding. In *IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, Seiten 1020–1027, Nov. 2013.

[Tillett 1975] P. Tillett. An operations research approach to the assignment of teachers to courses. *Socio-Economic Planning Sciences*, 9(3-4):101–104, 1975.

[Tomita et al. 2006] E. Tomita, A. Tanaka, und H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science - Computing and Combinatorics*, 363(1):28–42, Okt. 2006.

[Tsukiyama et al. 1977] S. Tsukiyama, M. Ide, H. Ariyoshi, und I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977.

[van Hoeve 2001] W. J. van Hoeve. The alldifferent constraint: A survey, 2001.

[van Omme et al. 2014] N. van Omme, L. Perron, und V. Furnon. or-tools user's manual. Technical report, Google, 2014.

[Vazirani 2001] V. V. Vazirani. *Approximation Algorithms*. Springer US, New York, NY, USA, 2001.

[Weidler 2012] V. Weidler. Stundenplanung als ganzzahliges lineares Optimierungsproblem. Universität Stuttgart, Juli 2012.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift