Hochschule für Angewandte Wissenschaften München
Fakultät 03 für Maschinenbau, Fahrzeugtechnik, Flugzeugtechnik

Optimierung großer Planungsprobleme unter Verwendung individueller Heuristiken

Am Beispiel eines Prüfungsplans

Bachelorarbeit

Georg Michael Puntigam

2. November 2020

Betreuer: Prof. Dr. Christian Möller

Erklärung zur Abschlussarbeit

Hiermit wird erklärt, dass die Arbeit mit obigem Thema selbständig verfasst und noch nicht anderweitig für Prüfungszwecke vorgelegt wurde. Weiterhin sind keine anderen als die angegebenen Quellen oder Hilfsmittel verwendet und wörtliche sowie sinngemäße Zitate als solche gekennzeichnet worden.

München, den 02.11.2020

Abstract

This thesis explores the possibilities of optimizing big planning problems. The dense exam schedule of the students at the Hochschule für angewandte Wissenschaften München will provide a practical example and data. The goal is to spread out the exam times for each student as evenly as possible within the limited time frame while respecting the constraints such as capacity and pre-planned exams. The main topics in this thesis will be modelling the problem as a mathematical optimization formula, applying constraints and looking for possibilities to sub-devide the problem for performance boosting without worsening the overall results too much. Several models for the problem will be formulated in Python and their results compared.

Zusammenfassung

Diese Arbeit befasst sich mit der Optimierung von großen Planungsproblemen. Der dicht gepackte Prüfungsplan der Hochschule für angewandte Wissenschaften München bietet dabei ein konkretes Beispiel und liefert die nötigen Daten. Das Ziel ist es, die Prüfungen für jeden Studenten zeitlich möglichst gleichmäßig entfernt voneinander zu legen. Dabei müssen Restriktionen wie die Kapazität der Räume und Prüfungen mit festen Terminen unbedingt eingehalten werden. Der Hauptteil der Arbeit besteht darin, die Aufgabe als ein mathematisches Optimierungsproblem mit Randbedingungen zu formulieren und Möglichkeiten zur Unterteilung des Problems zu suchen, die eine Laufzeitverbesserung ergeben, ohne die Qualität des Ergebnisses wesentlich zu beeinflussen. Verschiedene Modelle werden in Python formuliert und deren Ergebnisse verglichen.

Inhaltsverzeichnis

1 Einleitung

Planungs- und Optimierungsprobleme gibt es seit geraumer Zeit. Es gibt Probleme, mit so vielen potentiellen Lösungsmöglichkeiten, dass eine händische Evaluierung der verschiedenen Szenarien nicht möglich ist. Auch mit Hilfe von Computern ist es nicht möglich, die gesamte Anzahl der Szenarien mittels Brute-Force zu analysieren, da die Anzahl der Lösungsmöglichkeiten bei solchen Problemen die Anzahl der Partikel im bekannten Universum übertrifft. Seit Mitte des zwanzigsten Jahrhunderts wurde deswegen intensiv an den Problemen geforscht und erste Lösungsalgorithmen entwickelt. Die bekanntesten Vertreter dieser Algorithmen werden als Lineare Programme bezeichnet. Auch die Prüfungsplanung lässt sich als ein Optimierungsproblem formulieren.

Dabei handelt es sich bei diesem Problem um einen Spezialfall, in dem alle Lösungswerte ganze Zahlen annehmen müssen. Diese werden als ganzzahlige Optimierungsprobleme bezeichnet. Ähnliche Probleme sind zum Beispiel die wöchentliche Schichtplanung in einer Firma[7] und die Tourenplanung (Traveling Salesman), welche die kürzeste Rundreise zwischen einer Anzahl von Orten sucht. Einige dieser Probleme wurden bereits ausführlich untersucht, da deren Lösung fast immer mit finanziellen Vorteilen für die jeweiligen Firmen einhergehen. Oftmals sind Optimierungsprobleme in der Praxis aber auch so kompliziert, dass es nicht offensichtlich ist, ob es überhaupt eine Lösung gibt, die alle Anforderungen erfüllt.

An der Hochschule für angewandte Wissenschaften München ist die Prüfungsphase der Fakultät 03 etwa drei Wochen lang. In diesem Zeitraum gilt es über 80 Prüfungen zu planen für mehr als 1500 Studierende. Es gibt eine gigantische Anzahl von Möglichkeiten, die Prüfungen anzuordnen. Da der Prüfungszeitraum sehr kurz ist, soll nun jedem Studenten ein Prüfungsplan mit möglichst langen Pausen zwischen den einzelnen Prüfungen ermöglicht werden, um die Belastung nicht zusätzlich zu steigern. Wie können also zu diesem Zweck die Prüfungen bestmöglich anordnet werden?

In der Arbeit werden nun zunächst die Anforderungen des Problems genauer untersucht und die Daten analysiert. Anschließend werden verschiedene Lösungsmodelle vorgestellt und die Randbedingungen des Problems mathematisch ausgeschrieben. Diese Lösungsmodelle werden dann in Programmcode übersetzt. Zum Schluss werden die Ergebnisse dargestellt und untersucht.

2 Vorwort zur Notation

Zur Modellierung des Problemes wird es nötig sein, einige Datenfelder und Variablen einzuführen. Da es allgemein keine einheitlichen Normen dazu gibt, soll hier kurz die Notation in dieser Arbeit erläutert werden.

- einzelne große Buchstaben stellen eine Menge dar, zum Beispiel T, die Menge der Tage
- einzelne kleine Buchstaben stellen ein Element aus einer Menge dar, zum Beispiel Tag t aus Menge der T
- Eckige klammern weisen auf einen Vektor oder ein Datenfeld hin, zum Beispiel `normal_weeks`$[s, w]$, ein Datenfeld der Dimension $s \times w$
- Angaben zu den Dimensionen der Datenfelder beziehen sich auf die Tupel aller Elemente der Mengen, zum Beispiel $\forall s \in S$; $\forall w \in W$ bezieht sich auf alle Tupel des kartesischen Produktes der Mengen $(s, w) \in S \times W$
- in Codebereichen sind alle Kommentare und Variablen auf Englisch benannt, da auch die Schlüsselwörter der Sprache aus dem Englischen kommen
- der vollständige Code kann auf Github eingesehen werden unter:
  `https://github.com/gmpuntigam/exam_planner`
  Die Daten der Hochschule sind nicht öffentlich zugänglich. Der Autor ist bemüht, zeitnah ein Skript zur Erzeugung von Pseudodaten zur Verfügung zu stellen. Der zu bewertende Commit ist Nummer 7 vom 02.11.2020.

3 Optimierung des Prüfungsplans

3.1 Planungsprobleme Allgemein

Planungsprobleme aller Art beschäftigen sich damit, eine Anzahl von Objekten die untereinander interagieren, in eine solche Konfiguration zu bringen, dass die Gesamtheit der Interaktion möglichst günstig ist. Das kann zum Beispiel die Zuweisung von Aufgaben an Personen sein, die Platzierung von Gebäuden an verschiedenen Orten oder die Verteilung von Terminen in einem Zeitfenster. Alle Szenarien werden dabei nach gewissen Kriterien beurteilt und es gilt, das beste zu finden. Dabei gibt es normalerweise zu viele Konfigurationsmöglichkeiten, als dass sie alle einzeln innerhalb von einer realistischen Zeit bewertet werden könnten.

3.2 Problemstellung

Die Aufgabe ist es, möglichst vielen Studenten einen Prüfungszeitraum mit gleichmäßig verteilten Prüfungen zu gewährleisten. Die gegebenen Rahmenbedingungen, die das Problem beschreiben sind folgende:

- 3 Wochen Prüfungszeitraum
- 5 Prüfungszeiten pro Tag
- 80 verschiedene Prüfungsgruppen
- 1500 Studierende mit individuellen Prüfungswahlen
- pro Slot 250 - 620 verfügbare Plätze zum Prüfungschreiben
- zeitliche Einschränkungen zu einzelnen Prüfungen (feste Uhrzeiten oder Tage)

Das Programm muss also alle diese Rahmenbedingungen beachten. Eine weitere Anforderung ist die Laufzeit. Idealerweise sollte innerhalb von einigen Minuten bis wenigen Stunden ein brauchbarer bzw. sogar optimaler Prüfungsplan feststehen.

Es muss zunächst definiert werden, wie das Problem modelliert werden soll und wie die Informationen als Daten, Optimierungsziele und Restriktionen im Rahmen des Modells konkret erfasst werden können. Am Ende soll jeweils ein Stundenplan der Prüfungen ausgegeben werden, damit die Ergebnisse vergleichbar sind.

3.3 Inputs und Ergebnisse

Ausgangspunkt der Optimierung ist eine Liste aller individuellen Prüfungsanmeldungen, die Anmeldecodes, Informationen zu Studiengang und Studiengruppe enthält, sowie die Studenten ID, eine anonymisierte Form der Matrikelnummer.

ANCODE	STG	STGRU	Stud_ID
031	FAB	3B	1116
021	FAB	1C	582
026	FAB	1C	582
⋮	⋮	⋮	⋮

Des weiteren steht eine Liste zur Verfügung, die Informationen darüber enthält, welcher Anmeldecode zu welcher Prüfung gehört, in welchem Semester diese Prüfung stattfindet, ob diese Prüfung geplant werden muss und zu welcher Prüfungsgruppe sie gehört. Die Prüfungsgruppen fassen Prüfungen, die wegen verschiedenen Studienordnungen und Studiengängen unterschiedlich benannt sind, zusammen zu den Prüfungen, die dann tatsächlich geschrieben werden.

PP_Code	TITEL	Gruppen_ID	planen	Semester
FAB001	Elektrotechnik	G018	1	1
FAB003	Elektrotechnik	G018	1	1
FAB006	Ingenieurmathematik I	G072	1	1
⋮	⋮	⋮	⋮	⋮

Damit die Ergebnisse verglichen werden können, muss festgelegt werden, was der Output der Algorithmen sein soll. Folgendes Format wird für den Stundenplan festgelegt:

Prüfung	Woche	Tag	Slot
G018	1	1	1
G072	2	3	4
G200	0	4	2
⋮	⋮	⋮	⋮

3.4 Analyse der Daten

Die Rohdaten müssen für die Analyse zunächst in ein sinnvolles Format gebracht werden. Es bietet sich an, eine binäre Matrix zu erstellen, in denen die Zeilen für die einzelnen Studierenden stehen und die Spalten für die einzelnen Prüfungen. Die Matrix wird im folgenden `exam_matrix_binary` genannt. Bei S Studierenden und E Prüfungen hat die Matrix die Form $S \times E$. Ein Eintrag a hat genau dann den Wert 1, wenn Student $s \in S$ Prüfung $e \in E$ schreibt. Alle anderen Einträge haben den Wert 0.

$$\texttt{exam\_matrix\_binary}[e, s] = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,e} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,e} \\ \vdots & \vdots & \ddots & \vdots \\ a_{s,1} & a_{s,2} & \cdots & a_{s,e} \end{pmatrix}$$

Das Plotten der Matrix liefert eine erste Übersicht über die Daten. Zur besseren Lesbarkeit wird sie transponiert dargestellt. An Stelle jeder 1 wird ein blauer Balken gezeigt, an Stelle jeder 0 ein weißer:

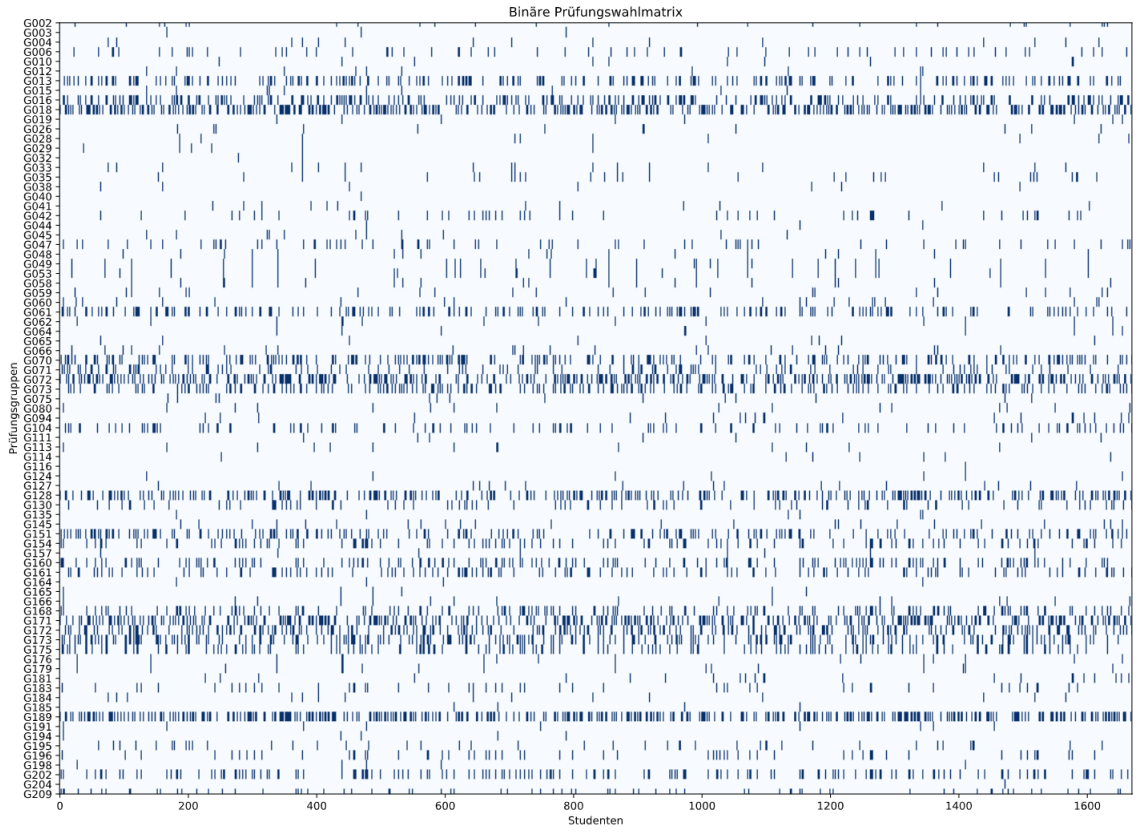

Abbildung 1: Ungeclusterte binäre Prüfungswahlmatrix

Wie erwartet, ist die Matrix (Abbildung 1) relativ dünn besetzt. Es ist erkennbar, dass insgesamt etwa 1650 Studenten Prüfungen angemeldet haben. Es zeigen sich

einige horizontale Streifen (z.B G013 oder G016). Diese weisen darauf hin, dass die Prüfungen von sehr vielen Studenten belegt wurden.

Da Prüfungsanmeldungen unverbindlich sind, sollte im vorliegenden Datenset zuerst die Anzahl der Prüfungsanmeldungen pro Student summiert werden, um zu sehen, ob es Studenten mit auffällig vielen Anmeldungen gibt. Ein Histogramm ermöglicht eine schnelle Übersicht.

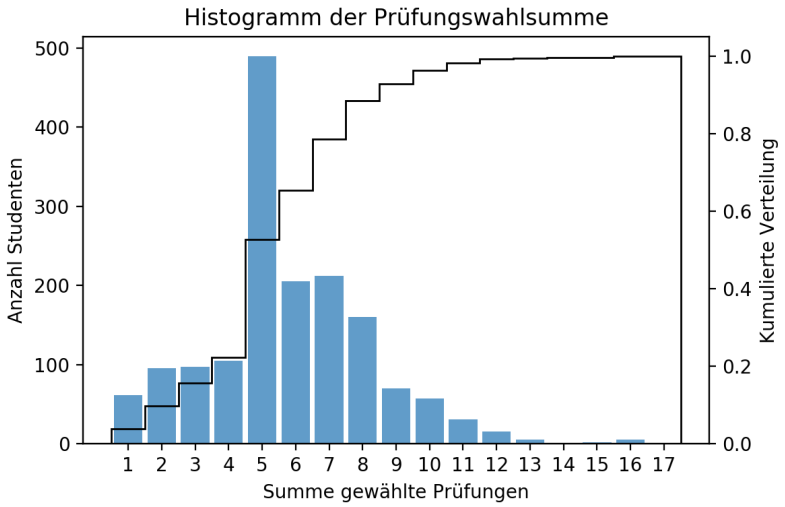

Abbildung 2: Histogramm der Prüfungswahlen

In Abbildung 2 wird ersichtlich, dass die meisten Studenten zwischen fünf und acht Prüfungen anmelden. An den Extremen der Verteilung sind Studenten mit nur einer Anmeldung und Studenten mit bis zu 17 Anmeldungen erkennbar. Es ist anzunehmen, dass die Studenten mit wesentlich mehr Prüfungsanmeldungen als der Durchschnitt niemals alle diese Prüfungen tatsächlich antreten werden. In der Optimierung des Prüfungsplanes werden daher alle Studenten mit mehr als neun Anmeldungen nicht berücksichtigt.

Da die Zeilen der Matrix in Abbildung 1 nicht nach Semester der Studierenden geordnet sind, lässt sich kein Muster bzw. Cluster in der Matrix erkennen. Es liegt aber nahe, dass insbesondere in den unteren Semestern viele Studenten die gleichen Prüfungen wählen werden. Eine Möglichkeit die Matrix zu Clustern bietet das sogenannte spektrale Co-clustering[10]. Es werden Spalten und Zeilen vertauscht, um möglichst große Blöcke von Einsern zu bilden. Dabei muss die Anzahl der gesuchten Blöcke angegeben werden. Da die genaue Anzahl der Cluster aber im Vorhinein nicht bekannt ist, wird der Algorithmus für verschiedene Zahlen ausgeführt und am Ende das beste Ergebnis gehalten. Zur Bewertung der Cluster und um somit die beste Zahl zu finden, gibt es zwei Möglichkeiten. Zum einen kann nach der Anzahl von Clustern gesucht werden, bei der die Cluster am dichtesten gepackt sind, andererseits kann auch nach der Anzahl der Cluster gesucht werden, bei der möglichst

wenige Elemente außerhalb der Cluster liegen.

Das Ziel des Clustern soll sein, herauszufinden, ob es bestimmte Belegungsmuster gibt, die häufig vorkommen. Zum Beispiel: Die Studenten 1 bis 100 wählen alle fast die Prüfungen X und Y. Daher macht hier das Clustern nach Dichte mehr Sinn. Nachdem die Studenten mit mehr als neun gewählten Prüfungen aus dem Datenset entfernt worden, ergibt die Zahl vier besonders dichte Cluster. Nach dem Clustern ergibt sich folgendes Bild:

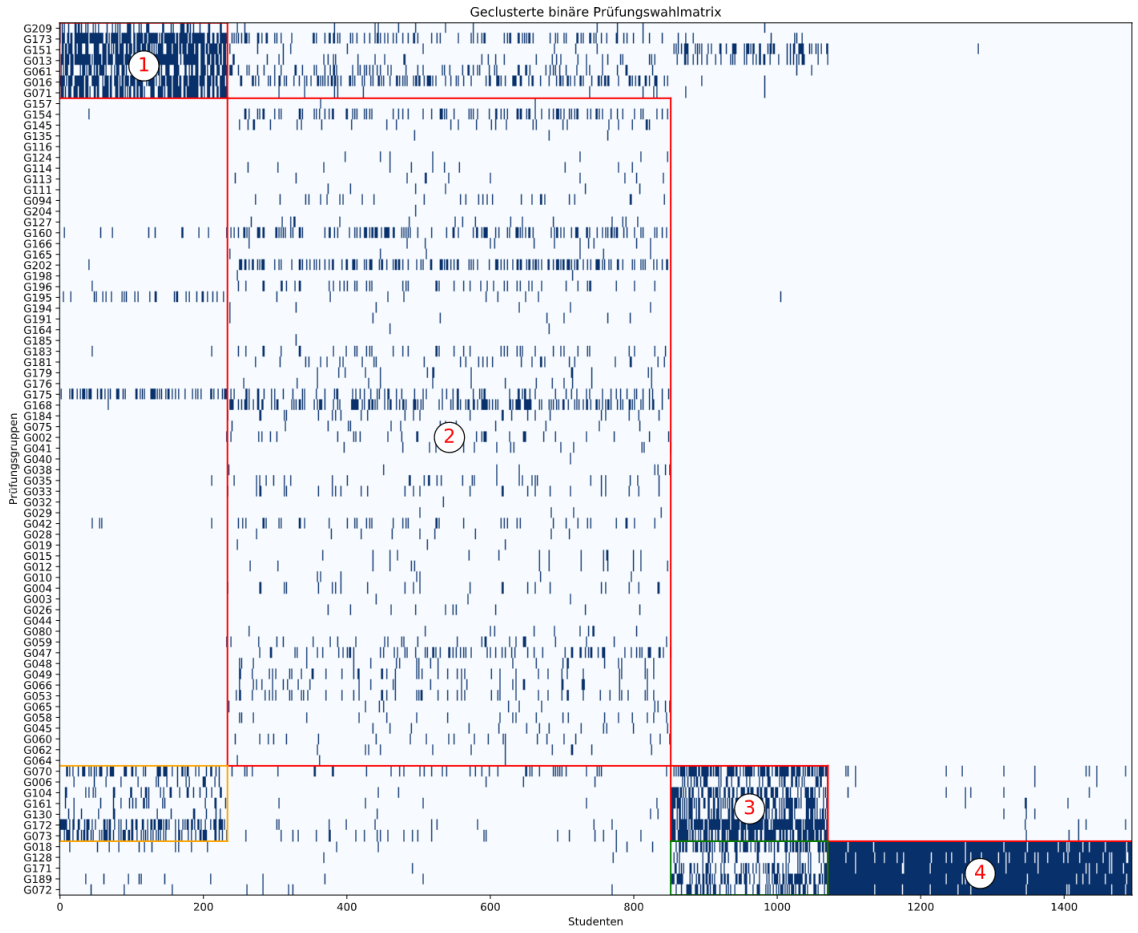

Abbildung 3: Geclusterte binäre Prüfungswahlmatrix

In Abbildung 3 zeigen sich die vier Cluster entlang der Diagonalen der Matrix. Weitere Bereiche hoher Dichte sind als Nebeneffekt abseits der Diagonalen entstanden. Beim Betrachten der G-Nummern und der dazugehörigen Prüfungen, wird ersichtlich, dass der Cluster 4 die Studenten des ersten Semesters darstellt. Der Cluster 3 zeigt die Studenten des zweiten Semesters. Wie aus dem (grün umrandetem) Block unterhalb von Cluster 3 zu erkennen ist, müssen sehr viele Studenten noch Prüfungen aus dem ersten Semester nachschreiben. Der Cluster 1 zeigt die Studenten des dritten Semesters. Auch hier gibt es wieder einige Studenten, die Prüfungen aus den unteren Semestern, insbesondere dem zweiten Semester nach-

schreiben müssen (orange umrandet). Cluster Bereich 2 in der Mitte des Bildes stellt die Studenten der höheren Semester dar. Hier fallen noch einige horizontale Linien auf, die auf Prüfungen mit hohen Studentenzahlen hindeuten. Dieser Ausschnitt der Matrix könnte separat erneut geclustert werden, um genauere Informationen über Belegungsmuster innerhalb der höheren Semester herauszufinden. Dem Autor ist aufgefallen, dass bereits geringe Veränderungen der vorliegenden Daten (zum Beispiel einige leere Spalten in der transponierten Matrix durch Fehler in der Datenvorbereitung) die ideale Anzahl der Cluster beeinflussen, sich jedoch immer die Cluster der ersten drei Semester bilden. Für die Lösung des Optimierungsproblems zeigen sich somit erste Tendenzen. Prüfungen aus verschiedenen Clustern können einigermaßen unabhängig voneinander platziert werden, Prüfungen aus dem gleichen Cluster jedoch nicht. Das Problem kann also vermutlich anhand der Cluster entkoppelt und in mehrere Optimierungsschritte unterteilt werden, ohne dass das Gesamtergebnis stark beeinflusst wird.

Wenn nun die Zeilen der Matrix summiert werden, ergibt sich eine Übersicht, wie groß die unterschiedlichen Prüfungen sind. Das gibt wichtige Informationen darüber, welche Prüfungen eine hohe Teilnehmerzahl haben und bei der Planung eine hohe Priorität haben, da sie bei ungeschickter Platzierung mehr Konflikte verursachen können.

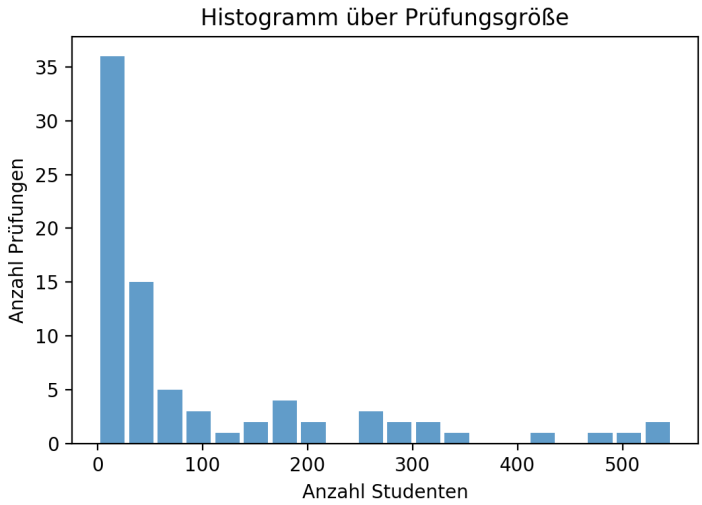

Abbildung 4: Histogramm der Prüfungsgrößen

In Abbildung 4 wird ersichtlich, dass der Großteil der Prüfungen nur von einer geringen Zahl von Studenten geschrieben wird. Das erste Drittel der Prüfungen liegt etwa im Bereich von bis zu 25 Studenten, das zweite bis etwa 100 und das dritte bildet den Rest.

3.5 Optimierungsziel: Zielfunktion und Nebenbedingungen

Um die Ergebnisse der Algorithmen bewerten zu können muss festgelegt werden, was das Ziel der Optimierung ist. Dazu ist eine Funktion nötig, die die Ergebnisse bewerten kann. Weiterhin gibt es Nebenbedingungen bzw. Restriktionen, die die Lösung auf jeden Fall erfüllen muss. Die Algorithmen sollen also gewährleisten, dass die Prüfungen für alle Studierenden möglichst gleichmäßig verteilt sind und diese Restriktionen bestmöglich erfüllen. Die Konkrete Zielfunktion und die Restriktionen lassen sich Anhand eines Prüfungsplan eines einzelnen Studenten aufstellen. So ein Prüfungsplan kann vier unerwünschte Eigenschaften aufweisen:

- mehrere Prüfungen am gleichen Tag, zur selben Uhrzeit
- mehrere Prüfungen an einem Tag
- eine hohe Anzahl von Prüfungen in einer Woche
- viele Prüfungen an aufeinanderfolgenden Tagen

Jedem Student sollte es möglich sein, alle seine gewählten Prüfungen zu schreiben. Daher muss es unbedingt vermieden werden, dass er zwei Prüfungen zur exakt gleichen Zeit schreiben soll. Daher ist zur Vermeidung dieses Ereignisses entweder eine harte Restriktion oder ein Optimierungsteilziel mit einer extrem hohen Strafe nötig.

Die anderen Ereignisse werden sich nicht alle vermeiden lassen. Deswegen können diese nicht alle als Restriktionen formuliert werden. Deswegen muss die Bewertungsfunktion also für jedes dieser Ereignisse in einem Prüfungsplan Strafpunkte verteilen. Weil nicht alle diese Szenarien gleichermaßen stören, muss es eine Gewichtung zwischen diesen geben. Diese Gewichtung kann nicht komplett objektiv erstellt werden und muss daher von dem Anwender ggf. angepasst werden, da sie einen großen Einfluss auf das Gesamtergebnis haben kann.

Mehrere Prüfungen an einem Tag sind sehr störend. Daher gibt es eine hohe Strafe für zwei Prüfungen an einem Tag. Mehr als zwei Prüfungen an einem Tag werden als Restriktion nicht erlaubt. Viele Prüfungen in einer Woche und mehrere Prüfungen an aufeinanderfolgenden Tagen stören am wenigsten und können kaum vermieden werden, deswegen gibt es nur eine geringe Strafe.

Diese Restriktionen und Optimierungsziele müssen im nächsten Schritt nun in einem Modell konkret formuliert werden, um das Problem zu lösen.

3.6 Modellkonzepte

Jedes Modell, das dieses Problem lösen kann, hat unterschiedliche Vorteile und Nachteile. Manche Modelle laufen sehr schnell, da sie statistische Methoden nutzen, können aber aufgrund ihrer Natur keine Aussagen machen, wie nahe die gefundene Lösung an der Ideallösung ist. Andere Modelle hingegen brauchen mehr Zeit, können aber prinzipiell eine ideale Lösung für das Problem finden. Im folgenden werden zwei unterschiedliche Modelle vorgestellt, die genutzt werden können, um dieses Problem zu lösen.

3.6.1 Linearer Solver

Ausgehend davon, dass das Problem als lineares, ganzzahliges Optimierungsproblem dargestellt werden kann (die Linearität hängt von der exakten Zielfunktion ab, die unter 4.1.1.4 aufgestellt wird), kann ein Linearer Solver zur Lösung des Problems genutzt werden.

Der Lineare Solver errechnet zuerst eine ideale Lösung, in der die Lösungswerte keine ganzen Zahlen sein müssen und eine Lösung, die nicht zwingend gut ist, aber die Bedingung erfüllt, dass die Lösungswerte ganze Zahlen sein müssen. Damit gibt es eine untere und eine obere Grenze für das Problem. Ausgehend von der idealen Lösung werden nun nach und nach die Fließkommazahlen in ganze Zahlen überführt, indem ein Suchbaum angelegt wird, an dessen Verzweigungen jeweils eine Fließkommazahlen auf ganzzahlige Werte beschränkt wird. Die Äste dieses Suchbaumes werden nach und nach abgearbeitet und die Ober- und Untergrenze des Problems nähern sich. Sobald diese Grenzen sich treffen, ist die Optimalität der Lösung bewiesen [1].

3.6.1.1 Herausforderungen

Die Annäherung der Grenzen bei Problemen mit sehr vielen Variablen kann äußerst lange dauern, da der zu durchsuchende Lösungsbaum sehr groß wird und es lange dauert, die Äste abzuarbeiten. Die Effektivität des Branch und Bound Algorithmus hängt davon ab, wie schnell er die Äste aus dem Suchbaum verwerfen kann [2]. Manchmal kann durch Untersuchen der bekannten, gültigen Lösungen der Lösungsraum einschränkt werden. Viele Probleme sind allerdings so komplex, dass es nicht möglich ist, in einer realistischen Zeit eine optimale Lösung zu finden.

Bei dem konkreten Problem stellt die mathematischen Formulierung der Bewertung des zeitlichen Abstands zwischen den Prüfungen eine weitere Herausforderung dar. Da für jede Prüfung an Tag t nur eine Aussage über den Abstand zur letzten Prüfung gemacht werden kann, wenn auch bekannt ist, ob eine Prüfung an Tag $t-1$ liegt, ist das Problem quadratisch. Viele quadratische Probleme lassen sich jedoch mit Hilfsvariablen in einem bestimmten Bereich linearisieren.

3.6.2 Simulated Annealing

Simulated Annealing ist dem Abkühlen und Kristallisieren einer Metallschmelze nachempfunden. In einer Schmelze sind die Atome zunächst ungeordnet. Beim Abkühlen bilden sich ausgehend von Keimen Kristalle heraus. Diese können jedoch bis zum völligen Erstarren immer wieder ihre Form und Struktur verändern.

Ähnlich dazu dient beim Simulated Annealing eine durch Zufall generierte, nicht unbedingt gute Lösung des Problems als Ausgangspunkt. In jedem Schritt wird nun eine benachbarte Lösung, also eine Lösung in der nur eine Variable geändert ist, generiert und mit der bekannten Lösung verglichen. Hierzu gibt es eine sogenannte Energiefunktion. Ist die neue Lösung besser, wird die alte Lösung verworfen. Ist die neue Lösung schlechter gibt es trotzdem eine Wahrscheinlichkeit, dass diese Lösung akzeptiert wird. Diese Wahrscheinlichkeit sinkt mit jeder Iteration [5].

Auf diese Art und Weise entstehen schnell gute Lösungen, eine ideale Lösung wird jedoch höchstens durch Zufall erreicht. Von diesem Algorithmus wurden schon einige Varianten entwickelt, die verschiedene Schwachstellen von ihm ausbessern sollen oder auf spezifische Probleme zugeschnitten sind. Er wird oftmals angewendet, wenn es wichtiger ist schnell eine gute Lösung zu finden, die allen Anforderungen entspricht, als eine Lösung die absolut ideal ist. Ein bekanntes Problem, bei dem Simulated Annealing oftmals angewandt wird, ist das “Travelling Salesman” Problem. Dabei geht es darum, die kürzeste Route zwischen mehrere Städten zu ermitteln.

3.6.2.1 Herausforderungen

Beim Simulated Annealing ist es besonders wichtig, dass der Algorithmus für jeden Zeitschritt so schnell wie möglich ist, da eine große Anzahl von Iterationen absolviert werden muss. Daher ist es wichtig, den Programmcode so effizient wie möglich zu gestalten. Ein weiteres Problem ist, dass der Algorithmus oftmals unterschiedlich lange braucht, um eine gute Lösung zu finden. Fortgeschrittene Varianten besitzen daher eine adaptive “Temperaturregelung”, die sich dem Verlauf der Energiefunktion anpasst. Die Version, die in dieser Arbeit genutzt wird, hat dies nicht.

4 Vergleich der Modelle

4.1 Linearer Solver

In diesem Abschnitt gilt für die mathematische Notation:

- W beschreibt die Menge der Wochen
- T die Menge der Tage pro Woche
- Z die Zeitslots am Tag
- E ist die Menge der Prüfungen
- S die Menge der Studenten

Übersicht der verwendeten Datenfelder:

Datenfeld	**Beschreibung**
`exam_placing`	One-Hot Platzierung der Prüfungen
`student_schedule`	One-Hot Prüfungsplan der Studenten
`normal_weeks`	Hilfsvariable Wochen
`double_weeks`	Hilfsvariable Wochen
`normal_days`	Hilfsvariable Tage
`double_days`	Hilfsvariable Tage
`consecutive_days_a`	Hilfsvariable konsekutive Tage
`consecutive_days_b`	Hilfsvariable konsekutive Tage
`exam_matrix_binary`	Student-Prüfungsmatrix
`MaxCapacity`	Kapazitätsvektor
`exam_choice`	Gewählte Prüfungen der Studenten
`planned_exams`	Vorgeplante Prüfungen
`planned_exams_timeonly`	Vorgeplante Prüfungszeiten

Vier Datenfelder liefern dabei alle nötigen Informationen für das Modell.
`exam_matrix_binary` ist die binäre Prüfungsmatrix aus 3.4.
`MaxCapacity` gibt die maximale Kapazität zu jedem Zeitslot an. In diesem Fall:

$$\texttt{MaxCapacity} = [620, 620, 620, 250, 620]$$

`exam_choice` wird aus der binären Prüfungsmatrix erstellt und ist eine Liste mit einem Eintrag für jeden Studenten, die in jedem dieser Einträge wieder eine Liste enthält mit den gewählten Prüfungen des jeweiligen Studenten. Dabei werden die Prüfungsgruppen in Nummern übersetzt und in dem Dictionary `group_to_row` gespeichert, also G002 $\rightarrow$ 0, G003 $\rightarrow$ 1, ...

Der Anfang der Liste könnte also zum Beispiel so aussehen:

Stud. ID	**Liste der Prüfungen**
0	[2, 4, 5, 7]
1	[5, 23, 46, 55, 64]
⋮	⋮

`planned_exams` ist eine Liste von Prüfungen, deren Zeitpunkt schon vor der Planung feststeht und nicht verändert werden darf. Dabei sind auch die Prüfungsgruppen durch entsprechende Nummern ausgetauscht. Die Liste könnte zum Beispiel so aussehen:

ID	**[Prüfungsnr, Woche, Tag, Slot]**
0	[0, 1 1 1]
1	[5, 2 4 1]
2	[43, 0 1 3]

4.1.1 Das Modell

4.1.1.1 Entscheidungsvariablen

Zur Erstellung eines Prüfungsplanes muss jeder Prüfung genau ein Zeitpunkt zugewiesen werden. Dazu dient das Datenfeld `exam_placing`[w,t,z,e], welches one-hot kodiert ist. Wenn in Woche w, an einem Tag t, zur Zeit z die Prüfung e stattfindet, ist ein Eintrag eins, sonst sind alle Einträge Nullen.

$$\texttt{exam\_placing}[w, t, z, e] \in \{0, 1\}$$

$$\forall w \in W;\ \forall t \in T;\ \forall z \in Z;\ \forall e \in E$$

Dieses Datenfeld ist das eigentliche Ergebnis der Optimierung. Ein zweites Datenfeld dient dazu, die Berechnung des Optimierungsziels zu vereinfachen. Dieses ist auch one-hot kodiert, und besitzt eine zusätzliche Dimension, die Informationen zu den einzelnen Studenten beinhaltet. Wenn Student s in der Woche w an einem Tag t, zur Zeit z die Prüfung e schreibt, ist ein Eintrag eins, sonst sind alle Einträge nullen.

$$\texttt{student\_schedule}[s, w, t, z, e] \in \{0, 1\}$$

$$\forall s \in S;\ \forall w \in W;\ \forall t \in T;\ \forall z \in Z;\ \forall e \in E$$

4.1.1.2 Randbedingungen

Zunächst müssen diese beiden Felder miteinander in Relation gebracht werden. Das geschieht mit Hilfe des bekannten Datenfelds `exam_choice`, das die Informationen über die gewählten Prüfungen der Studenten enthält:

$$\begin{aligned} \texttt{student\_schedule}[s,w,t,z,e] &= \texttt{exam\_placing}[w,t,z,e] \\ \forall s \in S;\ \forall w \in W;\ \forall t \in T; &\forall z \in Z;\ \forall e \in \texttt{exam\_choice}[s] \end{aligned} \tag{RB-1}$$

Jeder Student darf pro Slot maximal eine Prüfung schreiben:

$$\begin{aligned} \sum_{e \in E} \texttt{student\_schedule}[s,w,t,z,e] &\leq 1 \\ \forall s \in S;\ \forall w \in W;\ \forall t \in T;\ &\forall z \in Z \end{aligned} \tag{RB-2}$$

Zu jedem Zeitpunkt muss die Summe der Studenten, die eine Prüfung schreiben, geringer als die Verfügbare Kapazität sein:

$$\begin{aligned} \sum_{s \in S} \sum_{e \in E} \texttt{student\_schedule}[s,w,t,z,e] &\leq \texttt{MaxCapacity}[z] \\ \forall w \in W;\ \forall t \in T;\ &\forall z \in Z \end{aligned} \tag{RB-3}$$

Jeder Student schreibt genau so viele Prüfungen, wie er gewählt hat. Diese Randbedingung ist wichtig, weil sonst dem Linearen Solver keine Informationen über alle nicht gewählten Prüfungen vorliegen. Nur so ist klar, welche Prüfungen jeder Student exakt schreibt:

$$\sum_{e \in E} \texttt{exam\_matrix\_binary}[e,s] = \sum_{w \in W} \sum_{t \in T} \sum_{z \in Z} \sum_{e \in E} \texttt{student\_schedule}[s,w,t,z,e]$$

$$\forall s \in S \tag{RB-4}$$

Jede Prüfung bekommt genau einen Slot zugewiesen:

$$\sum_{w \in W} \sum_{t \in T} \sum_{z \in Z} \texttt{exam\_placing}[w,t,z,e] = 1 \quad \forall e \in E \tag{RB-5}$$

Prüfungen, deren Zeitpunkt schon vor der Erstellung des kompletten Planes feststeht, müssen als Randbedingung formuliert werden.

$$\texttt{exam\_placing}[w,t,z,e] = 1 \quad \forall (w,t,z,e) \in \texttt{planned\_exams} \tag{RB-6}$$

Die Prüfungen G013 und G160 müssen in Slot 1 geschrieben werden, also etwa um 10 Uhr, ein fester Tag ist jedoch nicht vorgesehen. In der Liste `planned_exams_timeonly` können Prüfungen angegeben werden, und mit einem bestimm-

ten Slot versehen werden. Dementsprechend müssen alle Slots im Datenfeld außer dem gewünschten Slot gesperrt werden. Als Randbedingung formuliert:

$$\texttt{exam\_placing}[w,t,z,e] = 0 \quad \forall w \in W;\ \forall t \in T;\ \forall z \in Z \setminus \{slot\};\ \forall (e, slot) \in \texttt{planned\_exams\_timeonly} \tag{RB-7}$$

4.1.1.3 Doppelte Prüfungen an einem Tag vermeiden

Um zu erfassen, wann ein Student zwei Prüfungen an einem Tag schreibt, können zwei Datenfelder genutzt werden.

$$\texttt{normal\_days}[s,w,t] \in \{0,1\} \quad \forall s \in S;\ \forall w \in W;\ \forall t \in T$$

$$\texttt{double\_days}[s,w,t] \in \{0,1\} \quad \forall s \in S;\ \forall w \in W;\ \forall t \in T$$

Das Datenfeld `normal_days` wird immer zuerst gefüllt werden, und beinhaltet alle straffreien Prüfungen. Das zweite zählt lediglich die zweiten Prüfungen.

Diese beiden Felder müssen mit dem Datenfeld des Prüfungsplans der Studenten verknüpft werden.

$$\texttt{normal\_days}[s,w,t] + \texttt{double\_days}[s,w,t] = \sum_{z \in Z} \sum_{e \in E} \texttt{student\_schedule}[s,w,t,z,e] \quad \forall s \in S;\ \forall w \in W;\ \forall t \in T \tag{RB-8}$$

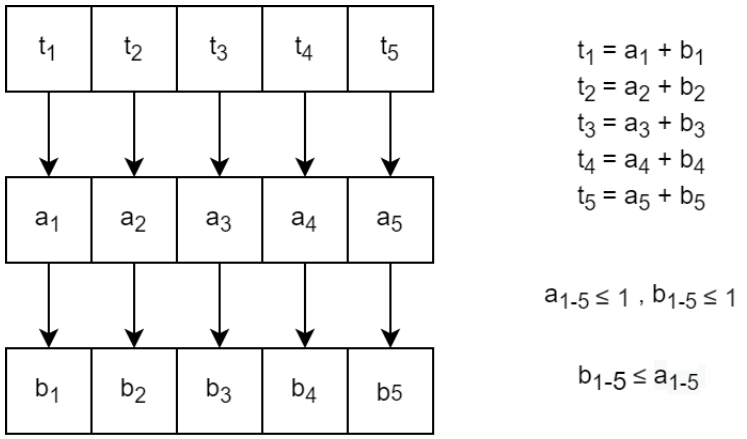

Abbildung 5: Diagramm zur Erfassung von doppelten Prüfungen an gleichen Tagen

Abbildung 5 zeigt, wie die Datenfelder gefüllt werden. Die Variablen t_1 bis t_5 enthalten die Anzahl der Prüfungen, die ein Student an dem jeweiligen Tag schreibt. Die Variablen a_1 bis a_5 gehören hier zum Datenfeld `normal_days`, die Variablen b_1 bis b_5 zum Datenfeld `double_days`. Wenn ein Student also an Tag 1 eine Prüfung schreibt, ist der Eintrag in a_1 1 und der Rest des Datenfelds null. Es gibt keine Straf-

punkte. Kommt nun eine zweite Prüfung an Tag 1 dazu, wird auch das Datenfeld b_1 1 und es gibt Strafen.

Für die Optimierung der aufeinanderfolgenden Tagen müssen einige Tricks genutzt werden.

4.1.1.4 Linearisierung eines quadratischen Problems

Wenn der Abstand zwischen einzelnen Prüfungen für die Studenten betrachtet wird, stellt sich ein quadratisches Problem. Folgende Formulierung macht das klar:

$$\min \sum_{s \in S} \sum_{w \in W} \sum_{t \in T \setminus \{1\}} (\sum_{z \in Z} \sum_{e \in E} \texttt{student\_schedule}[s, w, t, z, e] \cdot$$

$$\sum_{z \in Z} \sum_{e \in E} \texttt{student\_schedule}[s, w, t-1, z, e])$$

In dieser Formel ist der Betrag des innersten Produkts immer dann gleich 1, wenn ein Student an zwei aufeinanderfolgenden Tagen mindestens eine Prüfung schreibt. Das Datenfeld, welches hier tatsächlich verändert wird, ist `exam_placing`. Das Datenfeld `student_schedule` beschreibt dabei die Auswirkungen der Platzierung der Prüfungen für die individuellen Prüfungspläne der Studenten. So entsteht ein quadratisches Minimierungsproblem. Quadratische Probleme stellen eine große Herausforderung für Solver dar.

Eines der ersten quadratischen Optimierungsprobleme wurde bereits 1957 von Koopmans und Beckmann vorgestellt [8]. In diesem Problem müssen Einrichtungen an verschiedenen Orten platziert werden. Für alle Einrichtungen entstehen Kosten für die erstmalige Einrichtung sowie Kosten, die sich durch den Abstand zu den jeweils anderen Einrichtungen ergeben. Seitdem wurde viele Jahre lang an dem Problem geforscht. Normalerweise werden die Probleme durch neue Variablen und Randbedingungen linearisiert und das entstehende Lineare Problem gelöst. Zum Oktober 2006 [9] galten Probleme mit mehr als 20 zu platzierenden Gebäuden als eine große Herausforderung, was die Suche nach einer idealen Lösung betrifft. Für das Prüfungsplanproblem müssen also auch Hilfsvariablen genutzt werden, um das Problem zu linearisieren.

Dazu werden zwei weitere Datenfelder benötigt. Diese Datenfelder besitzen die Dimensionen Student, Woche und Tag (allerdings nur mit vier statt fünf möglichen Werten). Beide Datenfelder sind in jedem Element auf eins begrenzt. Die Einträge in dem ersten Datenfeld bleiben dabei straffrei, diejenigen in dem zweiten nicht. Diese Datenfelder werden folgend als `consecutive_days_a` und `consecutive_days_b` bezeichnet. Abbildung 6 veranschaulicht, wie die Prüfungen auf die beiden Datenfeldvektoren verteilt werden.

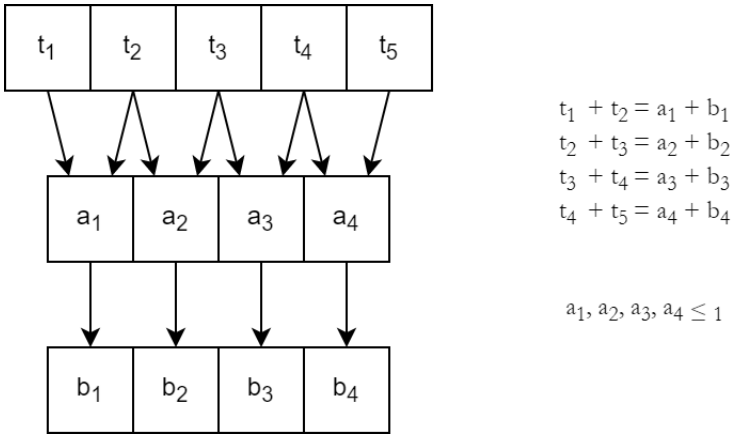

Abbildung 6: Diagramm zur Bewertung aufeinanderfolgender Tage

Die Variablen t_1 bis t_5 entsprechen dem Datenfeld `normal_days`, welches gleich 1 ist, wenn ein Student an dem jeweiligen Tag mindestens eine Prüfung schreibt. Die Variablen a_1 bis a_4 des Datenfelds `consecutive_days_a` enthalten jeweils die Summe der Prüfungen von zwei aufeinanderfolgenden Tagen und sind auf 1 begrenzt. Daher wird bei zwei aufeinanderfolgenden Prüfungen eine davon in dem Datenfeld `consecutive_days_b` gespeichert.

Ein Beispiel macht das ganze verständlicher. Hat ein Student am Montag und Mittwoch eine Prüfung, sind im Feld `consecutive_days_a` die ersten drei Einträge gleich eins, da in den Gruppen der aufeinanderfolgenden Tagen Montag und Dienstag, Dienstag und Mittwoch, sowie Mittwoch und Donnerstag jeweils eine Prüfung liegt. Jeder Eintrag in `consecutive_days_b` ist somit noch Null. Hat dieser Student nun zusätzlich am Dienstag eine Prüfung, gibt es in den Gruppen Montag und Dienstag, sowie Dienstag und Mittwoch jeweils zwei Prüfungen. Diese weitere Prüfung kann nun nicht mehr von dem Feld `consecutive_days_a` erfasst werden und somit sind die beiden Einträge für Montag und Dienstag, sowie Dienstag und Mittwoch im Feld `consecutive_days_b` auch gleich 1.

4.1.1.5 Prüfungen an aufeinanderfolgenden Tagen vermeiden

Mit diesem Wissen können die nötigen Hilfsdatenfelder definiert werden:

$$\texttt{consecutive\_days\_a}[s, w, t] \in \{0, 1\} \quad \forall s \in S;\ \forall w \in W;\ \forall t \in T$$

$$\texttt{consecutive\_days\_b}[s, w, t] \in \{0, 1\} \quad \forall s \in S;\ \forall w \in W;\ \forall t \in T$$

Die Summe der Prüfungen der Hilfsdatenfelder muss in jedem Eintrag der Summe der Prüfungen von zwei aufeinanderfolgenden Tagen entsprechen:

$$\begin{aligned} &\texttt{consecutive\_days\_a}[s, w, t] + \texttt{consecutive\_days\_b}[s, w, t] \\ &\quad = \texttt{normal\_days\_a}[s, w, t] + \texttt{normal\_days\_a}[s, w, t+1] \quad \text{(RB-9)} \\ &\qquad \forall s \in S;\ \forall w \in W;\ \forall t \in T \setminus \{5\} \end{aligned}$$

4.1.1.6 Optimierungsziel

Abbildung 7 zeigt nochmal übersichtlich, wie die verschiedenen Datenfelder zusammenhängen.

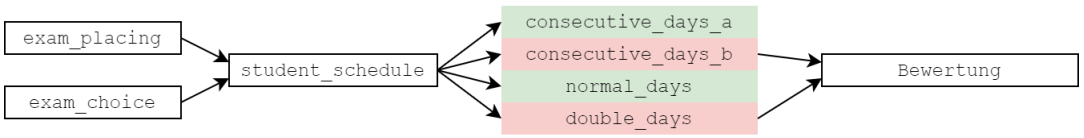

Abbildung 7: Zusammenhangs der Datenfelder zur Bewertung

Ausgehend von `exam_placing` und `exam_choice` wird `student_schedule` befüllt. Dementsprechend füllen sich die Hilfsdatenfelder. Die Felder `consecutive_days_a` und `normal_days` fügen keine Punkte zum Bewertungsscore hinzu. Die Einträge in Feldern `consecutive_days_b` und `double_days` hingegen generieren Strafpunkte, da diese Einträge durch zwei Prüfungen an aufeinanderfolgenden Tagen oder zwei Prüfungen an einem Tag entstehen. Diese Felder enthalten Informationen darüber, wie der Plan für jeden individuellen Studenten bewertet wird. Das Optimierungsziel ist es also, die Inhalte dieser beiden Felder für alle Studenten zu minimieren. Die Inhalte der Felder müssen jetzt noch addiert und zueinander ins Verhältnis gesetzt werden. Die Vermeidung von Prüfungen am selben Tag wurde hier als fünf mal wichtiger angesehen und mit einem entsprechenden Faktor versehen. Im Programm ist dies als Parameter mit Standartwert von fünf realisiert.

$$\min \sum_{s \in S} \sum_{w \in W} \Big(\sum_{t \in T \setminus \{1\}} \texttt{consecutive\_days\_b}[s, w, t] + 5 \cdot \sum_{t \in T} \texttt{double\_days}[s, w, t] \Big)$$

4.1.2 Teilprobleme und Heuristiken

Wenn der Lineare Solver in einem Schritt einen Prüfungsplan für alle Studierenden entwerfen soll, steht er vor einer großen Aufgabe. Selbst wenn theoretisch pro Slot nur eine Prüfung zugelassen wird und nur 75 Prüfungen geplant werden müssten, gibt es bei drei Wochen mit jeweils fünf Prüfungstagen und fünf Slots pro Tag schon $(3 \times 5 \times 5)! = 75! > 10^{109}$ Möglichkeiten, die Prüfungen zu platzieren. Die Forderung, dass ein Student maximal zwei Prüfungen an einem Tag schreiben darf, schließt zwar ein Teil dieser Möglichkeiten aus, allerdings gibt es in jedem Fall noch eine Unmenge von Möglichkeiten. Der Solver versucht zwar nicht dieses Problem mit 'Brute Force' zu lösen, allerdings gibt es bei den vielversprechenden Lösungskandidaten noch eine große Zahl von Variationen, die sich nur minimal in der Bewertung unterscheiden, daher wird der Baum der potentiellen Lösungen des Solvers sehr groß und der Solver braucht eine sehr lange Zeit, um die beste Lösung zu ermitteln. Deswegen ist dieses Problem ohne weitere Vereinfachung fast unlösbar.

Unter der Annahme dass es günstig für die Verteilung ist, wenn jeder Student in jeder Woche etwa gleich viele Prüfungen schreiben muss, ist ein erster möglicher Schritt zur Vereinfachung, die Prüfungen zuerst nur auf die drei Wochen zu verteilen und dabei Tage und Slots zu vernachlässigen. Damit reduziert sich die Anzahl der möglichen Lösungen enorm. Bei x Prüfungen gibt es also nur noch 3^x Möglichkeiten. Selbst hier ist es in einer angemessenen Zeit nicht möglich, alle Prüfungen auf einmal Planen zu lassen, allerdings können ohne Probleme die meisten großen Prüfungen geplant werden.

Wie sich gezeigt hat, lassen sich die Daten gut nach den Semestern der Studenten clustern. In einem ersten Schritt kann also der Solver zunächst nur die Prüfungen für einen Teil der Cluster planen, und dann iterativ die weiteren Cluster planen. Dieses Programm plant zunächst die Prüfungen der ersten beiden Semester, dann die der Semester drei und vier. Anschließend werden die Prüfungen gruppiert nach ihrer Größe geplant, wobei die größten Prüfungen zuerst geplant werden. Wie in Abbildung 4 (Kapitel 3.4) erkennbar ist, lassen sich die Prüfungen nach ihrer Größe in etwa drei gleich große Gruppen unterteilen. Die Planung der restlichen Prüfungen erfolgt also anhand dieser etwa gleich großen Gruppen.

Das Diagramm (Abblidung 8) auf der nächsten Seite zeigt die einzelnen Planungsschritte mit den jeweiligen Inputs und Outputs nochmal in einer grafischen Übersicht.

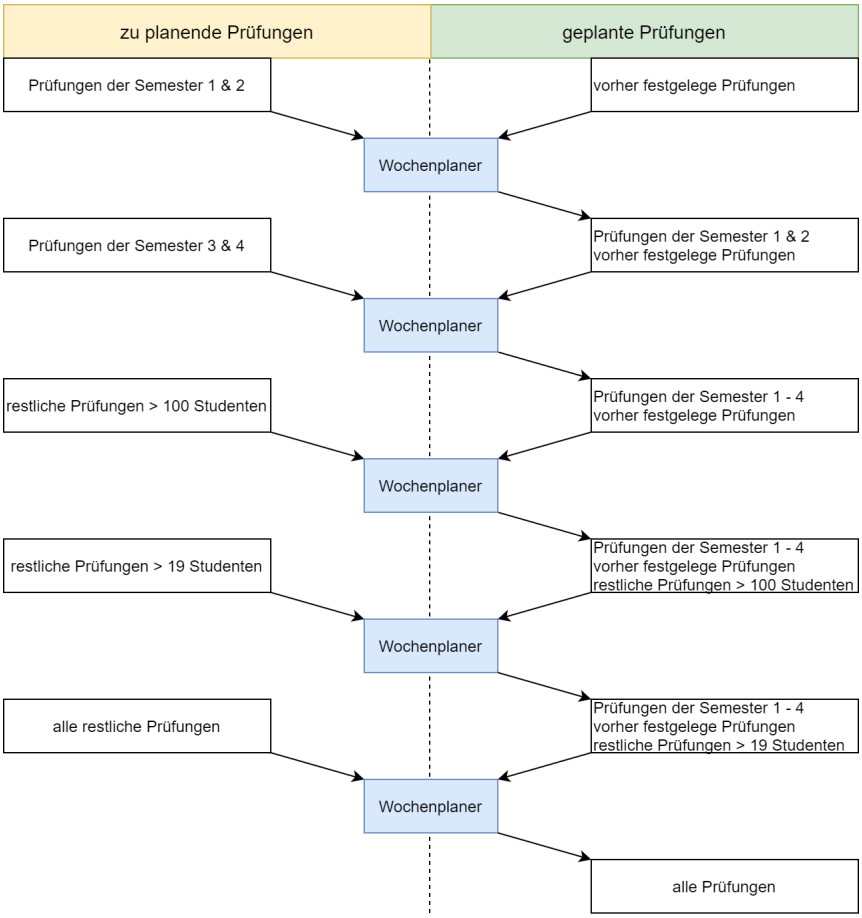

Abbildung 8: Teilschritte der Wochenplanung

Sobald die Prüfungen auf die Wochen verteilt sind, wird in einem zweiten Schritt die beste Verteilung innerhalb der Woche ermittelt. Auch hier gibt es zu viele Möglichkeiten, um alle Prüfungen auf einmal zu planen. In diesem Modell werden daher erst die zehn größten Prüfungen geplant und anschließend der Rest.

4.1.2.1 Wochenplanung

Für die Wochenplanung werden nun die bekannten Datenfeldern um die Dimensionen Tag und Slot reduziert und die Randbedingungen dementsprechend angepasst. Die Datenfelder `exam_placing` und `student_schedule` müssen wieder miteinander verknüpft werden (wie RB-1):

$$\texttt{student\_schedule}[s, w, e] = \texttt{exam\_placing}[w, e]$$

$$\forall e \in \texttt{exam\_choice}[s];\ \forall w \in W;\ \forall s \in S$$

Jeder Student schreibt genau so viele Prüfungen, wie er gewählt hat (wie RB-4):

$$\sum_{e \in E} \texttt{exam\_matrix\_binary}[e, s] = \sum_{e \in E} \texttt{student\_schedule}[s, w, e] \quad \forall s \in S$$

Jede Prüfung bekommt genau einen Termin (wie RB-5):

$$\sum_{w \in W} \texttt{exam\_placing}[w, e] = 1 \quad \forall e \in E$$

Prüfungen, deren Zeitpunkt schon vor der Erstellung des kompletten Planes feststeht, müssen in der richtigen Woche stattfinden (wie RB-6):

$$\texttt{exam\_placing}[w, e] = 1 \quad \forall [w, e] \in \texttt{planned\_exams}$$

Da das Optimierungsziel eine möglich gleichmäßige Verteilung der Prüfungen ist und jeder Student im Durchschnitt etwa sechs Prüfungen schreibt und der Prüfungszeitraum drei Wochen lang ist, können zwei Prüfungen ohne Strafen erlaubt werden. Ab der dritten werden dann Strafen verteilt. Die maximale Anzahl der Prüfungen pro Woche wird auf acht begrenzt. Das Datenfeld `normal_weeks` beinhaltet die straffreien Prüfungen, das Datenfeld `double_weeks` alle weiteren (wie RB-8):

$$\texttt{normal\_weeks}[s, w] \in \{0, 1, 2\} \quad \forall s \in S;\ \forall w \in W;\ \forall t \in T$$

$$\texttt{double\_weeks}[s, w] \in \{0, 1, 2, ..., 6\} \quad \forall s \in S;\ \forall w \in W;\ \forall t \in T$$

$$\sum_{e \in E} \texttt{student\_schedule}[s, w, e] = \texttt{normal\_weeks}[s, w] + \texttt{double\_weeks}[s, w]$$

$$\forall s \in S;\ \forall w \in W$$

Das Optimierungsziel ist es also, die Anzahl der Prüfungen zu reduzieren, die im Datenfeld `double_weeks` erfasst werden.

$$\min \sum_{s \in S} \sum_{w \in W} \texttt{double\_weeks}[s, w]$$

Im Code wird dies wie folgt umgesetzt:

Listing 1: Code für Wochenplanung

```
''' initialise variables '''
#initialise placement of exams
exam_placing = [[S.IntVar(0,1,'')
    for exam in range(N_EXAMS)]
    for week in range(N_WEEKS)]

#initialise timetable for students
student_schedule = [[[S.IntVar(0,1,'')
    for exam in range(N_EXAMS)]
    for week in range(N_WEEKS)]
    for student in range(N_STUDENTS)]

```

```
# initialise helper variables for week optimization
double_booked_weeks = [[S.IntVar(0,6,'')
    for week in range(N_WEEKS)]
    for student in range(N_STUDENTS)]

normal_booked_weeks = [[S.IntVar(0,2,'')
    for week in range(N_WEEKS)]
    for student in range(N_STUDENTS)]

''' Constraints '''
# every exam gets exactly one slot
for exam in range(N_EXAMS):
    S.Add(S.Sum([exam_placing[week][exam]
    for week in range(N_WEEKS)]) == 1)

# every student writes as much exams as chosen
for student in range(N_STUDENTS):
    S.Add(len(exam_choice[student]) ==
    S.Sum([student_schedule[student][week][exam]
        for exam in range(N_EXAMS)
        for week in range(N_WEEKS)]))

# fix preplanned exams
for planned in planned_exams:
    exam = planned[0]
    week = planned[1]
    S.Add(exam_placing[week][exam] == 1)

# fill timetable accoring to exams
for student in range(N_STUDENTS):
    for week in range(N_WEEKS):
        for exam in exam_choice[student]:
            student_schedule[student][week][exam] =
            exam_placing[week][exam]

''' objective/scoring '''
# helper variables must match schedule
for week in range(N_WEEKS):
    for student in range(N_STUDENTS):
        limit=S.Sum([student_schedule[student][week][exam]
            for exam in range(N_EXAMS)])
        S.Add(double_booked_weeks[student][week] +
            normal_booked_weeks[student][week] == limit)

exam_cost = S.Sum([
    double_booked_weeks[student][week]*rowweights[student]
    for week in range(N_WEEKS)
    for student in range(N_STUDENTS)])
S.Minimize(exam_cost)
```

Im Code werden zunächst alle Datenfelder definiert. `S.IntVar(0,1,'')` deklariert eine Variable, mit Obergrenze 1, Untergrenze 0 und ohne Namen. Über die `for` Schleifen werden die Datenfelder erzeugt.

Mittels `S.Add()` werden Beschränkungen per Gleichung und Ungleichung hinzugefügt. `S.Sum()` summiert die Liste innerhalb des Ausdrucks. Die `for` Schleifen innerhalb dieser Summen sind also äquivalent zum Startwert, Endwert und dem Laufindex in der mathematischen Notation. Die Code-Kommentare verdeutlichen, welche Randbedingung jeweils umgesetzt wird.

Am Ende wird das Optimierungsziel definiert. Die Variable `limit` dient dabei nur als Zwischenspeicher des Ergebnisses der ersten Summe. `S.Minimize()` teilt dem Solver mit, dass es sich um ein Minimierungsproblem handelt.

Der Vektor `rowweights` enthält Informationen darüber, wie oft es Kombinationen von Anmeldungen gab, um die Größe der Datenfelder und somit den benötigten Arbeitsspeicher zu reduzieren. `exam_matrix_binary` ist in diesem Fall also eine Matrix mit stets unterschiedlichen Reihen, von der jede Reihe mit `rowweights` gewichtet ist.

4.1.2.2 Tagesplanung

Für die Tagesplanung innerhalb einer Woche gelten die vollständigen Begrenzungen aus dem ursprünglichen Modell. Die Datenfelder können jeweils um die Dimension Woche Reduziert werden, da dieses Teilmodell alle drei Wochen nacheinander optimiert.
Zunächst müssen die Matrizen `exam_placing` und `student_schedule` miteinander verknüpft werden (wie RB-1):

$$\texttt{student\_schedule}[s,t,z,e] = \texttt{exam\_placing}[t,z,e]$$

$$\forall e \in \texttt{exam\_choice}[s],\ \forall z \in Z,\ \forall t \in T,\ \forall s \in S$$

Jeder Student darf maximal eine Prüfung pro Slot schreiben (wie RB-2):

$$\sum_{e \in E} \texttt{student\_schedule}[s,t,z,e] \leq 1 \quad \forall s \in S,\ \forall t \in T,\ \forall z \in Z$$

Die Kapazität der zur Verfügung stehenden Räume darf nicht überschritten werden (wie RB-3):

$$\sum_{s \in S} \sum_{e \in E} \texttt{student\_schedule}[s,t,z,e] \leq \texttt{MaxCapacity}[z] \quad \forall t \in T,\ \forall z \in Z$$

Jeder Student schreibt so viele Prüfungen, wie er gewählt hat (wie RB-4):

$$\sum_{e \in E} \texttt{exam\_matrix\_binary}[e, s] = \sum_{t \in T} \sum_{z \in Z} \sum_{e \in E} \texttt{student\_schedule}[s, t, z, e]$$

$$\forall s \in S$$

Jede Prüfung bekommt genau einen Slot (wie RB-5):

$$\sum_{t \in T} \sum_{z \in Z} \texttt{exam\_placing}[t, z, e] = 1 \quad \forall e \in E$$

Vorher geplante Prüfungen bekommen einen festen Slot (wie RB-6):

$$\texttt{exam\_placing}[t, z, e] = 1 \quad \forall [t, z, e] \in \texttt{planned\_exams}$$

Für Prüfungen mit fester Uhrzeit müssen die übrigen Slots gesperrt werden (wie RB-7):

$$\texttt{exam\_placing}[t, z, e] = 0 \quad \forall t \in T;\ \forall z \in Z \setminus \{slot\};$$

$$\forall [slot, e] \in \texttt{planned\_exams\_timeonly}$$

Für die Optimierung soll jede doppelte Prüfung an einem Tag, sowie Prüfungen an aufeinanderfolgenden Tagen bestraft werden.

Ähnlich wie bei der Wochen-Planung, können für doppelte Prüfungen an einem Tag zwei Datenfelder genutzt werden. Jede zweite Prüfung wird in dem Feld `double_days` gespeichert und jede erste Prüfung im Feld `normal_days`. Beide Felder werden auf 1 begrenzt.

$$\texttt{normal\_days}[s, t] \in \{0, 1\} \quad \forall s \in S;\ \forall t \in T$$

$$\texttt{double\_days}[s, t] \in \{0, 1\} \quad \forall s \in S;\ \forall t \in T$$

Diese beiden Felder müssen wieder mit dem Datenfeld des Prüfungsplans der Studenten verknüpft werden (wie RB-8):

$$\sum_{z \in Z} \sum_{e \in E} \texttt{student\_schedule}[s, t, z, e] = \texttt{normal\_days}[s, t] + \texttt{double\_days}[s, t]$$

$$\forall s \in S;\ \forall t \in T$$

Es gilt die bekannte Bewertungsformel mit den Hilfsdatenfeldern (wie RB-9):

$$\texttt{consecutive\_days\_a}[s,t] \in \{0,1\} \quad \forall s \in S;\, \forall t \in T$$

$$\texttt{consecutive\_days\_b}[s,t] \in \{0,1\} \quad \forall s \in S;\, \forall t \in T$$

$$\texttt{consecutive\_days\_a}[s,t] + \texttt{consecutive\_days\_b}[s,t] =$$

$$\texttt{normal\_days}[s,t] + \texttt{normal\_days}[s,t+1]$$

$$\forall s \in S;\, \forall t \in T \setminus \{5\}$$

Im Code wird dies wie folgt umgesetzt:

Listing 2: Code für Tagesplanung

```
''' initialise variables '''
#initialise placement of exams
exam_placing = [[[S.IntVar(0,1,'')
    for exam in range(N_EXAMS)]
    for slot in range(N_SLOTS)]
    for day in range(N_DAYS_PER_WEEK)]

# initialise timetable for students
student_schedule = [[[[S.IntVar(0,1,'')
    for exam in range(N_EXAMS)]
    for slot in range(N_SLOTS)]
    for day in range(N_DAYS_PER_WEEK)]
    for student in range(N_STUDENTS)]
# initialise helper Variables for day optimization
double_booked_days = [[S.IntVar(0,1,'')
    for day in range(N_DAYS_PER_WEEK)]
    for student in range(N_STUDENTS)]
normal_booked_days = [[S.IntVar(0,1,'')
    for day in range(N_DAYS_PER_WEEK)]
    for student in range(N_STUDENTS)]
consecutive_days_a = [[S.IntVar(0,1,'')
    for day in range(N_DAYS_PER_WEEK-1)]
    for student in range(N_STUDENTS)]
consecutive_days_b = [[S.IntVar(0,1,'')
    for day in range(N_DAYS_PER_WEEK-1)]
    for student in range(N_STUDENTS)]

'''Constraints'''
# every Student a maximum of one exam per slot
for student in range(N_STUDENTS):
    for day in range(N_DAYS_PER_WEEK):
        for slot in range(N_SLOTS):
            S.Add(
            S.Sum([student_schedule[student][day][slot][exam]
                  for exam in range(N_EXAMS)]) <=1)
```

```
# limit number of students to capacity
MaxCapacity = [620,620,620,250,620]
for day in range(N_DAYS_PER_WEEK):
    for slot in range(N_SLOTS):
        S.Add(slot_occupancy[day][slot] <= MaxCapacity[slot])

# every exam gets exactly one slot
for exam in range(N_EXAMS):
    S.Add(S.Sum([exam_placing[day][slot][exam]
    for slot in range(N_SLOTS)
    for day in range(N_DAYS_PER_WEEK)]) == 1)

# every Student a maximum of one exam per slot
for student in range(N_STUDENTS):
    for day in range(N_DAYS_PER_WEEK):
        for slot in range(N_SLOTS):
            S.Add(
            S.Sum([student_schedule[student][day][slot][exam]
            for exam in range(N_EXAMS)]) <=1)

# fix preplanned exams
for planned in planned_exams:
    exam = planned[0]
    day = planned[1]
    slot = planned[2]
    S.Add(exam_placing[day][slot][exam] == 1)

# block slots for exams with preplanned time
for planned in planned_exams_timeonly:
    exam = int(planned[0])
    required_slot = int(planned[1])
    for day in range(N_DAYS_PER_WEEK):
        for slot in range(N_SLOTS):
            if not slot == required_slot:
                S.Add(exam_placing[day][slot][exam] == 0)

#fill timetable accoring to exams
for student in range(N_STUDENTS):
    for day in range(N_DAYS_PER_WEEK):
        for slot in range(N_SLOTS):
            for exam in exam_choice[student]:
                student_schedule[student][day][slot][exam] =
                exam_placing[day][slot][exam]
```

```
'''objective/scoring'''
# helper variables must match actual schedule
for day in range(N_DAYS_PER_WEEK):
    for student in range(N_STUDENTS):
        limit=S.Sum([student_schedule[student][day][slot][exam]
                     for exam in range(N_EXAMS)
                     for slot in range(N_SLOTS)])
        S.Add(double_booked_days[student][day] +
              normal_booked_days[student][day] == limit)

for day in range(N_DAYS_PER_WEEK-1):
    for student in range(N_STUDENTS):
        S.Add(consecutive_days_a[student][day] +
              consecutive_days_b[student][day] ==
              S.Sum([normal_booked_days[student][day]])+
              S.Sum([normal_booked_days[student][day+1]]))

daily_double_count = S.Sum([
    double_booked_days[student][day]*rowweights[student]
    for day in range(N_DAYS_PER_WEEK)
    for student in range(N_STUDENTS)])
consecutive_count = S.Sum([
    consecutive_days_b[student][day]*rowweights[student]
    for day in range(N_DAYS_PER_WEEK-1)
    for student in range(N_STUDENTS)])
exam_cost = daily_double_count*5 + consecutive_count
S.Minimize(exam_cost)
```

Dieser Code ist ebenso mit der Beschreibung der Wochen-Planung zu verstehen, daher entfällt eine weitere Beschreibung an dieser Stelle.

4.1.3 Ergebnisse

Der fertige Stundenplan wurde vom Solver bewertet und liefert somit eine Kennzahl zur ersten Beurteilung der Qualität des Planes. Die Ergebnisse des Modells sollten nun aber auch intensiver untersucht werden, um festzustellen, was dies für die einzelnen Studenten bedeutet.

Einerseits sollte der Solver einen möglichst fairen Stundenplan erstellen, d.h die Varianz der individuellen Bewertung der einzelnen Prüfungspläne der Studenten (indem die Datenfelder `consecutive_days_b` und `double_days` für jeden individuellen Studenten betrachtet werden) sollte möglichst niedrig sein. Andererseits ist auch ein Blick auf den individuell schlechtesten Plan interessant, um auszuschließen, dass die Optimierung auf kosten einzelner Individuen geschieht. Einen ersten Überblick über die Qualität des Planes liefert ein Histogramm, in dem die Bewertungen der individuellen Prüfungspläre dargestellt werden.

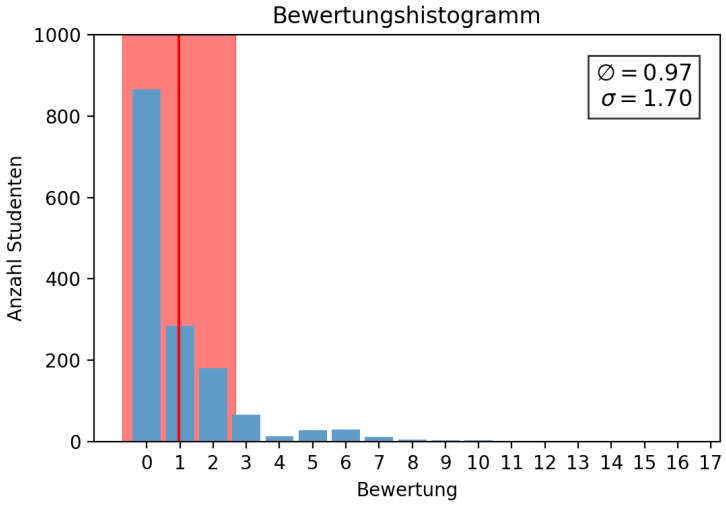

Abbildung 9: Bewertungshistogramm Linearer Solver

In Abbildung 9 ist erkennbar, dass ein Großteil der individuellen Bewertungen bei 0 liegt. Da es sich um ein Minimierungsproblem handelt, bedeutet eine kleine Bewertung ein gutes Ergebnis. Die meisten Scores sind mit 7 oder weniger bewertet, allerdings gibt es vereinzelte Bewertungen mit bis zu 17 Punkten. Durchschnittswert und Standartabweichung sind farbig rot markiert. Wenn der Plan fair erstellt wurde, heißt das auch, dass die Bewertungszahl der individuellen Stundenpläne mit der Anzahl der gewählten Prüfungen steigt. Auf folgendem Plot ist auf der X-Achse die Punktezahl aufgetragen, auf der Y-Achse die Zahl der gewählten Prüfungen und an der Größe der Punkte ist die Häufigkeit der jeweiligen Kombination zu erkennen.

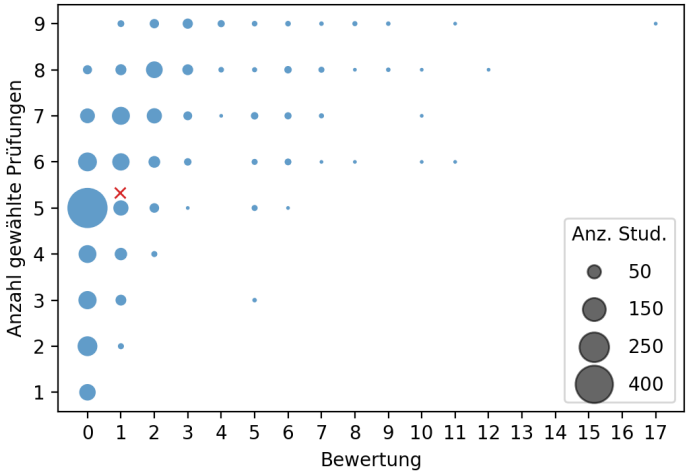

Abbildung 10: Korrelation gewählte Prüfungen und Bewertung Linearer Solver

In Abbildung 10 wird die Verteilung der Scores etwas besser ersichtlich. Der größte Punkt bei (0; 5) zeigt in erster Linie die Erstsemester, deren Prüfungen als

erstes und daher konfliktfrei geplant wurden. Der durchschnittliche Prüfungsplan ist als rotes ‘x’ markiert und liegt etwa bei (0,97; 5,5). Zwei Prüfungen am selben Tag gibt es erst ab drei gewählten Prüfungen. Weiterhin gibt es keinen Studenten, der bei neun gewählten Prüfungen nicht zumindest zweimal zwei Prüfungen an zwei Tagen hintereinander hat.

Zuletzt ist interessant, wie viele Studenten pro Tag insgesamt Prüfungen schreiben. Dies wurde zwar bei der Erstellung dieses Modells nicht berücksichtigt, ist aber zum Vergleich mit dem Benchmark Modell nötig.

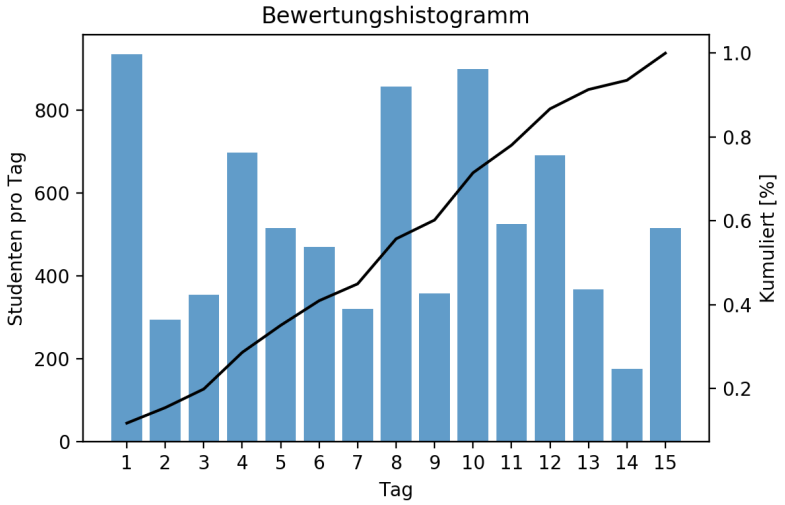

Abbildung 11: Studenten pro Tag Linearer Solver

In Abbildung 11 ist zu erkennen, dass es starke Schwankungen zwischen den einzelnen Tagen gibt, der Verlauf insgesamt aber relativ gleichmäßig ist. Es stellt sich die Frage, ob einzelne Prüfungen von Tagen mit einer großen Anzahl von geschriebenen Prüfungen an Tage mit einer geringen Anzahl umgelegt werden könnten und so den Score insgesamt noch verbessern könnten.

Der am schlechtesten bewertete individuelle Prüfungsplan sieht wie folgt aus:

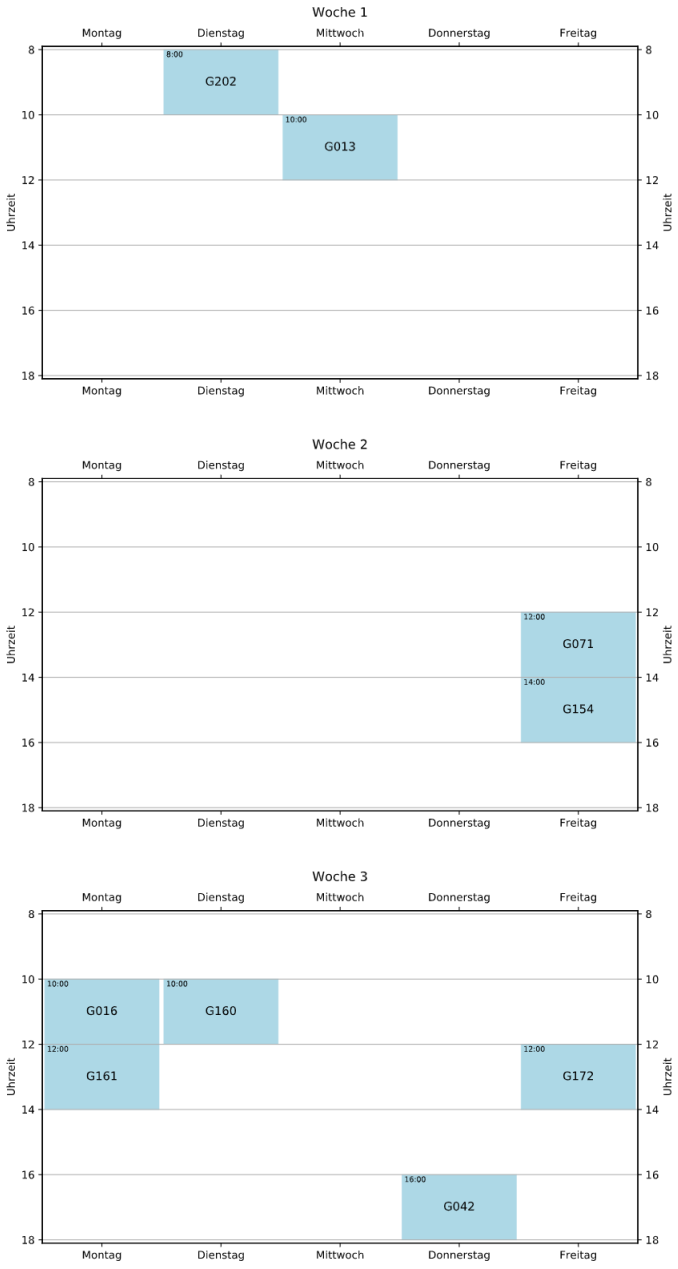

Abbildung 12: Schlechtester individueller Plan des Linearer Solvers

Ein Blick auf die doppelten Prüfungen ermöglicht eine Einschätzung, wie sinnvoll der Plan ist. Es sollten im Verhältnis zu den Studenten, die die jeweiligen Prüfungen schreiben nur wenige Studenten die Kombination haben. In Abbildung 12 erkennbar: Elektrische Antriebstechnik (G016) und Spanlose Fertigung (G161), in Kombination gewählt von 21 Studenten sowie Messtechnik (G154) und Numerik für Ingenieure (G071) in Kombination gewählt von fünf Studenten. Da dies alles relativ große Prüfungen sind, ist eine bessere Platzierung der betreffenden Prüfungen sehr unwahrscheinlich.

4.2 Simulated Annealing

4.2.1 Parameter und Simulationsstart

Zu Beginn des Algorithmus muss es bereits ein Initialstadium der Verteilung der Prüfungen geben. Diese wird zufällt generiert, kann aber auch eine Lösung eines vorherigen Durchlaufs sein. Oftmals hilft es, die Ergebnisse des ersten Durchlaufs als Startpunkt für den zweiten Durchlauf zu nehmen. Die Parameter, die den Ablauf des Prozesses wesentlich beinflussen sind folgende:

Parameter	**Funktion**
N_TEMP	Anzahl der Temperaturschritte, die durchlaufen werden
N_ITER	Anzahl an Iterationen pro Temperaturschritt
END_SCORE	Gewünschter Score zum Abbruch des Programms
MAX_ITER	Maximale Interationen des Algorithmus
p1	Chance eine schlechtere Lösung am Start zu akzeptieren
p50	Chance eine schlechtere Lösung am Ende zu akzeptieren

4.2.2 Randbedingungen

Simulated Annealing ist ein sehr einfacher Algorithmus und bietet im Gegensatz zum Linearen Solver zunächst keine Möglichkeit, Randbedingungen einzupflegen. Wie kann trotzdem gewährleistet werden, dass die Lösung diese Bedingungen erfüllt? Die Generierung von Nachbarlösungen könnte bereits auf solche Lösungen beschränkt werden, die diese Randbedingungen erfüllen. Um jedoch Simulated Annealing effizient zu nutzen, ist es wichtig, dass die Generierung und Bewertung der Nachbarlösung so schnell wie möglich passiert. Eine Ermittlung der geeigneten Lösungsmöglichkeiten würde die Laufzeit also stark verlängern.

Zum Glück kann im Falle des Prüfungsplanungsproblemes davon ausgegangen werden, dass eine gute Lösung bei einer geschickt gewählten Energiefunktion die meisten Randbedingungen automatische erfüllen wird. In diesem Fall betrachtet die Energiefunktion lediglich den Prüfungsplan jedes einzelnen Studenten. Wenn nun für den Fall, dass ein Student zwei zeitgleiche Prüfungen hat, eine hohe Strafe vergeben wird, wird die Lösung sehr wahrscheinlich diese Ereignisse vermeiden. Die Randbedingung der Kapazität sollte auch fast immer erfüllt sein, da eine gute Lösung die Prüfungen automatisch ziemlich gleichmäßig verteilen wird. Lediglich die verringerte Kapazität zur Zeit während die Masterprüfungen geschrieben werden (Wert 250 in `MaxCapacity`), stellt hier ein Hindernis dar. Durch die gleichmäßig verteilten Prüfungen sollte es allerdings möglich sein, die Reihenfolge der Prüfungen an jedem einzelnen Tag so zu ändern, dass die Kapazität nie überschritten wird. Durch die nachträgliche Änderung der Reihenfolge an einem Tag kann auch gewährleistet werden, dass die Prüfungen G013 und G160 immer um 10:00 Uhr starten. Bei

einer gleichmäßigen und energetisch günstigen Verteilung ist es auch äußerst unwahrscheinlich, dass mehr als 5 Prüfungen parallel gelegt werden. Prüfungen, deren Zeitpunkt bereits feststeht, werden nicht zur Generierung von Nachbarlösungen verwendet.

4.2.3 Reparieren der Lösung

Nachdem der Simulated Annealing Algorithmus einen zufriedenstellenden Wert erreicht, muss überprüft werden, ob die Lösung alle Randbedingungen erfüllt. Eventuell müssen dann Prüfungen an einem Tag an einen anderen Zeitpunkt gelegt werden, weil die Kapazität überschritten ist, oder Prüfungen wie G013 und G160 nicht auf die Richtige Uhrzeit gelegt wurden. Die Prüfung des Ergebnisses ist nicht aufwändig, allerdings ist der Aufwand ein Skript zu schreiben, dass die Prüfungen an eine geeignete Stelle legt nicht gering. Hier lässt sich aber sehr unkompliziert ein Linearer Solver verwenden. Die Randbedingungen werden entsprechend dem bereits vorgestellten Modell verwendet und für den Fall der Optimierung eines einzelnen Tages adaptiert. Eine Zielfunktion gibt es eigentlich nicht, denn bei der Betrachtung eines einzelnen Tages wurden nur harte Randbedingungen aufgestellt. Da der Solver aber eine Zielfunktion braucht, kann eine Variable einführt werden, die keinen Zusammenhang zum Rest der Lösung hat und nicht weiter beachtet wird. Der Solver stellt dann eine Lösung zur Verfügung, die alle Randbedingungen beachtet.

4.2.4 Nachteile des Verfahrens

Leider liefert das Verfahren bei gleicher Laufzeit stets unterschiedliche Ergebnisse bzw. es dauert unterschiedlich lang, um auf eine gleich gute Lösung zu kommen. Das Modell kann jedoch an einigen Stellen angepasst und erweitert werden, um dem entgegenzuwirken. Zum Beispiel lässt sich der Vorgang ohne Probleme parallelisieren. Dabei werden mehrere Durchläufe Gleichzeitig gestartet und am Ende das beste Ergebnis genommen. Eine Abwandlung davon selektiert die besten Ergebnisse eines parallelen Durchlaufs und mischt die Eigenschaften dieser Modelle um einen besseren Startpunkt für die nächste Iteration zu haben. Eine weitere Variante, das Adaptive Simulated Annealing, passt die Geschwindigkeit der Abkühlung der Temperatur an die Veränderung des Scores an.

4.2.5 Ergebnisse

Zur Prüfungsplanung wird das Simulated Annealing verfahren solange angewandt, bis ein gewünschter Score erreicht ist oder eine maximale Anzahl von Durchläufen erreicht ist. Ein Graph über den zeitlichen Verlauf des Scores zeigt, wie das Verfahren arbeitet.

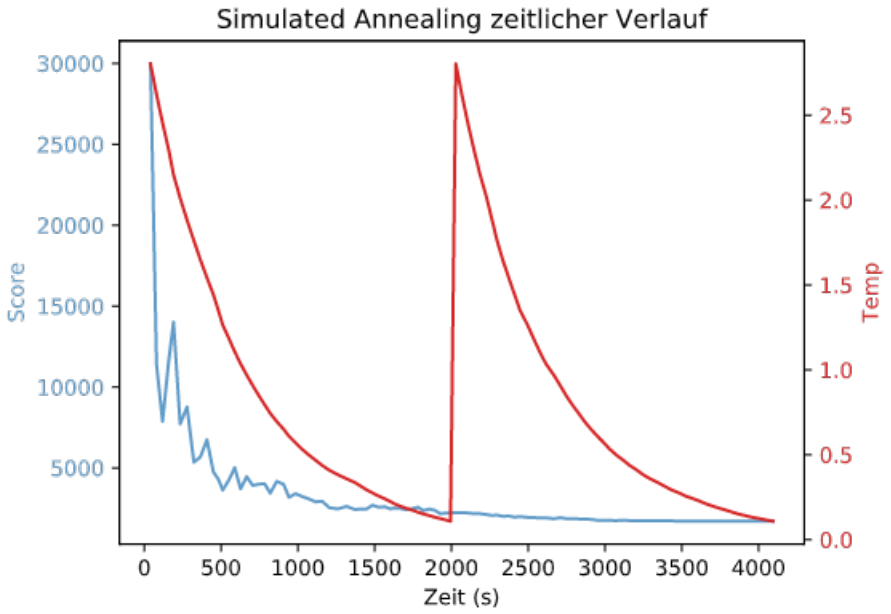

Abbildung 13: Zeitlicher Verlauf des Simulated Annealing

Abbildung 13 zeigt, dass der Score anfangs sehr hoch ist, da die Prüfungen ungeordnet sind. Mit jeder Iteration des Modells erfolgen einige Änderungen und des Score besser sich kontinuierlich. Nach dem ersten Durchlauf ist der Score noch nicht unter dem gewünschten Wert und die Temperatur wird erneut erhöht. Nach in dem zweiten Durchlauf sind die Veränderungen kleiner, da der Zustand schon insgesamt gut geordnet ist. Am Ende des zweiten Durchlauf ist der gewünschte Score erreicht und das Verfahren wird beendet. Auch bei diesem Modell erfolgt die erste Bewertung anhand eines Histogramms, gezeigt in Abbildung 14.

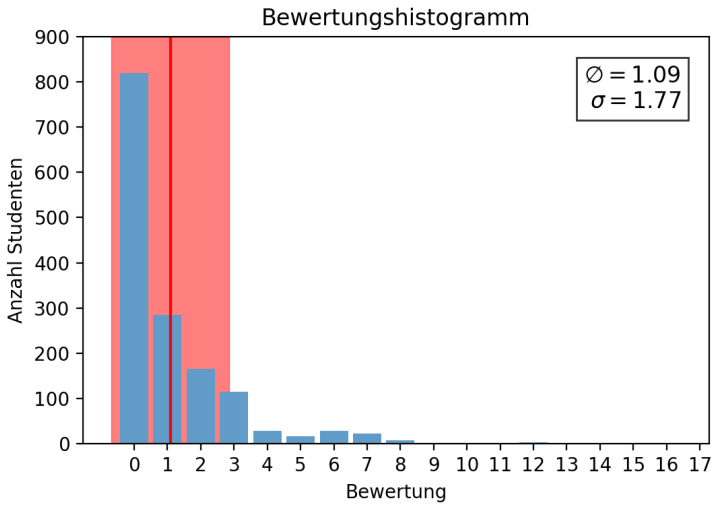

Abbildung 14: Ergebnishistogramm Simulated Annealing

Insgesamt ist das Ergebnis ähnlich zu dem des Linearen Solvers, allerdings leicht

schlechter. Daraus kann folgendes schlussgefolgert werden: Da das Verfahren des Simulated Annealing bei solchen Planungsproblemen üblicherweise sehr gute, aber keine idealen Lösungen generiert, ist davon auszugehen, dass die ideale Lösung für das Problem nicht viel besser ist. Somit ist bestätigt, dass die Entkopplung des Linearen Solver in Wochen- und Tagesplanung sowie das Planen von Prüfungsgruppen sinnvoll ist, da dies zu einem statistisch gesehen sehr gutem Ergebnis führt.

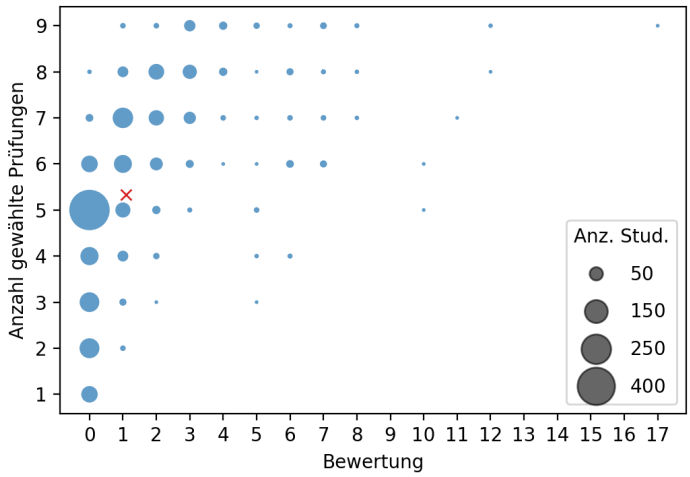

Abbildung 15: Korrelation gewählte Prüfungen und Bewertung Simulated Annealing

In Abbildung 15 ist wieder einen großen Punkt bei (0; 5) erkennbar. Durch die zufälligen Vertauschungen des Algorithmus und deren Bewertung wurden die Prüfungen der Erstsemester wie im Linearen Solver konfliktfrei geplant.

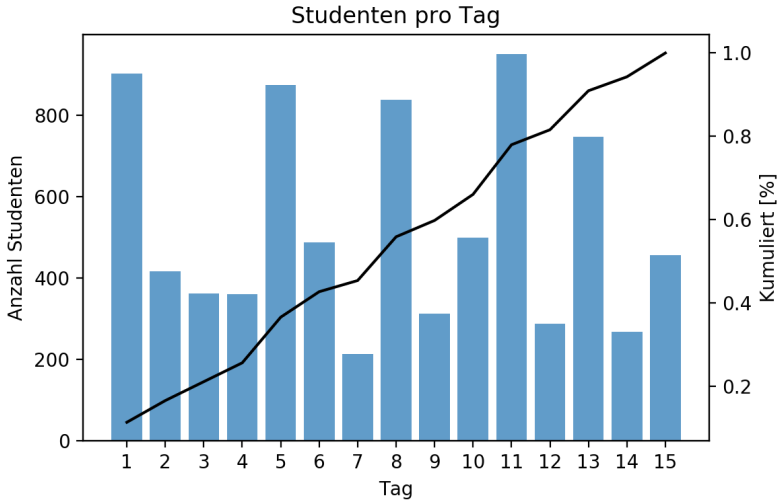

Abbildung 16: Studenten pro Tag Simulated Annealing

In Abbildung 16 zeigt sich betreffend der Anzahl der Studenten, die insgesamt pro Tag eine Prüfung schreiben, ein Verlauf ähnlich zu denen des Linearen Solvers. Die Schwankungen zwischen den Tagen sind jedoch deutlich höher.

Der am schlechtesten bewertete individuelle Prüfungsplan sieht wie folgt aus:

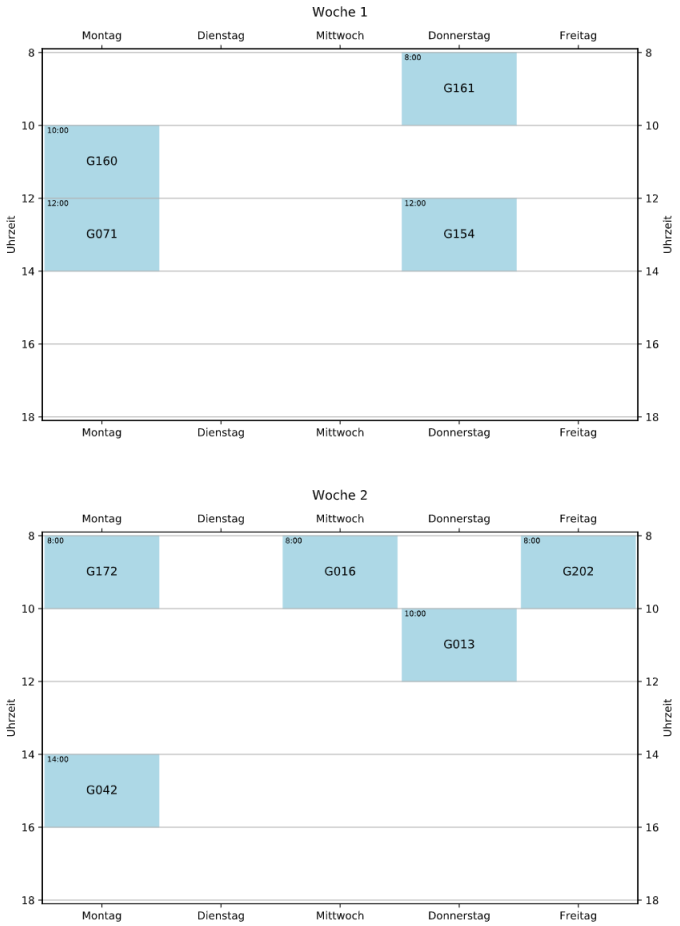

Abbildung 17: Schlechtester Plan des Simulated Annealing

In Abbildung 17 fällt auf, dass die Prüfungen des individuell schlechtesten Plans nur auf zwei Wochen verteil wurden. Auch hier ist eine Untersuchung der doppelten Prüfungen nötig. Spanende Fertigung und Betriebsorganisation (G160) und Numerik für Ingenieure (G071) wurden von 17 Studenten in Kombination gewählt. Spanlose Fertigung (G161) wurde in Kombination mit Messtechnik (G154) acht mal gewählt. Fahrzeugtechnik (G042) und Technische Mechanik II (G172) wurden von elf Studenten in Kombination gewählt.

4.3 Benchmark-Modell

Zum Vergleich soll hier noch ein Modell, dass bereits an der Fakultät existiert erwähnt werden, was wesentlich schneller läuft und mit einer anderen Zielfunktion und Restriktionen insbesondere zum Ziel hat, große Prüfungen möglichst kollisionsfrei an den Anfang des Prüfungszeitraumes zu legen.

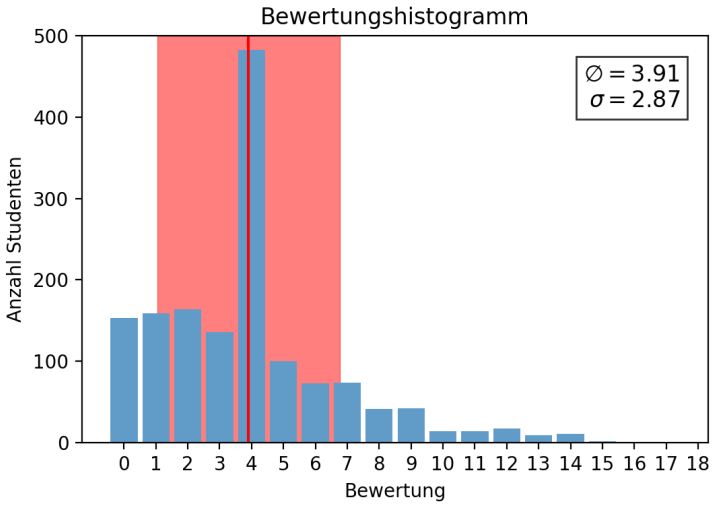

Abbildung 18: Ergebnishistogramm Benchmark-Modell

Am Histogramm in Abbildung 18 lässt sich erkennen, dass durch die fehlende zeitliche Optimierung wesentlich mehr Studenten Prüfungen an aufeinanderfolgenden Tagen haben. Dadurch ist der Durchschnitt und die Standartabweichung um einiges größer.

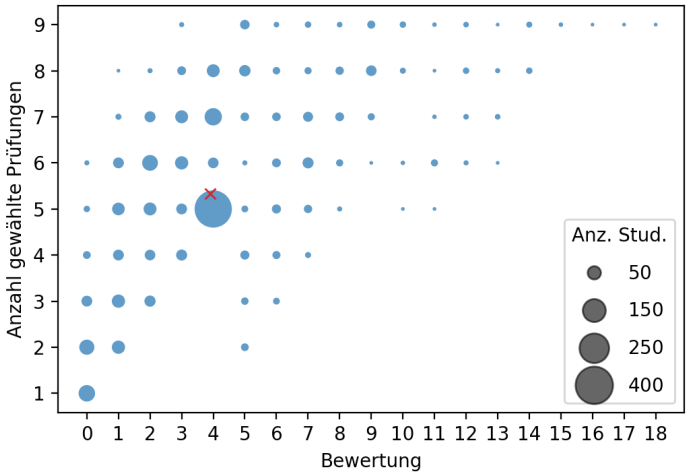

Abbildung 19: Korrelation gewählte Prüfungen und Bewertung Benchmark-Modell

Bei einer genaueren Betrachtung der Verteilung in Abbildung 19 lässt sich erkennen, dass es hier Studenten gibt mit nur zwei gewählten Prüfungen, die diese am gleichen Tag schreiben müssen. Der größte Punkt bei (4;5) lässt die Gruppe der Erstsemester vermuten. Ein großer Teil des Scores kommt allein durch diesen Punkt zustande. Dadurch ist auch der Durchschnittliche Prüfungsplan höher bewertet. Der schlechteste individuelle Score ist mit dem des Simulated Annealing vergleichbar.

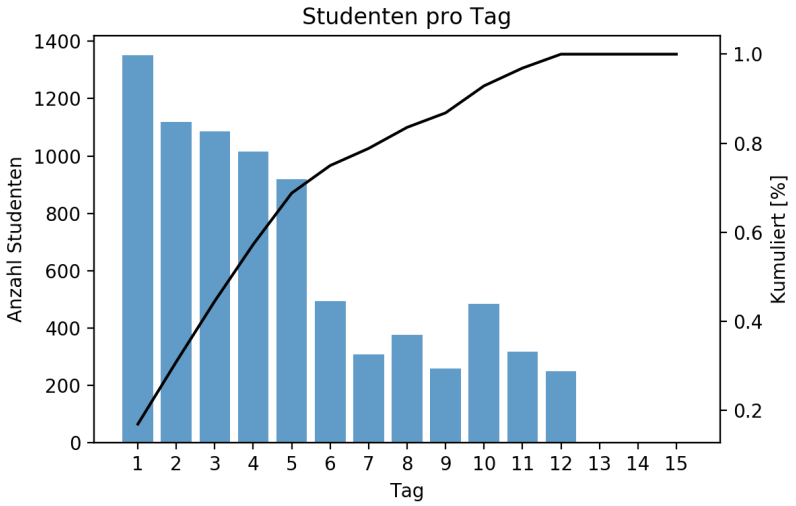

Abbildung 20: Studenten pro Tag Benchmark Modell

Bezüglich der Anzahl der Studenten, die täglich Prüfungen schreiben, zeigt sich in Abbildung 20 ein großer Unterschied zu den vorherigen. Insbesondere in der ersten Woche werden viel mehr Prüfungen geplant als in den beiden anderen Modellen. Außerdem sind bereits nach Tag 12 alle Prüfungen abgehalten worden. Dies gibt den Professoren und Lehrbeauftragten mehr Zeit, um Prüfungen fristgerecht zu korrigieren.

Der am schlechtesten bewertete Prüfungsplan sieht wie folgt aus:

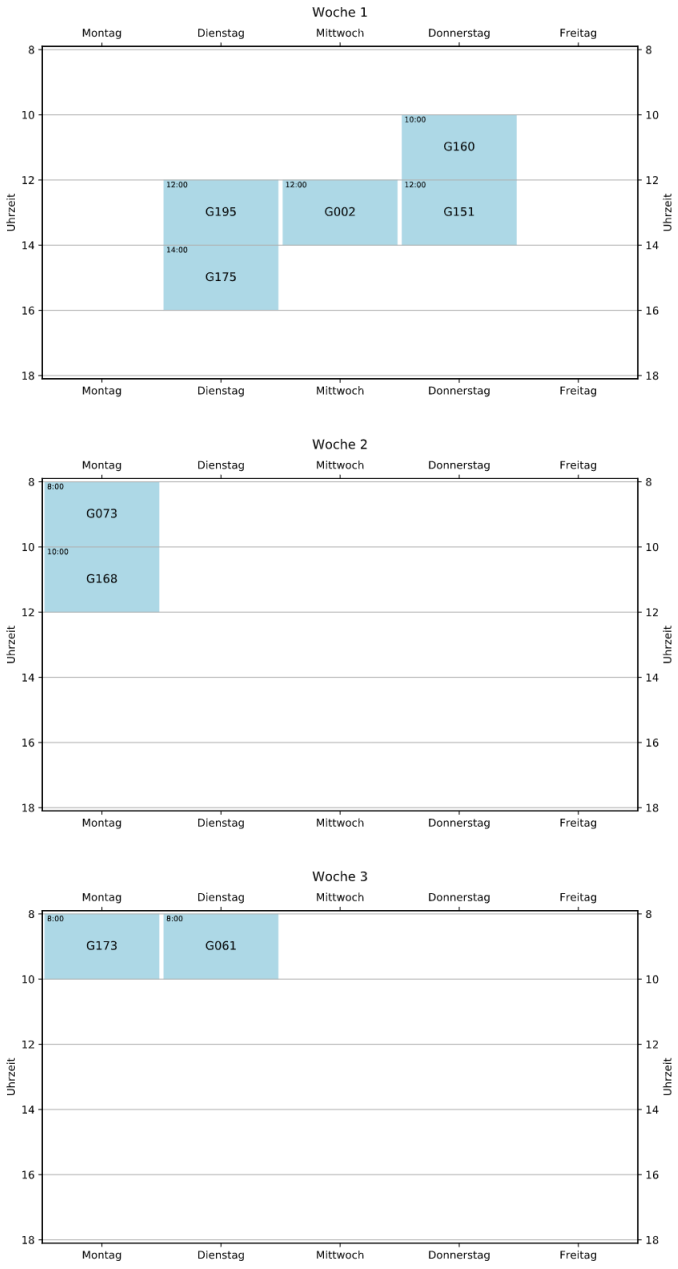

Abbildung 21: Schlechtester Plan des Benchmark Modells

Folgende Kombinationen treten in Abbildung 21 auf: Spanende Fertigung und Betriebsorganisation (G160) und Recht(G151) 24 Studenten; Thermodynamik(G175) und Bauelemente der Luftfahrzeuge II (G195) elf Studenten; Ingenieurmathematik II (G073) und Technische Dynamik (G168) 38 Studenten. Dies sind von allem große Prüfungen also auch keine besonders auffälligen Ergebnisse.

5 Vergleich der Ergebnisse

Um einen einfacheren Vergleich zu ermöglichen, werden die Ergebnisse parallel präsentiert.

	Modell		
Eigenschaft	**Simulated Annealing**	**Linearer Solver**	**Benchmark-Modell**
Laufzeit in s	ca. 1800 - 6000	ca. 3600	ca. 500
Score	1613	1445	5839
indiv. Score	17	17	18

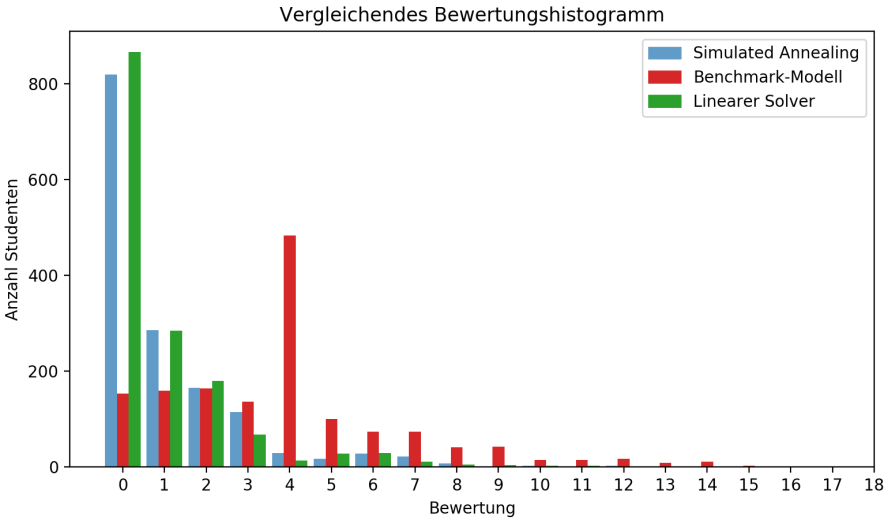

Abbildung 22: Vergleich der Ergebnishistogramme

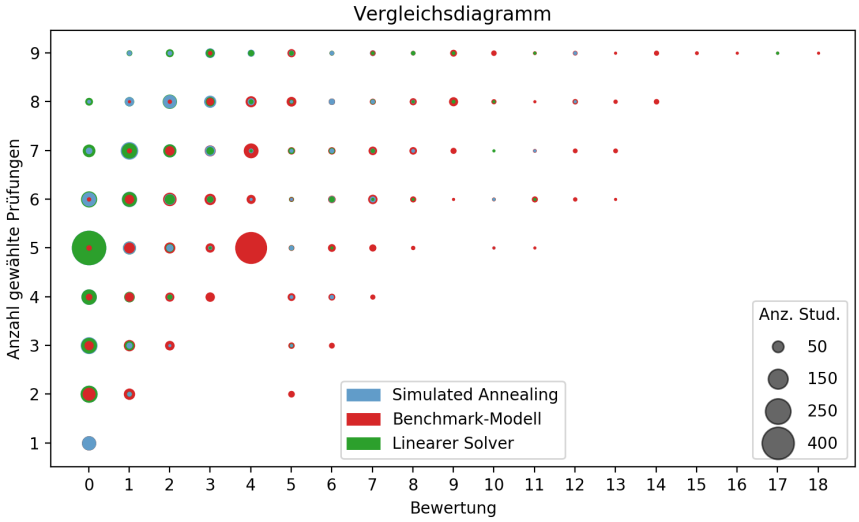

Abbildung 23: Vergleich der Korrelation der gewählten Prüfungen und Bewertungen

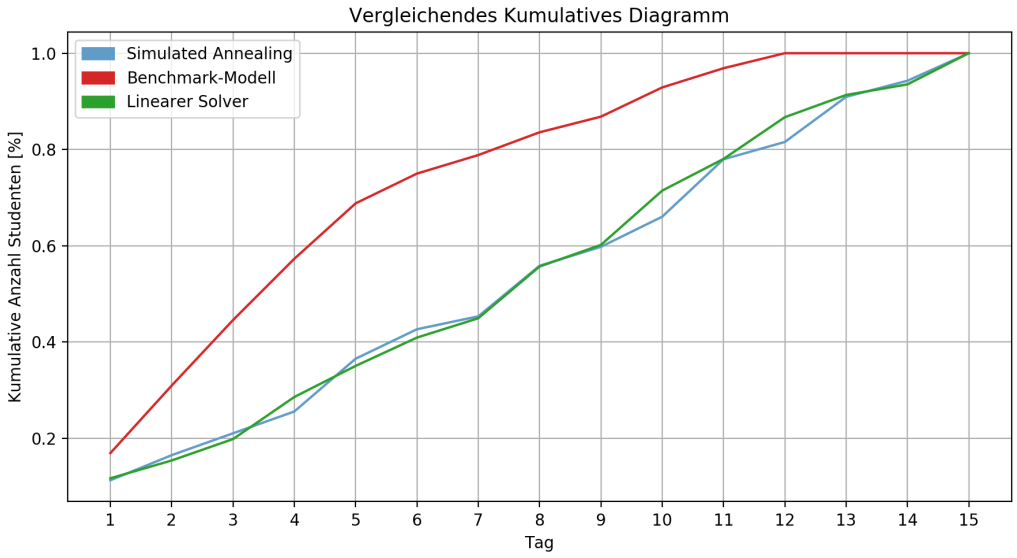

Abbildung 24: Vergleich der Studenten pro Tag

In den Abbildungen 22 - 24 werden die unterschiedlichen Zielsetzungen der Modelle erkennbar. Simulated Annealing sowie der Lineare Solver sind darauf getrimmt, die zeitliche Belastung der Studenten zu minimieren, das Benchmark-Modell hingegen darauf, möglichst viele Prüfungen an den Anfang der Prüfungszeitraumes zu legen.

Es fällt auf, dass die Lösungen des Simulated Annealing und des Linearen Solvers teilweise große Ähnlichkeiten besitzen. In an dem Punkt bei (0; 5) in Abbildung 23 erkennt man, dass beide die Prüfungen der Erstsemester konfliktfrei planen. Und auch in Abbildung 24 fällt es auf, dass die kumulative Anzahl der geschriebenen Prüfungen insbesondere von Tag 7 - 9 und 13 - 15 fast identisch ist.

Die Laufzeiten der Modelle unterscheidet sich sehr stark. Das Benchmark-Modell braucht nur ein paar Minuten. Der in Wochen und Tagesplanung entkoppelte Lineare Solver braucht hingegen etwa eine Stunde. Das Simulated Annealing Verfahren braucht unterschiedlich lange, um eine gute Lösung zu erreichen. Es kann prinzipiell schneller sein als der Lineare Solver, braucht aber in der Regel genau so lange oder länger, um eine vergleichbar gute Lösung zu erreichen.

6 Die Suche nach der Ideallösung

Alle vorgestellten Modelle vereinfachen oder entkoppeln das Problem in gewisser Weise zugunsten der Laufzeit. Gibt es jedoch eine Möglichkeit, die tatsächlich beste Platzierung aller Prüfungen aus den Ergebnissen der einzelnen Modelle zu ermitteln?

Bei einigen Linearen Solver gibt es die Möglichkeit zu Beginn bereits eine Lösung vorzugeben. Dadurch wird früh eine Obergrenze definiert, und der Solver kann Nodes mit schlechteren Lösungen schneller verwerfen. Zunächst können die Ergebnisse des entkoppelten Linearen Solvers für einen warmen Start des neuen Linearen Solvers genutzt werden. Dieser soll nun alle Prüfungen auf einmal planen und sie nicht wie vorher nach Größe und Semester gruppiert abarbeiten.

Leider bieten die Google OR-Tools nicht die Möglichkeit, Startwerte für das Problem anzugeben, es gibt jedoch einige andere Python Solver APIs, die diese Möglichkeit haben. Der bekannteste kommerzielle Solver heißt Gurobi und dieser bietet diese Möglichkeit. Außerdem soll Gurobi Optimierungsprobleme deutlich schneller als die Freeware-Alternativen bearbeiten können.

Der Code zur Formulierung des Modells in Gurobi unterscheidet sich nur unwesentlich von dem bereits beschriebenen Modell für Tage-Optimierung (Abschnitt 4.1) und wird nicht weiter erläutert. Lediglich die Ergänzungen werden hier gezeigt. Die Startlösung wird wie folgt angegeben:

Listing 3: Angabe einer Startlösung

```
# initialise all start values
for week in range(N_WEEKS):
    for day in range(N_DAYS_PER_WEEK):
        for slot in range(N_SLOTS):
            for exam in range(N_EXAMS):
                exam_placing[week,day,slot,exam].start = 0

# set startvalues for good solution
for i_exam in start:
    exam = i_exam[0]
    week = i_exam[1]
    day = i_exam[2]
    slot = i_exam[3]
    exam_placing[week,day,slot,exam].start = 1
```

Zunächst werden alle Platzierungsvariablen auf 0 gesetzt und anschließend alle Werte der letzten Lösung als Startwerte übergeben.

Ein warmer Start hat jedoch auch noch nach einigen Stunden einen Gap zwischen Ober- und Untergrenze von über 90%, dementsprechend scheint es auch so nicht möglich, eine ideale Lösung innerhalb von einer akzeptablen Zeit zu erreichen. Als weiteren Versuch, den Lösungsprozess zu beschleunigen, können die Platzierung der

Prüfungen mit Gewichten versehen werden. Ohne Gewichte gibt es mehrere gleichwertige Lösungen (zum Beispiel macht es keinen Unterschied für den Lösungsscore, wenn die Wochen am Ende vertauscht werden). Solche Äquivalenzen machen es dem Solver schwer, Lösungsmöglichkeiten im Suchbaum frühzeitig zu verwerfen und sorgen deswegen für eine längere Laufzeit.

Die Gewichte sollen den Solver nur in eine bestimmte Richtung leiten, aber die Endlösung soll die bestmögliche Lösung aus Studentensicht sein. Dementsprechend sind alle Gewichte etwa gleich eins oder kleiner. Die Gewichte sollen weiterhin so ausgewählt sein, dass sie Lösungen erzeugen, bei denen größere Prüfungen eher am Anfang des Prüngszeitraumes liegen. Die Gewichte werden wie folgt zu dem bestehenden Modell hinzugefügt:

Listing 4: Zusätzliche Gewichte

```
n_days = N_WEEKS*N_DAYS_PER_WEEK
weight_day = [(n_days - d)/n_days for d in range(n_days)]
# looks like this: [1 , 0.93 , 0,87 ...]
weight_slot = [0.6, 1, 0.8, 0.4, 0.2]
    f_time = gp.quicksum(exam_placing[week,day,slot,exam] *
    (weight_day[day+week*5]+weight_slot[slot]) * exam_sizes[exam]
    for exam in range(N_EXAMS)
    for slot in range(N_SLOTS)
    for day in range(N_DAYS_PER_WEEK)
    for week in range(N_WEEKS))
```

Dabei sind die Größen der Prüfungen, also die Anzahl der Studenten die sie schreiben normiert auf die größte Prüfung angegeben. Der Vektor exam_sizes ist also gleich eins für die größte Prüfung und kleiner für alle anderen. Da dieser Term größer wird, je weiter vorne eine große Prüfung platziert ist, muss er von der bestehenden Zielfunktion subtrahiert werden. Die neue Zielfunktion sieht also wie folgt aus, wenn der Rest aus der Bewertungsfunktion für die Tage-Planung übernommen wird:

Listing 5: Neue Zielfunktion

```
exam_cost = daily_double_count*5 + consecutive_count - f_time
m.ModelSense = GRB.MINIMIZE
m.setObjective(exam_cost)
```

Leider ergibt sich auch hierdurch kein großer Geschwindigkeitsboost. Der Gap zwischen Ober- und Untergrenze schließt sich etwa gleich schnell wie bei dem Modell mit der Startlösung ohne Gewichte. Ein Unterschied ist jedoch, dass mehr Nodes innerhalb des Suchbaumes abgearbeitet werden.

7 Ausblick

Die Ergebnisse dieser Arbeit weisen darauf hin, dass es in einer relativ kurzen Zeit möglich ist, die Prüfungen durch eine Entkopplung des Problems gleichmäßig zu platzieren. Der nächste logische Schritt scheint daher die Vereinigung des Benchmark Modells mit den neuen Ergebnissen dieser Arbeit zu sein. Es wäre denkbar, die Wochenplanung durch das Benchmark Modell zu erstellen, die Tagesplanung anschließend von dem neuen Modell. Alternativ könnten auch Gewichte zu dem neuen Modell hinzufügt werden, die große Prüfungen am Anfang priorisieren. Letztendlich muss dieser Zielkonflikt persönlich beurteilt werden. Der Autor dieser Arbeit würde es begrüßen, wenn in Zukunft ein Modell zur Prüfungsplanung genutzt wird, welches einen Kompromiss zwischen den Modellen schafft.

Weiterhin wäre es interessant zu untersuchen, ob mit Hilfe der Bewertungsfunktion auch eine künstliche Intelligenz auf das Problem trainiert werden kann. Die Daten hierzu könnten durch das Simulated Annealing und den Linearen Solver erstellt werden. Bei dem Linearen Solver müsste dazu nur jeweils zu Beginn für eine beliebige weitere Prüfung der Termin festlegt werden, um so eine Menge von ähnlichen Lösungen zu generieren. Eine künstliche Intelligenz kann vermutlich Pläne in der Qualität des Simulated Annealing erstellen, aber braucht dafür deutlich weniger Zeit.

Die Funktion des Programms könnte auch auf den Kopf gestellt und an andere ähnliche Probleme angepasst werden. Zum Beispiel könnten Stundenpläne so erstellt werden, dass es möglichst wenige Lücken zwischen Unterrichtsblöcken gibt.

Der beste Weg zum Gipfel eines Berges ist oft nicht eindeutig, aber in jedem Fall ist die Strecke dorthin mit großer Anstrengung verbunden. Ähnlich gestaltet sich die Suche nach der bestmöglichen Lösung in einem großen Planungsproblem. Es ist fraglich, ob es je einen idealen Prüfungsplan geben wird, allein schon weil die Definition des Ideals sehr subjektiv ist. Solange Planungsprobleme Bestandteil aktueller Forschung sind, bleibt die Suche nach der idealen Lösung spannend.

Literatur

[1] Tobias Achterberg: *Constraint Integer Programming (Ph.D. Dissertation)*. Technische Universität Berlin, 2007.

[2] Ed Klotz, Alexandra M. Newman: *Practical Guidelines for Solving Difficult Mixed Integer Linear Programs*. Colorado School of Mines, Golden, CO 80401, 2013.

[3] Serge Kruk: *Practical Python AI Projects*. Apress, 2018.

[4] Gurobi Optimization, LLC: *Gurobi Optimizer Reference Manual*. http://www.gurobi.com, 2020. Aufgerufen Okt. 2020.

[5] Nuno Leitea, Fernando Melicioa, Agostinho C. Rosac: *A fast simulated annealing algorithm for the examination timetabling problem*. University of Lisboa, 2019.

[6] Cuong Do: *Exam Scheduling Optimization with Simulated Annealing Initial Research*. University of Connecticut, 2019.

[7] Eric Stoltz: *Scheduling with ease: Cost optimization tutorial for Python*. https://www.towardsdatascience.com, 2018. Aufgerufen Okt. 2020.

[8] T. C. Koopmans, M. J. Beckmann: *Assignment problems and the location of economic activities*. Econometrica 25, 1957

[9] Yong Xia, Ya-xiang Yuan: *A new linearization method for quadratic assignment problems*. Chinese Academy of Sciences, 2006

[10] Ulrike von Luxburg: *A Tutorial on Spectral Clustering*. Statistics and Computing 17 (4), 2007

Abbildungsverzeichnis