Bachelorarbeit

Universität Augsburg
Fakultät für Angewandte Informatik

Faire und effiziente überschneidungsfreie Veranstaltungszuteilung in Digicampus

Dat Le Thanh

Geburtsdatum: 07. August 1998
Matrikelnummer: [...]
Studiengang: B. Sc. Informatik und Multimedia

5. Dezember 2019

Erstgutachter: Prof. Dr. Robert Lorenz
Zweitgutachter: Prof. Dr. Bernhard Bauer

Inhaltsverzeichnis

1 Einleitung

An der Universität Augsburg werden jedes Semester Lehrveranstaltungen zu den Grundlagen der Informatik angeboten, die von bis zu 650 Studierenden besucht werden. Die meisten Lehrveranstaltungen bieten neben Vorlesungen auch stoffvertiefende Übungen an, die von Studierenden höheren Semesters in Kleingruppen von etwa 25 bis 30 Personen gehalten werden. Beispielsweise hat es zur Lehrveranstaltung *Informatik 1* im Wintersemester 2019/2020 insgesamt 22 Übungsgruppen gegeben, die über die ganze Woche verteilt angeboten werden. Jeder Studierende soll neben der Vorlesung möglichst auch eine korrespondierende Übung besuchen. Da Grundlagenveranstaltungen oft in verschiedenen Studiengängen vertreten sind, kommt es zu zeitlichen Überschneidungen in der Planung. In diesem Fall ist das Ziel also, möglichst allen Studierenden bezüglich ihres gewünschten Stundenplans gerecht zu werden. Darunter fällt auch, zeitliche Überschneidungen zwischen den Terminen der Übungen zu vermeiden, sodass aufgrunddessen die Anzahl von Übungsgruppenwechsel möglichst gering gehalten wird.

Jeder Vorlesung und jeder Übung wird im Voraus ein geeigneter (wöchentlicher) Termin und Raum zugewiesen, zu welchen sich die Studierenden zu Anfang eines Semesters über die Lehr- und Lernplattform *Digicampus* der Universität Augsburg anmelden können [17, 24]. Dabei gibt es verschiedene Anmeldemodi, die angewendet werden können.

„First Come First Served“ (FCFS) ist eines der einfachsten Methoden zur Veranstaltungsanmeldung. Dabei wird zu einem angekündigten Zeitpunkt die Anmeldung freigegeben und diese ist dann solange möglich, bis die Kapazitäten der Veranstaltung erschöpft sind. Obwohl diese Methode von vielen Hochschulen genutzt wird, ist das „First Come First Served“-Prinzip nicht ohne Probleme. Studierende mit erhöhter Bedürftigkeit, also diejenigen, die sonst keine andere Übung besuchen können, werden nicht berücksichtigt. Weiterhin ist es möglich, dass nicht alle die gleiche Chance haben, sich zur gewünschten Veranstaltung anzumelden. Studierende, die ohne Eigenverschulden den Start der Anmeldung verpasst oder nicht mitbekommen haben, werden benachteiligt. Überschneidungsfreiheit liegt in der Verantwortung der Studierenden, dass sie sich früh genug in ihre Veranstaltungen eintragen.

Digicampus bietet weiter auch die Möglichkeit der Anmeldung zu Veranstaltungen mithilfe eines Losverfahrens unter Abgabe von Prioritäten. Im Wesentlichen werden für alle vergebenene Prioritäten nacheinander alle Veranstaltungen ausgelost: Zunächst wird eine zufällige Reihenfolge von Studierenden mit Priorität 1 zu einer bestimmten Veranstaltung bestimmt, die nacheinander in diese Veranstaltung eingetragen werden. Dann folgt die Priorität 2 und wieder werden alle Studierenden in Veranstaltungen, die sie als zweite Priorität angegeben haben, gelost. Falls in einer Losrunde bereits alle Plätze einer Veranstaltung vergeben sind, bekommen nicht zugeteilte Studierende für die nächste Runde einen Bonus, sodass die Wahrscheinlichkeit steigt, in eine gewünschte Veranstaltung zugeteilt zu werden. Die Ergebnisse dieser Verteilung sind für Studierende teilweise nicht zufriedenstellend, da dieser Verteilungsmechanismus keine Maximierung der Anzahl Studierende, die ihre am höchsten priorisierten Veranstaltungen erhalten, vorsieht. Zwar kann Digicampus innerhalb eines Anmeldeverfahrens mehrere Veranstaltungen zuteilen, bietet keine Möglichkeit der Gruppierung von Veranstaltungen und achtet zudem nicht auf Überschneidungsfreiheit [22].

Bisher ist keine Lernplattform bzw. kein Campus-Management-System bekannt, welches an deutschen Hochschulen eingesetzt wird und eine prioritätenbasierte, überschneidungsfreie Zuteilung zwischen verschiedener Lehrveranstaltungen ermöglicht. Oftmals werden eigens entwickelte Lösungen eingesetzt, wie das Matching-System an der Fakultät für Informatik an der Technischen Universität München [18, 3] oder die „Mathematisch Optimale Stundenplan-Erstellungs-Software“ (Moses) der Technischen Universität Berlin [13]. Am Institut für Informatik der Universität Augsburg wurde für die Übungsgruppenverteilung bis zum 1. Oktober 2018 das vom

Lehrstuhl des Prof. Dr. Hagerup bereitgestellte Vorlesungsverwaltungssystem verwendet [29, 8].

Diese Bachelorarbeit beschäftigt sich mit der Aufgabe, die Lehr- und Lernplattform Digicampus um ein weiteres Anmeldeverfahren zu erweitern, welches möglichst optimal und überschneidungsfrei Studierende ihren gewünschten Veranstaltungen zuteilt.

Kapitel 2 beginnt mit einer Einführung in das Zuteilungsproblem. Die Problemstellung wird erläutert und dabei werden notwendige Notationen und Begriffe definiert. Weiterhin werden wünschenswerte Eigenschaften von Zuteilungenmechanismen beschrieben. Schließlich wird ein Überblick über einige Mechanismen gegeben.

In Kapitel 3 wird der gewählte Mechanismus „Bundled Probabilistic Serial“ (BPS) genauer beschrieben. Zunächst wird in Kapitel 3.1 das Erzeugen von Präferenzen über Veranstaltungsbündel gezeigt. Anhanddessen soll eine zufällige kombinatorische Zuteilung generiert werden. Im Detail wird dieser Vorgang in Kapitel 3.2 beschrieben. Kapitel 3.3 erläutert das Zerlegen der zufälligen kombinatorischen Zuteilung in eine Linearkombination von zulässigen deterministischen kombinatorischen Zuteilungen näher. Abschließend wird in Kapitel 3.4 die Implementierung des Mechanismus in Python beschrieben.

Kapitel 4 beschreibt den Digicampus der Universität Augsburg und die Integration des Mechanismus in die Plattform. Zunächst wird in Kapitel 4.1 das Kernsystem und dessen Pluginmechanismus erläutert und in Kapitel 4.2 die technische Funktionsweise der Anmeldesets und ihre zugehörigen Anmelderegeln. Kapitel 4.3 beschreibt näher die Aufgaben und die Implementierung des Plugins, welches ein neue Anmeldeverfahren bereitstellt.

Anhand von Daten aus dem Sommersemester 2018 und dem Wintersemester 2019/2020 wurden beispielhafte Zuteilungen errechnet. Kapitel 5.1 erläutert dabei kurz den Datensatz und Kapitel 5.2 zeigt die Ergebnisse des angewandte Mechanismus.

Abschließend wird in Kapitel 6 diese Bachelorarbeit zusammengefasst und es wird ein Ausblick auf mögliche Erweiterungen gegeben.

2 Theoretischer Hintergrund

Wie können Plätze in Veranstaltungen möglichst „gut und fair“ unter einer Menge von Studierenden aufgeteilt werden? Instanzen des Zuteilungsproblem finden sich an fast allen Universitäten. In diesem Kapitel wird das Zuteilungsproblem näher erläutert und die gewünschten Eigenschaften von „guter und fairer“ Zuteilung gezeigt.

2.1 Zuteilungsproblem

Das **Zuteilungsproblem** beschreibt dabei die Zuteilung einer endlichen Anzahl von unteilbaren Objekten an eine ebenfalls endliche Menge von Agenten, wobei im universitären Kontext die Objekte einer Teilnahme an einem Kurs entsprechen und die Agenten die Studierenden sind. Jeder Agent hat dabei Präferenzen zu seinen gewünschten Objekten ausgedrückt, die ohne Einsatz eines Marktmechanismus bestmöglichst erfüllt werden sollen. Angebot und Nachfrage sollen demnach nicht über Austausch von Geldmitteln geregelt werden. Im **kombinatorischen Zuteilungsproblem** können die Agenten nicht nur ein, sondern auch Kombinationen von mehreren Objekten als Bündel beanspruchen. Das bedeutet im Kontext der Übungsgruppenzuteilung am Institut für Informatik, dass mehrere Übungen aus verschiedenen Lehrveranstaltungen dem Studierenden zugeteilt werden soll.

2.1.1 Bezeichnung *Es gibt eine endliche Menge an **Studierenden** (bzw. Agenten)* $S = \{1, 2, \ldots, n\}$.

2.1.2 Bezeichnung *Es gibt eine endliche Menge von **Kursen** (bzw. Objekten)* $C = \{c_1, c_2, \ldots, c_m\}$ *mit einer maximalen Teilnehmendenanzahl* $q = (q_1, q_2, \ldots, q_m)$

2.1.3 Bezeichnung *Ein **Veranstaltungsbündel*** $b = (c_1, \ldots, c_k)$ *ist ein Tupel aus den Elementen von* C *und besteht aus einer Kombination von* k *Kursen.*

2.1.4 Bezeichnung B *sei die Menge aller Veranstaltungsbündel.* $B_i \subset B$ *sei die Menge von Veranstaltungsbündel des Studierenden* i *und enthalte nur Bündel von Kursen der Lehrveranstaltungen, die der Studierende besucht.*

Die Menge aller möglichen Reihenfolgen von möglichen Veranstaltungsbündel werde als $\mathcal{P}$ bezeichnet. Jeder Studierende $i \in S$ hat eine vollständige und transitive **Präferenzrelation** $\succ_i \in \mathcal{P}$ über seine Veranstaltungsbündel B_i. Es ist zu beachten, dass jeder Studierende eine unterschiedliche Anzahl von Lehrveranstaltungen besuchen kann. Die Länge k eines jeden Bündels $b \in B_i$ eines Studierenden i richtet sich daher nach der Anzahl der Lehrveranstaltungen, an dessen Übungsbetrieb er teilnehmen möchte.

Beschreiben beispielsweise a und b jeweils Veranstaltungsbündel sowie gilt weiterhin $a \succ_i b$, $a \neq b$, dann steht dies für ein Bevorzugen des Studierenden i von Veranstaltungsbündel a gegenüber Bündel b. In der Theorie werden strikte Präferenzen angenommen, jedoch soll diese Annahme im späteren Verlauf diskutiert werden.

Das Präferenzprofil $\succ = (\succ_1, \succ_2, \ldots, \succ_n) \in \mathcal{P}^{|S|}$ ist ein n-Tupel von Präferenzrelationen und beschreibt im Gesamten die verschiedenen Präferenzen aller Studierenden.

2.1.5 Definition *Eine **deterministische kombinatorische Zuteilung** ist eine injektive Abbildung* $\varphi : S \mapsto B$ *und ordnet jedem Studierenden ein Veranstaltungsbündel zu.*

Ein Veranstaltungsbündel soll nun als binärer Vektor $b \in \{0,1\}^{|C|}$ repräsentiert werden, wobei $b_j = 1$ gilt, wenn Kurs c_j im Bündel enthalten ist, ansonsten gilt $b_j = 0$. Die Größe eines Veranstaltungsbündels sei weiterhin als $size(b) = \sum_{j=1}^{m} b_j$ definiert. Schließlich sei ein Zuteilungsvektor $x \in \{0,1\}^{|S \times B|}$ definiert, wobei $x_{ib} = 1$ gilt, wenn dem Studierenden i das Bündel b zugeteilt wurde, ansonsten gilt $x_{ib} = 0$. Mit diesen gegebenen Definitionen können Angebot und Nachfrage als lineare Restriktionen modelliert werden.

$$\sum_{i \in S,\ b \in B} x_{ib}\ b_j \leq q_j \qquad j \in C \tag{Angebot}$$

$$\sum_{b \in B} x_{ib} \leq 1 \qquad \forall i \in S \tag{Nachfrage}$$

$$x_{ib} \in \{0,1\} \qquad \forall i \in S, b \in B \tag{Zugeteilt}$$

Angebot sagt aus, dass die Anzahl der zugewiesenen Bündel, die Kurs c_j enthalten, die maximale Teilnehmendenanzahl von c_j nicht überschreiten darf. Weiterhin darf kein Studierender mehr als ein Bündel zugeteilt bekommen (Nachfrage). Im Rahmen der Übungsgruppenzuteilung sei die maximale Größe eines Veranstaltungsbündels $size(b) \leq k < m$. Der Studierende besucht in der Regel nur jeweils eine Übungsgruppe von seinen k Lehrveranstaltungen. In jedem Fall soll der Studierende nicht alle m angebotenen Übungstermine aller Lehrveranstaltungen wahrnehmen. Eine deterministische Zuteilung ist dann **zulässig**, wenn sie Angebot und Nachfrage erfüllt.

Es werden weiter auch Wahrscheinlichkeiten von Zuteilungen betrachtet. Hierzu benötigt es die **zufällige kombinatorische Zuteilung**, welche jedem Agenten im Allgemeinen „Teile von Objektbündeln“ zuordnet. Das zugeordnete Teil des Bündels werde dann als Wahrscheinlichkeit, dass der Agent das Bündel erhält, interpretiert. Konkret bedeutet es, dass dem Studierenden Wahrscheinlichkeiten zu Veranstaltungsbündel zugeordnet werden. Im Zuteilungsvektor x gilt dann $0 \leq x_{ib} \leq 1$, wobei x_{ib} nun die Wahrscheinlichkeit beschreibt, dass dem Studierenden i das Veranstaltungsbündel b zugeteilt wird.

Eine zulässige *zufällige* kombinatorische Zuteilung kann als Wahrscheinlichkeitsverteilung über *deterministische* kombinatorische Zuteilungen dargestellt werden [21]. Im Wesentlichen stützt sich Nguyen, Peivandi und Vohra auf eine generalisierte Form des Satzes von Birkhoff und von Neumann.

Der **Satz von Birkhoff und von Neumann** besagt, dass eine $n \times n$-Matrix genau dann bistochastisch ist, wenn sie eine Konvexkombination von Permutationsmatrizen ist. Konvexkombinationen sind jene Linearkombinationen, dessen Koeffizienten aus dem Einheitsintervall $[0,1]$ stammen und die Summe der Koeffizienten eins beträgt. Die Permutationsmatrizen sind Matrizen, bei der in jeder Zeile und in jeder Spalte genau ein Eintrag eins ist und alle anderen Einträge null betragen. Weiterhin sind Permutationsmatrizen die Extremalpunkte der Menge der bistochastischen Matrizen. Bei (nicht-kombinatorischen) 1-zu-1-Zuteilungsproblemen, bei dem einem Agenten ein einzelnes Objekt zugeteilt wird, kann eine zufällige Zuteilung als bistochatische Matrix beschrieben werden. Dabei steht p_{ij} für die Wahrscheinlichkeit, dass Agent a_i das Objekt o_j zugeteilt bekommt. Die Konvexkombination von Permutationsmatrizen zu dieser bistochastischen Matrix entspricht einer Wahrscheinlichkeitsverteilung über zulässige deterministische Zuteilungen (**Lotterie**) [4, 35].

2.1.6 Beispiel *Im Folgenden betrachten wir ein Beispiel zur Zerlegung einer zufälligen nichtkombinatorischen Zuteilung mithilfe des Satzes von Birkhoff und von Neumann. Während die bistochastische Matrix p die Wahrscheinlichkeiten beschreibt, dass ein Agent a_i ein Objekt o_j erhält, zeigt die Konvexkombination die Wahrscheinlichkeiten über mögliche deterministische Zuteilungen auf. Es kann leicht überprüft werden, dass das Aufaddieren der Linearkombination wieder die bistochastische Matrix ergibt. Weiterhin ist zu beachten, dass sowohl die zufällige Zuteilung als auch die deterministischen Zuteilungen alle zulässig sind.*

$$\mathbf{p} = \begin{array}{c} \\ o_1 \\ o_2 \\ o_3 \end{array} \begin{array}{c} \begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \\ \begin{pmatrix} 0.3 & 0.4 & 0.3 \\ 0.5 & 0 & 0.5 \\ 0.2 & 0.6 & 0.2 \end{pmatrix} \end{array} = 0.3 \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} + 0.2 \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} + 0.3 \cdot \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} + 0.2 \cdot \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \tag{1}$$

Nun sollen diejenigen deterministischen Zuteilungen gefunden werden, die Angebot und Nachfrage erfüllen. Weiterhin muss das Theorem so angepasst werden, dass es auf kombinatorische Zuteilungen anwendbar ist. Glücklicherweise haben Nguyen, Peivandi und Vohra eine Möglichkeit gefunden, das Birkhoff-von-Neumann-Theorem zu generalisieren. Jede zufällige kombinatorische Zuteilung, die Angebot und Nachfrage erfüllt, kann in eine Wahrscheinlichkeitsverteilung von deterministischen kombinatorischen Zuteilungen, welche die Restriktion bezüglich des Angebots um maximal $k-1$ Plätze pro Kurs verletzt, zerlegt werden. Diese Überallokation ist nur von der maximalen Größe k eines Veranstaltungsbündels abhängig, jedoch nicht von der Anzahl von Studierenden bzw. von angebotenen Kursen.

2.1.7 Theorem (nach Theorem 2.1 aus [21]) *Jede (fraktionale) Lösung von Zuteilungswsk., Angebot und Nachfrage kann als Lotterie über (ganzzahlige) Lösungen, die Zugeteilt, Angebot + $k-1$ und Nachfrage erfüllen, implementiert werden.*

$$\sum_{i \in S,\ b \in B} x_{ib}\ b_j \leq q_j + k - 1 \qquad j \in C \qquad \text{(Angebot + } k-1\text{)}$$

$$0 \leq x_{ib} \leq 1 \qquad \forall i \in S, b \in B \qquad \text{(Zuteilungswsk.)}$$

Zunächst wird das Birkhoff-von-Neumann-Theorem für die Zerlegung einer zufälligen nichtkombinatorischen Zuteilung in eine Wahrscheinlichkeitsverteilung von deterministischen Zuteilungen betrachtet. Wenn die zufällige Zuteilung zulässig ist, lassen sich ebenfalls zulässige deterministische Zuteilungen finden. Da die fraktionale Lösung x insbesondere als Konvexkombination integraler Lösungen dargestellt werden kann, bedeutet das, dass x von allen integralen Lösungen „umschlossen“ ist. Nguyen, Peivandi und Vohra zeigen im Beweis zu Theorem 2.1 aus [21], dass es keine Hyperebene gibt, die x von der Menge der integralen Lösungen trennt.

Nun kann das Birkhoff-von-Neumann-Theorem in einer anderen jedoch äquivalenten Weise betrachtet werden: Die fraktionale Lösung x, welche eine $n \times n$-Matrix ist, sei nun ein Vektor $x \in \mathbb{R}^{n^2}, 0 \leq x_{ib} \leq 1$ und analog integrale Lösungen $\bar{x} \in \{0,1\}^{n^2}$. x entspricht nun dem Vektor der Zuteilungswahrscheinlichkeiten zwischen Studierenden und Veranstaltungsbündel und $\bar{x}$ entspricht dem Vektor einer deterministischen Zuteilung.

Dadurch, dass x als Konvexkombination integraler Lösungen dargestellt wird, also x von integralen Lösungen „umschlossen“ ist, kann in allen Richtungen mit einem beliebigen Kostenvektor

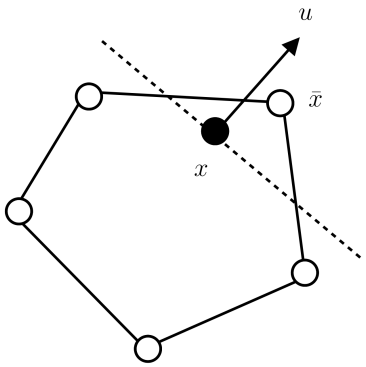


Abbildung 1: x kann als Konvexkombination integraler Lösungen dargestellt werden. Abbildung aus [21].

u von x aus eine integrale Lösung $\bar{x}$ gesucht und gefunden werden. Abbildung 1 zeigt beispielhaft eine fraktionale Lösung x und ihre Zerlegung in eine Konvexkombination. Hier kann man sehen, dass bei jedem Kostenvektor u eine integrale Lösung gefunden werden kann. Es gibt laut Nguyen, Peivandi und Vohra für jede gegebene fraktionale Lösung x, die Angebot und Nachfrage erfüllt, bei einem beliebigen Kostenvektor u eine ganzzahlige Lösung $\bar{x}$, sodass gilt $u \cdot x \leq u \cdot \bar{x}$.

Dieses Theorem und insbesondere die alternative Betrachtung dessen gilt jedoch nur für 1-zu-1-Zuordnungen von Agenten und Objekten. Für die Zuteilung von Bündel, also $k > 1$, muss aber die Restriktion des Angebots relaxiert werden. Die wesentliche Idee ist, dass man eine Extrempunktlösung x^* findet, die $u \cdot x$ maximiert. x^* kann dann auf eine ganzzahlige Lösung $\bar{x}$ gerundet werden, die wiederum die Restriktionen der Nachfrage und des Angebots $+ k - 1$ erfüllt. Das heißt, dass ein Kurs im Veranstaltungsbündel maximal um eins weniger als die Größe eines Bündels k überbelegt werden kann. Nguyen, Peivandi und Vohra zeigen zudem, dass jede fraktionale Lösung x in der konvexen Hülle der Menge von ganzzahligen Lösungen ist. Somit kann x als Konvexkombination von denjenigen ganzzahligen Lösungen dargestellt werden, die eine geringe euklidische Distanz zu x haben [21].

2.2 Eigenschaften

Wünschenswerte Eigenschaften einer „guten und fairen“ Zuteilung sind im Allgemeinen Effizienz, Neidfreiheit sowie Schutz gegen strategisches Handeln. Jeder Mechanismus erzeugt entsprechend ihrer Prozedur unterschiedliche, aber nicht zwingend einzigartige, Zuteilungen. Die Mechanismen können anhand dieser Eigenschaften gut charakterisiert und verglichen werden. Im Folgenden sollen Effizienz, Neidfreiheit und Schutz gegen strategisches Handlen definiert werden.

Stochastische Dominanz In dieser Arbeit betrachten wir Mechanismen, die *zufällige* kombinatorische Zuteilungen erzeugen. Somit muss vorher das Kernkonzept der stochatischen Dominanz zur Definition der Eigenschaften eingeführt werden. Dies ermöglicht die Vergleichbarkeit von zufälligen Zuteilungen.

Sei Δ nun die Menge aller möglichen zufälligen Zuteilungen. Ein **zufälliger Zuteilungsmechanismus** $g : P^{|S|} \mapsto \Delta$ bildet die Menge der Präferenzprofile auf die Menge aller möglichen zufälligen Zuteilungen ab. p_i beschreibt dabei die Zuteilungen des Studierenden i. Es gelte $0 \leq p_{ib} \leq 1$, wobei p_{ib} für die Wahrscheinlichkeit steht, dass Studierender i das Veranstaltungsbündel b erhält.

Gegeben zwei zufälliger Zuteilungen $p_i, q_i \in \Delta$ gilt ein stochatisch dominantes Präferieren von p_i gegenüber q_i, wenn für alle Veranstaltungsbündel b die Wahrscheinlichkeit, dass p_i dem Studierenden ein Veranstaltungsbündel, welches genauso gut oder besser als b ist, zuteilt, mindestens so groß ist wie bei q_i.

2.2.1 Definition *Ein* ***stochastisch dominantes Präferieren*** $p_i \succeq_i^{SD} q_i$ *des Studierenden* $i \in S$ *von* $p_i \in \Delta$ *gegenüber* $q_i \in \Delta$ *gilt dann, wenn*

$$\sum_{b' \succeq_i b} p_{ib'} \geq \sum_{b' \succeq_i b} q_{ib'}, \qquad \forall b \in B$$

Es ist zu beachten, dass $\succeq^{SD}$ keine vollständige Relation ist, daher kann es zufällige Zuteilungen p, q geben, die nicht miteinander vergleichbar sind.

2.2.1 Effizienz

Eine wichtige Eigenschaft von Zuteilungen ist, dass die Wünsche aller Studierenden bestmöglichst erfüllt werden sollen. Dabei darf es in einer Pareto-optimalen Zuteilung (bzw. einer Zuteilung im Pareto-effizienten Zustand) keinen Studierenden geben, der ein höher präferierten Kurs erhalten kann, ohne dass ein anderer Studierender dadurch schlechter gestellt ist.

2.2.2 Definition *Eine zufällige Zuteilung* $p \in \Delta$ *ist* ***ex-post-effizient****, wenn* p *als Lotterie von Pareto-optimalen deterministischen Zuteilungen dargestellt werden kann.*

Ex-post-Effizienz ist jedoch nur eine schwache Voraussetzung für zufriedenstellende Zuteilungen. In der Nachbetrachtung (ex post) einer zufälligen Zuteilung seien aus der Menge der möglichen deterministischen Zuteilungen, welche für die Lotterie herangezogen werden können, ein wesentlicher Teil Pareto-optimal [14]. Obwohl eine solche Zuteilung nicht mehr so veränderbar ist, dass ein Studierender eine höher präferierte Übung erhalten kann, ohne dass andere Studierende schlechter dastehen, müssen diese Zuteilungen aber nicht zwingend allen Studierenden ihre bestmögliche Priorität zugeordnet haben. Weiterhin haben Abdulkadiroğlu und Sönmez gezeigt, dass es bei einigen zufälligen Zuteilungen mit $n \geq 4$ Agenten eine Zerlegung in Wahrscheinlichkeitsverteilungen mit *ineffizienten* deterministischen Zuteilungen möglich ist [1].

2.2.3 Definition *Eine zufällige Zuteilung* $p \in \Delta$ *ist* ***bezüglich der Ordinalskala zu den Prioritäten effizient (ordinal effizient)****, wenn es keine andere zufällige Zuteilung* q *gibt, die gegenüber* p *stochatisch dominant bevorzugt wird.*

Eine Ordinalskala zu den Prioritäten entspricht einer Rangordnung von Kursen in einer Prioritätenliste. Im Gegensatz zur Kardinalität wird nur ausgesagt, dass ein Kurs gegenüber einem anderen Kurs bevorzugt wird, aber nicht um welchen Betrag. Bogomolnaia und Moulin führen die Effizienz bezüglich der Ordinalskala zu den Prioritäten aus der Betrachtung ex ante, also in Erwartung einer zufälligen Zuteilung, ein [5]. Aus der Menge aller möglicher zufälliger Zuteilungen soll diejenige Zuteilung p ausgewählt werden, die keine andere zufällige Zuteilung q hat, die stochatisch dominant bevorzugt werden würde.

2.2.2 Fairness und Neidfreiheit

Zufällige Zuteilungen gelten als *fair*, wenn für Studierende mit identischen Präferenzen identische Wahrscheinlichkeiten zu Veranstaltungsbündel zugeordnet werden. Es darf nicht der Fall

eintreten, dass Neid entsteht. Das geschieht dann, wenn ein Studierender eine höhere Wahrscheinlichkeit zur Teilnahme an einer Übung im Gegensatz zu anderen hat, obwohl er gleiche oder niedrigere Prioritäten zu dieser Übung abgegeben hat. Im Wesentlichen muss der Zuteilungsmechanismus das Prinzip „Equal treatment of equals“ (dt. *Gleichbehandlung von Gleichen*) erfüllen.

2.2.4 Definition *Eine zufällige Zuteilung $p \in \Delta$ ist **(streng) stochastisch dominant neidfrei**, wenn alle Studierenden ihre Zuteilungen gegenüber den Zuteilungen der jeweils anderen Studierenden stochastisch dominant bevorzugt.*

$$\forall i, j \in S : p_i \succeq_i^{SD} p_j$$

2.2.5 Definition *Eine zufällige Zuteilung $p \in \Delta$ ist **(schwach) stochastisch dominant neidfrei**, wenn es keinen Studierenden gibt, der eine Zuteilung eines anderen Studierenden strikt gegenüber seiner eigenen Zuteilung stochastisch dominant bevorzugt.*

$$\nexists i, j \in S : p_i \succeq_i^{SD} p_j$$

2.2.3 Schutz gegen strategisches Handeln

Der Zuteilungsmechanismus soll und darf keine Anreize schaffen, dass ein Studierender falsche Präferenzen abgibt, damit er dadurch bessere Ergebnisse und somit seine tatsächliche Präferenz bekommt. Wenn ein Mechanismus nicht gegen strategisches Handeln geschützt ist, kann das zu Ungerechtigkeit gegenüber Studierenden führen, die die Strategie nicht kennen und somit möglicherweise eine schlechtere Zuteilung bekommen. Weiterhin erschwert das die weitere Planung von Übungen anhand von gegebenen Präferenzen. Anhand falscher Präferenzen kann nicht die tatsächliche Nachfrage abgeschätzt werden, sodass genügend Übungen angeboten werden.

2.2.6 Bezeichnung *$\succ_{i \to i'}$ sei ein alternatives Präferenzprofil, bei welchem Studierender i falsche Angaben zu seiner Präferenz $\succ_{i'}$ angegeben hat und alle anderen Studierenden ihre tatsächlichen Präferenzen angegeben haben.*

$$\succ_{i \to i'} = (\succ_1, \ldots, \succ_{i-1}, \succ_{i'}, \succ_{i+1}, \ldots, \succ_n)$$

2.2.7 Definition *Ein zufälliger Zuteilungsmechanismus g ist **(streng) gegen strategisches Handeln geschützt**, wenn für alle Präferenzprofile $\succ$, für alle Studierende $i \in S$ und alle alternativen Profile $\succ_{i \to i'}$ gelte, dass keine alternative Abgabe des Studierenden i bei gleicher Präferenz aller anderen Studierenden stochastisch dominant bevorzugt wird.*

$$\forall \succ, \forall \succ_{i \to i'}, \forall i \in S : g(\succ) \succeq_i^{SD} g(\succ_{i \to i'})$$

2.2.8 Definition *Ein zufälliger Zuteilungsmechanismus g ist **(schwach) gegen strategisches Handeln geschützt**, wenn für alle Studierenden eine Zuteilung basierend auf ihren tatsächlichen Präferenzen gegenüber einer Zuteilung basierend auf falscher Abgaben schwach stochastisch dominant bevorzugt wird.*

$$\forall \succ, \exists i \in S, \nexists \succ_{i \to i'} : g(\succ_{i \to i'}) \succeq_i^{SD} g(\succ)$$

2.2.4 Unmöglichkeitstheoreme und Kompromiss zwischen Eigenschaften

Idealerweise sollte ein Zuteilungsmechanismus möglichst alle Anforderungen erfüllen, jedoch haben Satterthwaite und Gibbard u. a. in ihrem grundlegendem Theorem (Gibbard-Satterthwaite-Theorem) gezeigt, dass Effizienz und Schutz gegen strategisches Handeln in Konflikt zueinander stehen [32, 11]. Nur Mechanismen, die im Wesentlichen eine Diktatur sind, bieten Schutz gegen strategisches Handeln, müssen dafür aber bei der Effizienz einbüßen. Diktatorische Zuteilungsmechanismen haben stets eine ausgezeichnete Person, die über die Zuteilung anhand ihrer Präferenz entscheidet. Andere Mechanismen können bezüglich der Präferenzerhebung manipuliert werden, sodass sie durch nicht-wahrheitsgemäße Prioritäten ihre tatsächlich gewünschten Präferenzen erhalten. Weiterhin schreibt Kesten, dass ein Zuteilungsmechanismus nicht gegen strategisches Handeln geschützt sein kann, wenn es effiziente und neidfreie Zuteilungen erzeugt [16]. Daraus folgt, dass es keinen Mechanismus geben kann, der gegen strategisches Handeln geschützt ist und Pareto-optimale sowie neidfreie Zuteilungen erzeugt, falls eine solche Zuteilung überhaupt existiert. Konkret heißt es, dass je nach Anwendungsfall stets ein Kompromiss zwischen Effizienz, Neidfreiheit und Schutz gegen strategisches Handeln getroffen werden muss. Für die Übungsgruppenzuteilung am Institut für Informatik wurde der Bundled Probabilistic Serial Mechanismus gewählt, der eine ordinal effiziente und neidfreie Zuteilung erzeugt. Bezüglich des strategischen Handelns ist der Mechanismus jedoch nur schwach geschützt.

2.3 Verwandte Arbeiten

Das Feld der nicht-kombinatorischen Zuteilung ist im Allgemeinen gut erforscht. Obwohl es einige verschiedene Mechanismen gibt, die auch in der Praxis eingesetzt werden, um einzelne Objekte an eine Menge von Agenten zu verteilen, lassen sich die meisten Algorithmen nicht auf die Zuteilung von Objektbündel generalisieren bzw. anpassen. Hier soll ein kurzer Überblick gegeben werden, die für die Problemstellung der Zuteilung von Objektbündel und insbesondere der Übungsgruppenzuteilung eingesetzt werden können.

Random Serial Dictatorship (RSD) ist eines der einfachsten Zuteilungsmechanismen. Dabei wird die Reihenfolge der Agenten zufällig mit gleicher Wahrscheinlichkeit bestimmt. Jeder Agent kann nacheinander sein gewünschtes Objekt auswählen, falls es noch verfügbar ist. *Bundled Random Serial Dictatorship (BRSD)* ist die Erweiterung des RSD. Es gelte weiterhin eine zufällige Permutation der Reihenfolge von Studierenden, welche statt einem Objekt das jeweils am meisten präferierte Bündel auswählen, dessen Objekte noch verfügbar sind. Obwohl RSD ein Mechanismus ist, der stark gegen strategisches Handeln geschützt ist und Fairness bietet, sind die erzeugten Zuteilungen nicht ordinal bezüglich den Präferenzen effizient, sondern nur ex-post-effizient [20].

Eine weitere Möglichkeit der Zuteilung von Sitzplätzen zu Kursen ist das *Approximate Competitive Equilibrium from Equal Income (A-CEEI)* [7]. Nachdem die Studierenden ihre vollständigen Präferenzen zu Veranstaltungsbündeln abgegeben haben, wird jedem Studierenden virtuelles Geld gegeben, um mit Veranstaltungsbündel zu handeln. Der Mechanismus berechnet ein angenähertes Gleichgewichtspreis für die Bündel und den Studierenden wird das passende Bündel entsprechend seiner Präferenzen, seines Budgets und des Preises zugewiesen. Obwohl A-CEEI ein Mechanismus ist, der Pareto-optimale Zuteilungen erzeugt, annähernd neidfrei und teilweise gegen strategisches Handeln geschützt ist, liegt die Schwierigkeit in der Berechnung und Berechenbarkeit eines solchen Preisvektors und der Allokation. Es ist nicht garantiert, dass ein solcher Preisvektor und Allokation existieren. Weiterhin sind die zu verwendenden Algorithmen komplex und können möglicherweise nicht auf größere Probleminstanzen angewendet werden [28].

Die Technische Universität Berlin löst das Zuteilungsproblem mithilfe ganzzahliger Optimierung, wobei eine Zielfunktion $\sum X_{s,c,t} \cdot prio(s, c, t)$ für alle Studierende s, Lehrveranstaltungen c und Zeitslots t minimiert werden soll. Jedem Studierenden sollte die kleinstmögliche Prio-

rität zugewiesen werden, wobei eine kleinere Prioritätenzahl für eine höhere Präferenz steht. $X \in \{0,1\}^{|S|}$ sei der Vektor, der beschreibt, ob dem Studierenden s die Übung der Lehrveranstaltung c zum Zeitslot t zugeteilt wird. Es gelten weiterhin die linearen Restriktionen, dass jedem Studierenden eine Übung pro besuchter Lehrveranstaltung zugewiesen werden muss, in einem Zeitslot maximal eine Übung besucht werden kann sowie die maximale Teilnehmendenanzahl einer Übung nicht überschritten werden darf. Dieses Modell wurde später so erweitert, dass zur Zuteilung von Studierenden auf Übungen auch die Zeitslots der Übungen passenden Räume zugeordnet werden. Angaben zur Pareto-Effizienz, Fairness sowie zum Schutz gegen strategischen Handelns sind nicht bekannt [13].

Letztlich sei der *Probabilistic Serial Mechanismus (PS)* von Bogomolnaia und Moulin erwähnt, welcher eine neidfreie und ordinal effiziente Zuteilung bezüglich der abgegebenen Präferenzen erzeugt [5]. Dieser Mechanismus ist nur schwach gegen strategisches Handeln geschützt. *Bundled Probabilistic Serial (BPS)* von Nguyen, Peivandi und Vohra ist eine generalisierte Form dieses Mechanismus und errechnet eine zufällige Zuteilung zu Veranstaltungsbündel statt zu einzelnen Objekten [21]. Der generalisierte Mechanismus behält die gleichen Eigenschaften der Neidfreiheit, ordinalen Effizenz sowie schwachen Schutz gegen strategisches Handeln. Die genaue Funktionsweise des Bundled Probabilistic Serial Mechanismus werde im folgenden Kapitel genauer erläutert.

3 Bundled Probabilistic Serial

Bundled Probabilistic Serial (BPS) als Zuteilungsmechanismus erzeugt zunächst eine zufällige kombinatorische Zuteilung, die ordinal über die Präferenzen von Veranstaltungsbündel effizient ist, Neidfreiheit garantiert und gegen strategisches Handeln schwach geschützt ist. Anschließend wird diese fraktionale Lösung in eine Wahrscheinlichkeitsverteilung von zulässigen deterministischen Zuteilungen zerlegt, von denen eine deterministische Zuteilung in einer Lotterie ausgewählt wird. Dieses Kapitel beschreibt näher die Funktionsweise des Mechanismus und die Implementierung dessen in Python 3.

3.1 Präferenzen über Veranstaltungsbündel

BPS ist ein zufälliger Zuteilungsmechanismus $BPS : \mathcal{P}^{|S|} \mapsto \Delta$, der ein Präferenzprofil $\succ \in \mathcal{P}^{|S|}$ auf eine zufällige kombinatorische Zuteilung $p \in \Delta$ abbildet. Das Präferenzprofil $\succ$ ist ein n-Tupel von Präferenzrelationen der Studierenden über Veranstaltungsbündel. Nehmen wir einen Studierenden des ersten Semesters in der Informatik als Beispiel, würde dieser typischerweise die Lehrveranstaltungen *Informatik 1*, *Mathematik für Informatiker 1* sowie *Diskrete Strukturen und Logik* belegen. Jede dieser Lehrveranstaltungen bieten zahlreiche Übungstermine an: Im Wintersemester 2019/2020 sind das 17 Übungstermine in *Informatik 1*, 10 Übungstermine in *Mathematik für Informatiker 1* und 9 Übungstermine in *Diskrete Strukturen und Logik*. Es ist nicht zumutbar, dass ein Studierender exponentiell viele (hier: 1530) Bündel in eine strikte Präferenzordnung bringt. Weiterhin soll auch beachtet werden, dass diese Veranstaltungsbündel in sich überschneidungsfrei sind.

Es muss eine einfachere Sprache gefunden werden, welche es den Studierenden trotzdem ermöglicht, ihre Präferenzen adäquat ausdrücken zu können. Im bisherigen Vorlesungsverwaltungssystem vom Lehrstuhl des Prof. Dr. Hagerup sowie im Lernmanagementsystem Stud.IP geben Studierende Prioritäten zu Übungen innerhalb einer Lehrveranstaltung ab. Zusätzlich wurde im Verwaltungssystem des Lehrstuhls eine Mindestanzahl sowie gegebenenfalls weitere Anforderungen zu abzugebenen Prioritäten gefordert. Diese Sprache der Präferenzerhebung soll beibehalten werden und aus den Prioritätenlisten zu den verschiedenen Lehrveranstaltungen eines Studierenden soll eine gesamte Präferenzordnung zu Veranstaltungsbündel induziert werden.

Ein Ansatz ist, für jeden Studierenden das kartesische Produkt seiner Prioritätenlisten von verschiedenen Lehrveranstaltungen zu erzeugen, sodass man eine Liste von Tupel, welche den Veranstaltungsbündel entsprechen, erhält. Weiter sollen diejenigen Veranstaltungsbündel entfernt werden, dessen Übungen sich zeitlich überschneiden. Schließlich soll die Liste der Bündel so sortiert werden, dass die Kombination der Prioritäten von den Übungen des Bündels aufsteigend gereiht sind. Jedem Bündel wird durch ein Scoring anhand der Prioritäten einen Wert zugewiesen, dabei gilt ein kleinerer Score als stärker bevorzugt. Betrachten wir beispielsweise drei verschiedene Veranstaltungsbündel der Lehrveranstaltungen *Informatik 1*, *Mathematik für Informatiker 1* und *Diskrete Strukturen und Logik* jeweils mit den Prioritäten $(1, 1, 2)$, $(1, 2, 1)$ und $(2, 1, 1)$ wären diese beim Scoring ohne Beachtung der Kapazitäten gleichwertig. BPS erwartet jedoch eine strikte Präferenzordnung über Veranstaltungsbündel, um eine zufällige kombinatorische Zuteilung zu erstellen.

Um diese Ambivalenzen zu beseitigen, spielen die Kapazitäten der jeweiligen Übungstermine nun eine Rolle. Bei gleichem Score zweier oder mehr Bündel ist es „einfacher“ die Prioritäten zu erfüllen, wenn zunächst diejenigen Übungen mit größerer Kapazität bevorzugt zugeteilt werden. Konkret bedeutet das zum Beispiel, dass für Übungen der *Informatik 1*, *Mathematik für Informatiker 1* und *Diskreten Strukturen und Logik* das Bündel mit den Kapazitäten $(30, 30, 30)$ gegenüber dem Bündel mit Kapazitäten $(24, 30, 30)$ bei gleichem Score ohne Betrachtung der

Kapazitäten in der Präferenzordnung einen höheren Rang erhalten soll.

$$score(b) = \sum_{j=1}^{k} \exp(prio_i(c_j)) + \frac{1}{\sum_{j=1}^{k} q_j} \tag{2}$$

$prio_i$ ist hier eine Hilfsfunktion, die die Priorität von Kurs c_j eines Studierenden i in der entsprechenden Liste zur Lehrveranstaltung ausgibt.

Für den eher unwahrscheinlichen Fall, dass Bündel trotz Scoring mit Kapazitäten einen gleichwertigen Score erhalten, geschieht die Reihung willkürlich und abhängig von der Implementierung.

Algorithm 1 Erzeugen einer Präferenzordnung über überschneidungsfreie Veranstaltungsbündel

```
1:  function GenerateBundles(rankings, overlaps)
2:      bundles ← r_1 × r_2 × ··· × r_k, ∀r_l ∈ rankings
          ▷ rankings ist die Menge an Prioritätenlisten verschiedener Lehrveranstaltungen eines
        Studierenden
3:      for all bundle = (c_1, ..., c_k) in bundles do      ▷ bundle ist ein k-Tupel aus Übungen
4:          for j ← 1, k do
5:              if overlaps[c_j] contains any in (c_1, ..., c_{j-1}, c_{j+1}, ..., c_k) then
                                          ▷ Entfernen eines Bündels, falls sich Übungen überschneiden
6:                  bundles ← bundles\bundle
7:                  break
8:              end if
9:          end for
10:     end for
11:     ≻_i ← sort(bundles)                    ▷ Sortieren anhand von Wert berechnet in (2)
12:     return ≻_i
13: end function
```

Montag, 08:15 Uhr (Kapazität 54)	1
Mittwoch, 10:00 Uhr (Kapazität 30)	2
Dienstag, 08:15 Uhr (Kapazität 54)	3
...	...
Freitag, 14:00 Uhr (Kapazität 30)	17

Dienstag, 08:15 Uhr (Kapazität 30)	1
Montag, 08:15 Uhr (Kapazität 24)	2
Dienstag, 15:45 Uhr (Kapazität 30)	3
...	...
Donnerstag, 12:15 Uhr (Kapazität 36)	10

Donnerstag, 12:15 Uhr (Kapazität 24)	1
Mittwoch, 10:00 Uhr (Kapazität 36)	2
Freitag, 10:00 Uhr (Kapazität 24)	3
...	...
Montag, 17:30 Uhr (Kapazität 120)	9

Tabelle 1: Prioritätenlisten eines Studierenden zu *Informatik 1*, *Mathematik für Informatiker 1* und *Diskrete Strukturen und Logik*

3.1.1 Beispiel *Für jeden Studierenden werde die Funktion* GenerateBundles *(Algorithmus 1) ausgeführt. Gegeben dreier Prioritätenlisten (Tabelle 1) soll nun eine Präferenzordnung über Veranstaltungsbündel erzeugt werden.*

Nun werde das kartesische Produkt über die drei Prioritätenlisten gebildet und es entstehen in diesem Fall 1530 Veranstaltungsbündel über Informatik 1, Mathematik für Informatiker 1 und Diskrete Strukturen und Logik. Einige Veranstaltungsbündel überschneiden sich und müssen aus der Liste entfernt werden. Übungstermine am Institut für Informatik der Universität Augsburg

Veranstaltungsbündel						**Score**
	Informatik 1		*Mathematik f. Informatiker 1*		*Diskrete Strukturen u. Logik*	
1	Montag, 08:15 Uhr	1	Dienstag, 08:15 Uhr	1	Donnerstag, 12:15 Uhr	8.1641
1	Montag, 08:15 Uhr	1	Dienstag, 08:15 Uhr	2	Mittwoch, 10:00 Uhr	12.8339
~~1~~	~~Montag, 08:15 Uhr~~	~~2~~	~~Montag, 08:15 Uhr~~	~~1~~	~~Donnerstag, 12:15 Uhr~~	~~12.8348~~
2	Mittwoch, 10:00 Uhr	1	Dienstag, 08:15 Uhr	1	Donnerstag, 12:15 Uhr	12.8375
~~1~~	~~Montag, 08:15 Uhr~~	~~2~~	~~Montag, 08:15 Uhr~~	~~2~~	~~Mittwoch, 10:00 Uhr~~	~~17.5052~~
~~2~~	~~Mittwoch, 10:00 Uhr~~	~~1~~	~~Dienstag, 08:15 Uhr~~	~~2~~	~~Mittwoch, 10:00 Uhr~~	~~17.5068~~
2	Mittwoch, 10:00 Uhr	2	Montag, 08:15 Uhr	1	Donnerstag, 12:15 Uhr	17.5092
1	Montag, 08:15 Uhr	1	Dienstag, 08:15 Uhr	3	Donnerstag, 12:15 Uhr	25.5304
			...			
17	Freitag, 14:00 Uhr	10	Donnerstag, 12:15 Uhr	9	Montag, 17:30 Uhr	24185082.31

Tabelle 2: Anhand der Prioritätenlisten erzeugte Präferenzreihung über Veranstaltungsbündel eines Studierenden. Die sich überschneidenden Bündel sind hier durchgestrichen.

sind in der Regel 90 Minuten lang. Welche Termine sich überschneiden, wurde vorher bereits errechnet und in der Hashtabelle `overlaps` *gespeichert. So ist beispielsweise ein Veranstaltungsbündel mit der Übung zu Informatik 1 montags um 08:15 Uhr (Priorität 1) und der Übung zu Mathematik für Informatiker 1 ebenfalls montags um 08:15 (Priorität 2) nicht möglich. Sich überschneidende Bündel wurden in Tabelle 2 durchgestrichen.*

Die Reihung der Bündel ergibt sich durch den Score: Mit jeder schlechteren Priorität pro Lehrveranstaltung steigt der Score exponentiell an. Gleichwertige Bündel mit den Prioritäten $(1, 1, 2)$*,* $(1, 2, 1)$ *und* $(2, 1, 1)$ *haben jeweils die Kapazitäten* $(54, 30, 36)$*,* $(54, 24, 24)$ *und* $(30, 30, 24)$*. Es ist das Veranstaltungsbündel zu bevorzugen, welches insgesamt eine größere Kapazität anbieten kann. Entsprechend bekommt das Bündel mit Priorität* $(1, 1, 2)$*, also das Bündel mit den Übungen am Montag, um 08:15 Uhr für Informatik 1, am Dienstag, 08:15 Uhr für Mathematik für Informatiker 1 sowie am Mittwoch, 10:00 Uhr für Diskrete Strukturen und Logik, Vorrang gegenüber den anderen Bündel.*

Nachdem für jeden Studierenden nun eine Präferenzordnung über seine Veranstaltungsbündel erzeugt wurde, kann eine zufällige kombinatorische Zuteilung erzeugt werden. Nun soll jedem Studierenden Wahrscheinlichkeiten zugeordnet werden, dass dieser möglichst seine am stärksten favorisierten Bündel erhält.

3.2 Erzeugen zufälliger kombinatorischer Zuteilungen

Wir betrachten zunächst die Funktionsweise des ursprünglichen Probabilistic Serial Mechanismus von Bogomolnaia und Moulin, welche eine zufällige nicht-kombinatorische Zuteilung erzeugt. Eine zufällige Zuteilung ordnet jedem Agenten eine Wahrscheinlichkeit zu, dass dieser ein bestimmtes Objekt zugeteilt bekommt. Bogomolnaia und Moulin beschreiben in ihrem Papier den Mechanismus intuitiv als das Konsumieren („Essen“) eines Objektes [5]. Der Teil des eigentlich unteilbaren Objekts, der von einem Agenten konsumiert wurde, werde als Wahrscheinlichkeit, dass dieses Objekt dem Agenten als Ganzes zugeteilt wird, interpretiert. Der Zeitraum, in dem Agenten ihre gewünschten Objekte konsumieren, sei im Intervall $[0, 1]$. Alle Agenten fangen zum gleichen Zeitpunkt $t = 0$ an, mit gleicher Geschwindigkeit ihr gewünschtes Objekt zu konsumieren. Ein Agent könnte ohne Konkurrenz ein Objekt als Ganzes konsumieren, das heißt beim Zeitpunkt $t = 1$ wäre die Wahrscheinlichkeit gleich eins, dass dieser Agent das Objekt erhält. Wenn ein Objekt aufgebraucht ist, wechseln die Agenten entsprechend zum nächsten noch ver-

fügbaren Objekt aus ihrer Präferenzordnung. Zum Zeitpunkt $t = 1$ erhält somit jeder Agent seine Wahrscheinlichkeiten der Zuteilung.

3.2.1 Beispiel *Angenommen, es gibt vier Agenten* $A = \{1, 2, 3, 4\}$ *und vier Objekte* $O = \{a, b, c, d\}$. *Jeweils ein Objekt als Ganzes wird einem Agenten zugeteilt. Weiterhin seien die Präferenzordnungen der Agenten folgendermaßen: Agenten 1 und 2 wünschen sich das Objekt* a *am meisten, gefolgt von* b *und* c *sowie abschließend* d. *Hingegen bevorzugen die Agenten 3 und 4 das Objekt* b, *danach* a, c *und schließlich* d.

$$a \succ_{1,2} b \succ_{1,2} c \succ_{1,2} d$$

$$b \succ_{3,4} a \succ_{3,4} c \succ_{3,4} d$$

Agenten 1 *und* 2 *konsumieren zunächst das Objekt* a, *während die Agenten* 3 *und* 4 *mit Objekt* b *starten. Zum Zeitpunkt* $t = 0.5$ *sind die Objekte* a *und* b *jeweils aufgebraucht, denn ein Objekt wird von zwei Agenten beansprucht, wodurch jeder Agent die Hälfte des Objekts erhält. Somit wechseln die Agenten zum nächsten noch verfügbaren Objekt entsprechend ihrer Präferenzen. Im Zeitintervall zwischen* $t = 0.5$ *und* $t = 0.75$ *sowie* $t = 0.75$ *und* $t = 1$ *werden die Objekte* c *beziehungsweise* d *konsumiert. Alle vier Agenten haben die gleiche Präferenzreihung zu den Objekten* c *und* d, *weshalb die Objekte gleichmäßig auf die Agenten verteilt werden.*

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1/2 & 0 & 1/4 & 1/4 \\ 1/2 & 0 & 1/4 & 1/4 \\ 0 & 1/2 & 1/4 & 1/4 \\ 0 & 1/2 & 1/4 & 1/4 \end{pmatrix} \end{array}$$

Somit haben wir eine zufällige Zuteilung erhalten, die laut Bogomolnaia und Moulin ordinal effizient und fair ist.

Nun kann man sich fragen, warum bei der erzeugten zufälligen Zuteilung beispielsweise die Wahrscheinlichkeiten der Zuteilung des Objektes b an die Agenten 1 bzw. 2 gleich null sind. Sollten diese Agenten nicht wenigstens eine geringe Chance darauf haben, das Objekt zugeteilt zu bekommen? Im Wesentlichen müssen hier die ordinale Effizienz und die Fairness erfüllt werden. Zunächst gilt das Prinzip „Equal treatment of equals“. Es darf nicht möglich sein, dass mehrere Agenten mit identischen Präferenzen unterschiedliche Zuteilungswahrscheinlichkeiten erhalten. Daher gelten für die Agenten 1 und 2 sowie 3 und 4 jeweils die gleichen Wahrscheinlichkeiten. Weiterhin sei $a \succ_{1,2} b$ sowie $b \succ_{3,4} a$: Für Agenten 1 und 2 steht das Objekt b in der Präferenzordnung nur an zweiter Stelle, während für Agenten 3 und 4 Objekt b die stärkste Priorität hat. Damit Agent 1 (bzw. Agent 2) einen Teil von Objekt b abbekommt, hätte seine Präferenzordnung anders gewählt sein müssen. Das würde bedeuten, dass Agent 1 (bzw. Agent 2) das Objekt b mindestens gleichwertig zu den Agenten 3 und 4 bevorzugt. Jedoch hat Agent 1 (bzw. Agent 2) eine strikte Präferenz von Objekt a gegenüber Objekt b ausgedrückt und erhält entsprechend die Zuteilungswahrscheinlichkeiten, sodass die ordinale Effizienz erfüllt ist. Der Beweis zur Erfüllung dieser Eigenschaften kann dem Papier von Bogomolnaia und Moulin [5] entnommen werden.

Die Intuition der Erweiterung des Probabilistic Serial Mechanismus für die Zuteilung von Objektbündel ist, dass die Agenten alle Objekte eines Bündels gleichzeitig konsumieren. Sobald das Angebot eines einzelnen Objektes aus einem Bündel erschöpft ist, sollen die Agenten die Objekte des nächsten Bündels aus der Präferenzordnung konsumieren, dessen Objekte noch

Kapazität verfügbar haben. Die Generalisierung Bundled Probabilistic Serial erzeugt nun Wahrscheinlichkeitsverteilungen zu Objektbündel, die ein Agent haben kann. Im Folgenden werde der BPS-Mechanismus für die Übungsgruppenzuteilung formell beschrieben:

3.2.2 Bezeichnung $t(0) = 0 \leq t(1) \leq t(2) \leq \cdots \leq t(v) \leq t(v+1) \leq \ldots$ *seien die Zeitpunkte (Schritte) im Intervall* $[0, 1]$*, wann mindestens ein Kurs keine Sitzplätze zum Zuteilen übrig hat.*

3.2.3 Bezeichnung $BPS(v)$ *sei die zufällige Zuteilung zum Zeitpunkt* $t(v)$*. Initial sei* $BPS(0) = 0$*, wobei kein Studierender ein Kurs zugeteilt bekommt.*

3.2.4 Bezeichnung $G(v)$ *sei die Menge an Kursen, die zum Zeitpunkt* $t(v)$ *mindestens einen Sitzplatz anbieten können. Am Anfang stehen alle Sitzplätze der Kurse zur Verfügung, es gelte* $G(0) = C$*.*

3.2.5 Bezeichnung z^v *sei eine deterministische Zuteilung, die nicht notwendigerweise zulässig ist und allen Studierenden ihr am meisten präferiertes Veranstaltungsbündel bestehend aus Kursen in* $G(v)$ *zuteilt. Diese beschreibt die angeforderten Bündel der Studierenden zum Zeitpunkt* $t(v)$*.*

3.2.6 Bezeichnung $m_j(v)$ *beschreibt die Anzahl der zugeteilten Sitzplätze eines Kurses* j *in der Zuteilung* z^v*.*

3.2.7 Bezeichnung r_j^v *gibt das Maß an, wie viel vom Kurs* j *zum Zeitpunkt* $t(v)$ *bereits konsumiert wurde. Anfangs wurde noch nichts konsumiert, also gelte* $r_j^0 = 0$*.*

Der Mechanismus besteht aus folgenden Schritten:

- Die durch BPS erzeugte zufällige Zuteilung sei zum Zeitpunkt $t(v)$:

$$BPS(v) = BPS(v-1) + (t(v) - t(v-1)) \cdot z^{v-1}$$

  $(t(v) - t(v-1))$ repräsentiert den Teil des Bündels in z^{v-1}, welchen jeder Studierende im Zeitintervall $(t(v-1), t(v)]$ konsumiert hat.

- Es wird die nächste kleinste Instanz $t(v)$ gesucht, zu der ein beliebiger Kurs keine Sitzplätze mehr zur Verfügung hat. Gegeben der verfügbaren Kurse $G(v-1)$ ist $t_j(v)$ der spätestmögliche Zeitpunkt, wann ein Kurs j vollbesetzt wäre. In anderen Worten ist $t_j(v)$ der maximal noch zuteilbare Teil pro Anzahl zuzuteilender Sitzplätze $m_j(v-1)$ des Kurses j in der Anforderung z^v.

$$t_j(v) = sup\{t \in [0,1] \mid r_j^{v-1} + m_j(v-1) \cdot (t - t(v-1)) \leq q_j\}$$

$$t(v) = min\{t_j(v) \mid j \in G(v-1)\}$$

- Der zum Zeitpunkt $t(v)$ aufgebrauchte Kurs wird aus der Menge der noch verfügbaren Kurse entfernt:

$$G(v) = G(v-1) \backslash \{j \in G(v-1) \mid t_j(v) = t(v)\}$$

  Sowie das konsumierte Maß des Kurses j zum Zeitpunkt $t(v)$ werde um den im letzten Schritt konsumierten Teil aktualisiert:

$$r_j^v = r_j^{v-1} + m_j(v-1) \cdot (t(v) - t(v-1))$$

  Aus den (noch) verfügbaren Kursen in $G(v)$ wird eine deterministische Zuteilung z^v erzeugt, die jedem Studierenden ihr am meisten präferiertes Bündel zuteilt.

- BPS terminiert dann, wenn $t(v) = 1$, wobei v der kleinste Index ist, dass $t(v) = 1$ gilt.

Algorithm 2 Erzeugen einer zufälligen Zuteilung mit BPS

function GenerateFracAlloc($(\succ_i)_{i \in S}$)
  $t \leftarrow 0.0$
  $x_{ib} \leftarrow 0.0, \forall i \in S, b \in B$ ▷ Entspricht $BPS(0)$
  while $t < 1.0$ **do**
    $dem_j \leftarrow |\{i \in S \mid b = first(\succ_i), j \in b\}|$
      ▷ Auswahl von $first(\succ_i), \forall i \in S$ entspricht z^v
    $\Delta = min\left\{\frac{q_j}{dem_j} \mid j \in C\right\}$
    $\Delta^* = \Delta - (t - min\{1, t\})$ ▷ $x_{ib} \not> 1$
    $t \leftarrow t + \Delta$ ▷ Entspricht $(t(v) - t(v-1))$
    for all $i \in S$ **do**
      $x_{ib} \leftarrow x_{ib} + \Delta^*, b = first(\succ_i)$
    end for
    for all $j \in C$ **do**
      $q_j \leftarrow q_j - \Delta^* \cdot dem_j$ ▷ Aktualisieren des konsumierten Maßes
      if $q_j = 0$ **then**
        Entferne jene Bündel in $\succ_i$, die j enthalten.
          ▷ Entspricht dem Entfernen des Kurses aus $G(v-1)$
      end if
    end for
  end while
  return x_{ib}
end function

3.2.8 Theorem (nach Theorem 4.1 aus [21]) *Der Bundled Probabilistic Serial (BPS) Mechanismus ist ordinal effizient, neidfrei und schwach gegen strategisches Handeln geschützt. Bei einer Präferenzordnung über Objektbündel mit einer maximalen Größe k können die Bündel so zugeteilt werden, dass von jedem Objekt maximal $k - 1$ Einheiten zu viel zugeteilt wird.*

Eine Implementierung des BPS-Mechanismus kann dem Algorithmus 2 entnommen werden. Der generalisierte Mechanismus arbeitet im Wesentlichen nach der gleichen Greedy-Strategie. Der Beweis zum Theorem folgt ähnlich zum Probabilistic Serial von Bogomolnaia und Moulin und kann genauer aus dem Appendix C des Papiers von Nguyen, Peivandi und Vohra entnommen werden [5, 21]. Hier werde der Beweis nur skizziert.

BPS ist ordinal effizient, denn wenn es eine andere stochastisch dominante Zuteilung gäbe, hätte diese Zuteilung mindestens eine gleichwertige Präferenz haben müssen, sodass BPS dieser Zuteilung eine höhere Wahrscheinlichkeit zuordnet, oder diese Zuteilung ist nicht zulässig und verletzt die Restriktionen von Angebot und Nachfrage. Weiterhin gelte ein schwacher Schutz vor strategischem Handeln, da eine stochastisch dominante Zuteilung aufgrund von falscher Prioritäten wegen der Greedy-Strategie unzulässig sein muss. Jeder Studierende kriegt zu einem beliebigen Zeitpunkt t stets jenes Bündel, welches zu dem Zeitpunkt am meisten bevorzugt wird. Neidfreiheit ist auch gegeben, denn es ist nicht möglich, dass einem Studierenden eine höhere Wahrscheinlichkeit zu einem Bündel zugeordnet wird, als ihm zusteht. BPS ordnet Teile des Bündels nur Studierenden zu, wenn alle Kurse des Bündels noch Kapazitäten verfügbar haben. Wenn einem Studierenden eine höhere Wahrscheinlichkeit zugeordnet wurde, dann muss dies

nach Erschöpfung des Angebots geschehen sein. Somit wäre eine solche Zuteilung ebenfalls unzulässig. Schließlich folgt eine Überallokation von maximal $k-1$ Einheiten aus dem Theorem 2.1.7.

3.2.9 Beispiel *Sei angenommen, dass es vier Studierende $S = \{s_1, s_2, s_3, s_4\}$ und zwei Lehrveranstaltungen A und B mit jeweils zwei Übungstermine $C = \{a_1, a_2, b_1, b_2\}$ gibt. Jeder Übungstermin hat eine Kapazität für zwei Studierende. Die Präferenzordnung $\succ$ sei wie folgt:*

$$
\begin{aligned}
s_1 &: (a_1, b_1) \succ (a_1, b_2) \succ (a_2, b_1) \succ (a_2, b_2) \\
s_2 &: (a_1, b_1) \succ (a_2, b_1) \succ (a_1, b_2) \succ (a_2, b_2) \\
s_3 &: (a_1, b_2) \succ (a_2, b_2) \succ (a_2, b_1) \succ (a_1, b_1) \\
s_4 &: (a_1, b_2) \succ (a_2, b_1) \succ (a_2, b_2) \succ (a_1, b_1)
\end{aligned}
$$

Iteration $v = 0$***:*** *Initialisierung, anfangs wurden noch keine Zuteilungen gemacht und alle Kurse haben ihre vollständige Kapazität zur Verfügung.*

$$
q^0 = \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & b_1 & b_2 \end{array} \\ \begin{pmatrix} 2 & 2 & 2 & 2 \end{pmatrix} \end{array}
$$

$$
\begin{aligned}
t(0) &= 0 \\
BPS(0) &= 0 \\
G(0) &= C
\end{aligned}
$$

Jeder Studierende kann sein am meisten präferiertes Bündel anfordern. Dies sei nun die deterministische Zuteilung z^0.

$$
z^0 = \begin{array}{c} \begin{array}{c} s_1, (a_1, b_1) \\ s_2, (a_1, b_1) \\ s_3, (a_1, b_2) \\ s_4, (a_1, b_2) \end{array} \\ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \end{array}
$$

Iteration $v = 1$***:*** *Es wird nun der kleinste zuteilbare Teil eines Bündels gesucht, sodass die Kapazität eines beliebigen Kurses ausgelastet ist. In anderen Worten: Für jeden Kurs wird der spätestmöglichste Zeitpunkt bestimmt, wann dieser Kurs vollbesetzt wäre. Die kleinste Instanz dessen ist $t(1)$.*

$$
t_{a_1}(1) = \frac{2}{4} = \frac{1}{2} \qquad t_{b_1}(1) = \frac{2}{2} = 1 \qquad t_{b_2}(1) = \frac{2}{2} = 1
$$

$$
\Rightarrow t(1) = \frac{1}{2}
$$

Das konsumierte Maß aller Kurse, die in der deterministischen Zuteilung z^0 enthalten sind,

wird nun aktualisiert und von der Kapazität abgezogen.

$$\begin{aligned} r^1_{a_1} &= r^0_{a_1} + m_{a_1}(0) \cdot (t(1) - t(0)) \\ &= 0 + 4 \cdot (0.5 - 0) = 2 \\ r^1_{b_1} &= 0 + 2 \cdot (0.5 - 0) = 1 \\ r^1_{b_2} &= 0 + 2 \cdot (0.5 - 0) = 1 \end{aligned}$$

$$q^1 = \begin{array}{c} a_1 \quad a_2 \quad b_1 \quad b_2 \\ (\,0 \quad\; 2 \quad\; 1 \quad\; 1\,) \end{array}$$

Der Kurs a_1 ist nun aufgebraucht und wird aus der Menge der noch verfügbaren Kurse entfernt:

$$G(1) = \{a_2, b_1, b_2\}$$

Es werden jene Bündel aus dem Präferenzprofil $\succ$ entfernt, die Kurs a_1 enthalten. Anhand der Menge der noch verfügbaren Kurse wird wieder eine deterministische Zuteilung z^1 erstellt, die die Anforderungen der Studierenden beschreibt:

$$\begin{aligned} s_1 &: (a_2, b_1) \succ (a_2, b_2) \\ s_2 &: (a_2, b_1) \succ (a_2, b_2) \\ s_3 &: (a_2, b_2) \succ (a_2, b_1) \\ s_4 &: (a_2, b_1) \succ (a_2, b_2) \end{aligned}$$

$$z^1 = \begin{array}{l} s_1, (a_2, b_1) \\ s_2, (a_2, b_1) \\ s_3, (a_2, b_2) \\ s_4, (a_2, b_1) \end{array} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Die resultierende zufällige Zuteilung ist:

$$\begin{aligned} BPS(1) &= BPS(0) + (t(1) - t(0)) \cdot z^0 \\ &= 0 + (0.5 - 0) \cdot \begin{array}{l} s_1, (a_1, b_1) \\ s_2, (a_1, b_1) \\ s_3, (a_1, b_2) \\ s_4, (a_1, b_2) \end{array} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\ &= \begin{array}{l} s_1, (a_1, b_1) \\ s_2, (a_1, b_1) \\ s_3, (a_1, b_2) \\ s_4, (a_1, b_2) \end{array} \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} \end{aligned}$$

Iteration $v = 2$***:*** *Bestimmen des kleinsten zuteilbaren Teiles:*

$$t_{a_2}(2) = \frac{2}{4} \qquad t_{b_1}(2) = \frac{1}{3} \qquad t_{b_2}(2) = \frac{1}{1}$$

$$\Rightarrow t(2) = t(1) + \frac{1}{3}$$

Aktualisieren des konsumierten Maßes bzw. der Kapazitäten:

$$q^2 = \begin{matrix} a_1 & a_2 & b_1 & b_2 \\ (0 & 2/3 & 0 & 2/3) \end{matrix}$$

Kurs b_1 ist nun aufgebraucht und wird aus der Menge noch verfügbarer Kurse entfernt.

$$G(1) = \{a_2, b_2\}$$

Entsprechend wird das Präferenzprofil $\succ$ angepasst.

$$\begin{aligned} s_1 &: (a_2, b_2) \\ s_2 &: (a_2, b_2) \\ s_3 &: (a_2, b_2) \\ s_4 &: (a_2, b_2) \end{aligned}$$

Anhand des Präferenzprofils $\succ$ wird nun eine neue deterministische Zuteilung erzeugt:

$$z^2 = \begin{matrix} s_1, (a_2, b_2) \\ s_2, (a_2, b_2) \\ s_3, (a_2, b_2) \\ s_4, (a_2, b_2) \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Die zufällige Zuteilung im zweiten Iterationsschritt ist:

$$\begin{aligned} BPS(2) &= BPS(1) + (t(2) - t(1)) \cdot z^1 \\ &= \begin{matrix} s_1, (a_1, b_1) \\ s_2, (a_1, b_1) \\ s_3, (a_1, b_2) \\ s_4, (a_1, b_2) \end{matrix} \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix} + \frac{1}{3} \cdot \begin{matrix} s_1, (a_2, b_1) \\ s_2, (a_2, b_1) \\ s_3, (a_2, b_2) \\ s_4, (a_2, b_1) \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \end{aligned}$$

$$= \begin{matrix} s_1, (a_1, b_1) \\ s_1, (a_2, b_1) \\ s_2, (a_1, b_1) \\ s_2, (a_2, b_1) \\ s_3, (a_1, b_2) \\ s_3, (a_2, b_2) \\ s_4, (a_1, b_2) \\ s_4, (a_2, b_1) \end{matrix} \begin{pmatrix} 1/2 \\ 1/3 \\ 1/2 \\ 1/3 \\ 1/2 \\ 1/3 \\ 1/2 \\ 1/3 \end{pmatrix}$$

***Iteration** $v = 3$**:** Bestimmen des kleinsten zuteilbaren Teiles:*

$$t_{a_2}(3) = \frac{\frac{2}{3}}{4} = \frac{1}{6} \qquad t_{b_2}(3) = \frac{\frac{2}{3}}{4} = \frac{1}{6}$$

$$\Rightarrow t(3) = t(2) + \frac{1}{6} = 1$$

Aktualisieren des konsumierten Maßes bzw. der Kapazitäten:

$$q^3 = \begin{array}{c} \begin{matrix} a_1 & a_2 & b_1 & b_2 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Sowohl Kurse a_2 als auch b_2 sind nun komplett aufgebraucht, es können keine weiteren Anforderungen z^3 durch Studierende gestellt werden. Weiterhin ist $t(3) = 1$, somit terminiert der Algorithmus. Die finale zufällige Zuteilung lautet nun:

$$\begin{aligned} BPS(3) &= BPS(2) + (t(3) - t(2)) \cdot z^2 \\ &= \begin{matrix} s_1,(a_1,b_1) \\ s_1,(a_2,b_1) \\ s_2,(a_1,b_1) \\ s_2,(a_2,b_1) \\ s_3,(a_1,b_2) \\ s_3,(a_2,b_2) \\ s_4,(a_1,b_2) \\ s_4,(a_2,b_1) \end{matrix} \begin{pmatrix} 1/2 \\ 1/3 \\ 1/2 \\ 1/3 \\ 1/2 \\ 1/3 \\ 1/2 \\ 1/3 \end{pmatrix} + \frac{1}{6} \cdot \begin{matrix} s_1,(a_2,b_2) \\ s_2,(a_2,b_2) \\ s_3,(a_2,b_2) \\ s_4,(a_2,b_2) \end{matrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\ &= \begin{matrix} s_1,(a_1,b_1) \\ s_1,(a_2,b_1) \\ s_1,(a_2,b_2) \\ s_2,(a_1,b_1) \\ s_2,(a_2,b_1) \\ s_2,(a_2,b_2) \\ s_3,(a_1,b_2) \\ s_3,(a_2,b_2) \\ s_4,(a_1,b_2) \\ s_4,(a_2,b_1) \\ s_4,(a_2,b_2) \end{matrix} \begin{pmatrix} 1/2 \\ 1/3 \\ 1/6 \\ 1/2 \\ 1/3 \\ 1/6 \\ 1/2 \\ 1/2 \\ 1/2 \\ 1/3 \\ 1/6 \end{pmatrix} \end{aligned}$$

Beispiel 3.2.9 zeigte den BPS-Mechanismus unter der Annahme, dass alle Studierenden beide angebotenen Lehrveranstaltungen A und B besuchen. Typischerweise besuchen aber Studierende je nach Studiengang und Fachsemester unterschiedliche Veranstaltungen und haben entsprechend Veranstaltungsbündel mit unterschiedlicher Länge k. Die Präferenzrelation $\succ_i$ eines jeden Studierenden geht nur über seine Veranstaltungsbündel.

Für die Erzeugung einer zulässigen zufälligen Zuteilung mithilfe von BPS spielt jedoch die Länge der Bündel keine Rolle, denn es werden stets nur Angebot und Nachfrage der jeweiligen angeforderten Kurse betrachtet. BPS sucht immer den kleinsten zuteilbaren Teil für jeden angefordeten Kurs, sodass dieser Kurs vollbesetzt ist. Das heißt, dass es bei den restlichen Kursen mindestens so große zuteilbare Teile gibt. Insbesondere bedeutet das, dass es bei keinem anderem Kurs mehr als die vorhandene Kapazität zugeteilt wird und der zuzuteilende Teil von allen Kursen erfüllt werden kann. Für jeden Studierenden werden Bündel aus seiner Präferenzordnung nur entfernt, wenn mindestens ein Kurs des Bündels keine Kapazitäten frei hat.

Nachdem nun eine ordinal effiziente und neidfreie zufällige Zuteilung berechnet wurde, muss diese Zuteilung in eine Wahrscheinlichkeitsverteilung von zulässigen deterministischen Zuteilungen zerlegt werden.

3.3 Dekomposition in zulässige deterministische Zuteilungen

Das Ziel ist es nun, die erhaltene zufällige kombinatorische Zuteilung möglichst gut durch deterministische kombinatorische Zuteilungen anzunähern. In Kapitel 2.1 hat man gesehen, dass im nicht-kombinatorischen Fall das Birkhoff-von-Neumann-Theorem zur Zerlegung von zufälligen Zuteilungen in eine Konvexkombination aus deterministischen Zuteilungen genutzt werden kann. Weiter haben Nguyen, Peivandi und Vohra gezeigt, dass dieses Theorem auf Anwendungsfälle der kombinatorischen Zuteilung generalisiert werden kann, wenn man die Restriktionen des Angebots lockert (Theorem 2.1.7). Insbesondere in der alternativen Betrachtung des Birkhoff-von-Neumann-Theorems sieht man, dass mithilfe eines beliebigen Kostenvektors u immer eine integrale Lösung gefunden werden kann, sodass gilt $u \cdot \bar{x} \geq u \cdot x$. Es müssen jedoch in der Zuteilung von Bündel, also $k \geq 1$, gegebenenfalls die Kapazitäten von einigen Kursen um maximal $k - 1$ überschritten werden.

Zum Beweis der Generalisierung haben Nguyen, Peivandi und Vohra einen Lotterie-Algorithmus (eng. *„Lottery Algorithm“*) vorgelegt, der in polynomieller Zeit die fraktionale Lösung annähern kann. Die grundsätzliche Idee ist, schrittweise eine konvexe Hülle mit maximal $dim(x) + 1$ integralen Lösungen um die zufällige Zuteilung zu bilden. $dim(x)$ bezeichne die Dimension des Vektors x. Eine konvexe Hülle ist die Menge aller möglichen Konvexkombinationen von einer Lösungsmenge. Jede (fraktionale) Lösung innerhalb dieser konvexen Hülle kann durch eine Konvexkombination integraler Lösungen, die diese Hülle aufspannen, dargestellt werden. Es soll eine Lösungsmenge gefunden werden, sodass x durch eine Konvexkombination integraler Lösungen angenähert dargestellt werden kann. Dieser Algorithmus kann sich beliebig nahe der fraktionalen Lösung (zufälligen Zuteilung) annähern.

Zunächst wird das Iterative Rundungsverfahren (eng. *Iterative Rounding Algorithm*) erläutert, welches es ermöglicht unter einem gegebenen Kostenvektor u eine integrale Lösung zu finden. Eine gefundene integrale Lösung soll der konvexen Hülle hinzugefügt werden, wenn sie zur Annäherung beiträgt.

Bei Zuteilungen von vielen Studierenden ist es möglich, dass die fraktionale Lösung am Rand der Menge von Lösungen, die Nachfrage erfüllen, liegt. Die Problematik dieses Falls und das Beheben des Problems durch die Skalierung der fraktionalen Lösung wird in Kapitel 3.3.2 erläutert.

Abschließend kann die Funktionsweise des Lotterie-Algorithmus und die Implementierung dessen näher beschrieben werden.

3.3.1 Iteratives Rundungsverfahren

Das Iterative Rundungsverfahren wird als Subroutine im Lotterie-Algorithmus verwendet, um eine integrale Lösung $\bar{x}$ zu erhalten, sodass gilt $u \cdot \bar{x} \geq u \cdot x$. Der Kostenvektor u wird vom Lotterie-Algorithmus festgelegt.

In Kapitel 2.1 wurde gezeigt, dass bei einer Konvexkombination integraler Lösungen, die eine fraktionale Lösung umhüllt, stets eine integrale Lösung mit Kostenvektor u gefunden werden kann. Dies gilt nicht nur für die integralen Lösungen der Konvexkombination, sondern insbesondere für die Menge der integralen Lösungen, die Nachfrage und Angebot $+ k - 1$ erfüllen (Beweis zu Theorem 2.1 aus [21]).

3.3.1 Lemma (nach Lemma 2.2 aus [21]) *Bei jedem (nicht zwingend nicht-negativen) Kostenvektor u und jeder fraktionalen Lösung x, welche Zuteilungswsk., Nachfrage und Angebot erfüllt, kann in polynomieller Zeit eine integrale Lösung $\bar{x}$ gefunden werden, die Zugeteilt, Nachfrage und Angebot $+ k - 1$ erfüllt, sodass gilt $u \cdot \bar{x} \geq u \cdot x$.*

Das Verfahren kann als Optimierungsproblem formalisiert werden und die Implementierung dessen kann Algorithmus 3 entnommen werden:

$$
\begin{array}{lrcll}
\max\limits_{\bar{x}} & u\bar{x} & & & \\
\text{unter} & \bar{x} & \geq & 0 & \\
 & \bar{x} & \in & \{0,1\}^{|S\times B|} & \text{(Integral)} \\
 & \sum\limits_{i\in S,\ b\in B} \bar{x}_{ib} b_j & \leq & q_j \qquad j \in C & \text{(Angebot)} \\
 & \sum\limits_{b\in B} \bar{x}_{ib} & \leq & 1 \qquad \forall i \in S & \text{(Nachfrage)}
\end{array}
\tag{3}
$$

Solver werden spezielle Computerprogramme genannt, die Optimierungsprobleme lösen. Dadurch, dass es in kombinatorischen Zuteilungsproblemen möglicherweise keine optimale integrale Lösung gibt, die die Restriktionen bezüglich Nachfrage und Angebot erfüllen, muss eine relaxierte integrale Lösung iterativ gefunden werden.

Zunächst sucht das Iterative Rundungsverfahren einen Extrempunkt x^*, welches das Optimierungsproblem ohne die Bedingung (Integral) löst. Die Komponenten der Lösung des Optimierungsproblems müssen aufgrunddessen nicht 0 oder 1 sein. Durch die Maximierung der Zielfunktion können jedoch einige Komponenten 0 beziehungsweise 1 sein. Wenn alle Komponenten des Extrempunktvektors ganzzahlig sind, terminiert der Algorithmus.

Falls es Komponenten in der Extrempunktlösung gibt, die ganzzahlig sind, werden diese Komponenten fixiert. Es wird ein aktualisiertes lineares Optimierungsproblem konstruiert und diese Komponenten sind nicht mehr variabel. Entsprechend müssen auch die Restriktionen bezüglich des Angebots angepasst werden: Wenn eine Komponente $x^*_{ib} = 1$ für einen Studierenden i und einem Bündel b gilt, sind dem Studierenden i in dieser Lösung die Kurse aus dem Bündel zugewiesen und die Kapazitäten der jeweiligen Kurse müssen reduziert werden. Gilt $x^*_{ib} = 0$ für einen Studierenden i und einen Bündel b, wird dem Studierenden i das Bündel b nicht zugewiesen.

Wenn alle Komponenten der Extrempunktlösung x^* fraktional sind, muss ebenfalls ein neues Optimierungsproblem konstruiert werden. Dieser Fall tritt ein, wenn es mindestens einen Kurs gibt, welcher von mehreren Studierenden beansprucht wird, aber die Kapazitäten zur Erfüllung dieser Ansprüche nicht ausreichen. Hier müssen die Restriktionen des Angebots gelockert werden. Es gibt mindestens einen Kurs, der durch eine Überbelegung von maximal $k-1$ Plätzen die Anforderungen der Studierenden erfüllt. Die Restriktionen bezüglich solcher Kurse im Optimierungsproblem sollen entfernt werden und somit wird x^* auf eine Lösung gerundet, welche Nachfrage und Angebot $+\ k-1$ erfüllt.

Es werden solange neue Optimierungsprobleme entsprechend der zwei angegebenen Fälle konstruiert und gelöst, bis alle Komponenten entweder 0 oder 1 sind.

3.3.2 Skalieren einer fraktionalen Lösung

Es werden zwei weitere Parameter δ und ϵ für den Lotterie-Algorithmus benötigt, um die Berechnung einer konvexen Hülle um die fraktionale Lösung durchzuführen. Die fraktionale Lösung x soll durch eine Konvexkombination aus integralen Lösungen angenähert dargestellt werden. Dazu wird schrittweise eine konvexe Hülle gebildet, sodass ein Punkt y, welches in der konvexen Hülle enthalten ist, der fraktionalen Lösung nah ist.

3.3.2 Bezeichnung *ϵ sei die maximale Distanz, die ein angenäherter Punkt y der konvexen Hülle zur fraktionalen Lösung x haben muss, sodass y und ihre Konvexkombination als Wahrscheinlichkeitsverteilung über integrale Lösungen angenommen wird.*

Algorithm 3 Iteratives Rundungsverfahren

function IRA(u, x, k)
  $x^* \leftarrow max\{u \cdot x \mid (Nachfrage), (Angebot), x \geq 0\}$
    ▷ Lösen des Optimierungsproblems nach (3)
  while any $x^*_{ib} \neq 0$ **or** $x^*_{ib} \neq 1, \forall i \in S, b \in B$ **do**
    if any $x^*_{ib} = 0$ **or** $x^*_{ib} = 0$, $\forall i \in S, b \in B$ **then**
      for all $x^*_{ib} \in \{0, 1\}, \forall i \in S, b \in B$ **do**
        Fixiere Variable x^*_{ib}.
        for all $j \in b$ **do**
          $q_j \leftarrow q_j - 1$
        end for
      end for
    else if all $0 < x_{ib} < 1, \forall i \in S, b \in B$ **then**
      for all $j \in C$ **do**
        if $\sum_{i \in S,\ b \in B} \lceil x_{ib} \rceil b_j \leq q_j + k - 1$ **then**
          Entferne zugehörige Restriktion (*Angebot*) für Kurs c_j.
        end if
      end for
    end if
    $x^* \leftarrow max\{u \cdot x \mid (Nachfrage), (Angebot), x \geq 0\}$
      ▷ Lösen des aktualisierten Optimierungsproblems
  end while
  return x^{opt}
end function

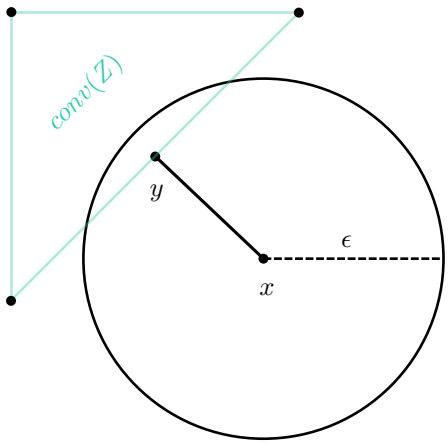


Abbildung 2: Nächster Punkt y der konvexen Hülle $conv(Z)$, welcher als Lösung akzeptiert wird. Adaptiert aus [34].

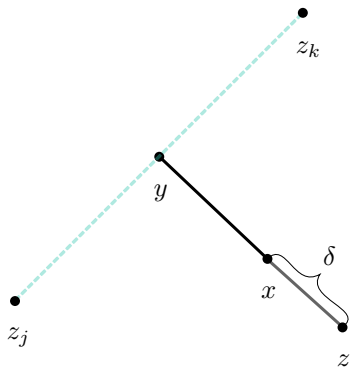


Abbildung 3: Projizieren des Punktes y auf die Gegenseite von x mit Abstand δ. Adaptiert aus [34].

Abbildung 2 zeigt beispielhaft eine Lösungsmenge Z mit drei integralen Lösungen. y liegt in der konvexen Hülle der Lösungsmenge $conv(Z)$ und kann insbesondere durch eine Konvexkombination dieser drei integralen Punkte dargestellt werden. Weiterhin ist y der nächste Punkt zur fraktionalen Lösung x. Dieser angenäherter Punkt und dessen Konvexkombination wird als Lösung akzeptiert, da der Abstand zwischen x und y kleiner als ϵ beträgt.

3.3.3 Bezeichnung *δ definiert den Abstand zwischen der fraktionalen Lösung x und einem weiteren (nicht zwingend integralen) Punkt $z = \delta\frac{x-y}{\|x-y\|}$. z ist der Punkt, der sich durch die Projektion des Punkts y auf die Gegenseite von x mit Abstand δ ergibt.*

In Abbildung 3 liegt Punkt y auf einer Facette der konvexen Hülle, welche im zweidimensionalen Fall eine Gerade ist. Insbesondere kann y alleinig durch die Punkte z_j und z_k der Lösungsmenge Z beschrieben werden. Es kann ein neuer Punkt z durch die Projektion auf die Gegenseite von x gefunden werden. Die Gegenrichtung ist durch $x-y$ bestimmt und der Abstand des Punktes z zu x beträgt δ, da der Vektor $x - y$ normiert und um δ gestreckt wurde.

Es ist jedoch zu beachten, dass der projizierte Punkt z weiterhin die Restriktionen bezüglich der Nachfrage erfüllen muss. Die Summe der Zuteilungswahrscheinlichkeiten eines Studierenden darf nicht größer eins sein. Daher muss δ so gewählt werden, dass es dem minimalen Abstand zum Rand der Menge von Lösungen, die die Nachfrage erfüllen, entspricht. Andernfalls ist es möglich, δ so zu wählen, dass die Summe der Zuteilungswahrscheinlichkeiten eines Studierenden größer eins ist. Dies würde die Restriktion der Nachfrage verletzen. Die Restriktion Angebot $+ k-1$ muss zunächst durch die Projektion nicht beachtet werden, denn z wird auf $\bar{z}$ gerundet, sodass Angebot $+ k - 1$ und Nachfrage erfüllt werden. Das Iterative Rundungsverfahren versucht Angebot zu erfüllen, ansonsten wird gegebenenfalls die Restriktion gelockert, sodass Angebot $+ k - 1$ erfüllt ist. Ist jedoch die Restriktion der Nachfrage bereits verletzt, können keine gerundete integrale Lösung gefunden werden kann, sodass Angebot $+ k - 1$ und Nachfrage gilt.

$$\delta = min\{1 - \sum_{b \in B} x_{ib} \mid \forall i \in S\}$$

Abbildung 4 stellt die Wahl eines minimalen δ grafisch dar. Die Menge aller Lösungen, die die Nachfrage erfüllen, wird durch die graue Fläche visualisiert und ist durch zwei Linien begrenzt. Diese Linien entsprechen jeweils der maximale Summe der Zuteilungswahrscheinlichkeiten zweier Studierenden. Lösungen außerhalb der grauen Fläche würden die Restriktion der Nachfrage

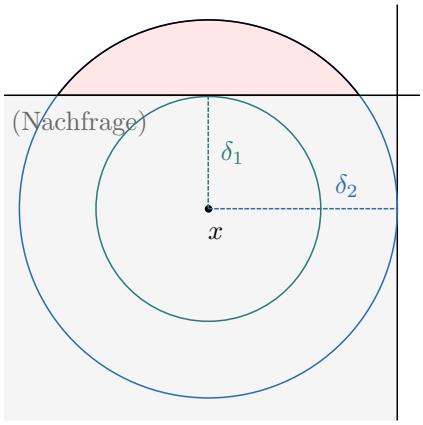


Abbildung 4: Wahl von δ, sodass minimaler Abstand zum Rand der Menge, die Nachfrage erfüllt, gilt.

verletzen. Gilt $\delta = \delta_2$, dann ist es möglich, ein Punkt z zu finden, welcher außerhalb der grauen Fläche ist. Daher muss δ stets so gewählt werden, dass es dem minimalen Abstand zum Rand der Menge, die Nachfrage erfüllen, entspricht.

Nun ist es insbesondere in großen Probleminstanzen mit vielen Studierenden und Bündeln möglich, dass der minimale Abstand zum Rand gleich null ist beziehungsweise die Summe der Zuteilungswahrscheinlichkeiten eines Studierenden gleich eins ist. Die fraktionale Lösung muss entsprechend mit einem Faktor d herunterskaliert werden, dass die Lösung sich abseits vom Rand aber innerhalb der Menge, die die Nachfrage erfüllt, befindet. Insbesondere muss dann auch der Radius der ϵ-Umgebung angepasst werden, da das Verschieben der ϵ-Umgebung dazu führen könnte, dass ungewollte Annäherungen akzeptiert werden. Weiterhin soll die fraktionale Lösung so herunterskaliert werden, dass der Abstand zum Rand kleiner ϵ beträgt. Andernfalls liege x auf dem Rand oder außerhalb der ϵ-Umgebung und es können keine Annäherungen akzeptiert werden. Die Methode der Skalierung folgt im Wesentlichen der Masterarbeit von Uzunoglu [34].

3.3.4 Bezeichnung *$x_{sc} = d \cdot x$, $0 < d < 1$ sei eine um Faktor d skalierte fraktionale Lösung.*

3.3.5 Bezeichnung *β sei der maximale Abstand, der zwischen ursprünglichen fraktionalen Lösung und der skalierten fraktionalen Lösung gelten soll. Weiterhin darf der Abstand nicht größer als ϵ sein. Parameter $\eta \in \mathbb{R}$ mit $\eta > 1$ bestimmt den Bruchteil von ϵ, um welchen sich die skalierte Lösung vom Rand beziehungsweise von der ursprünglichen Lösung befinden soll.*

$$\beta = \frac{\epsilon}{\eta}$$
$$\|x_{sc} - x\| \leq \beta$$

3.3.6 Bezeichnung *Die angepasste ϵ-Umgebung um die skalierte fraktionalen Lösung x_{sc} sei durch Radius α gegeben. Der Radius α ist so gewählt, dass angenäherte Lösungen, die von der angepassten ϵ-Umgebung akzeptiert werden, ebenfalls durch die ursprüngliche ϵ-Umgebung angenommen wären.*

$$\alpha = \frac{\eta - 1}{\eta} \cdot \epsilon$$

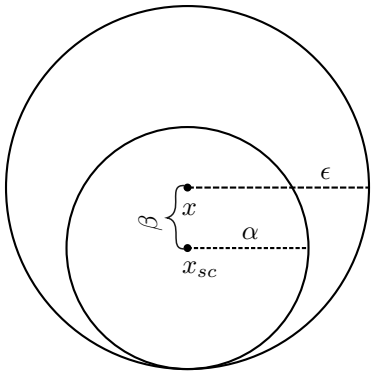


Abbildung 5: Skalieren der fraktionalen Lösung x und ihrer ϵ-Umgebung. Adaptiert nach [34].

Nun soll der kleinstmögliche Faktor d bestimmt werden, welches dennoch $\|x_{sc} - x\| \leq \beta$ erfüllt. Es wird hierbei eine skalierte fraktionale Lösung mit größtmöglichem Abstand zur ursprünglichen fraktionalen Lösung gesucht:

$$
\begin{aligned}
& \|x_{sc} - x\| = \beta \\
\Leftrightarrow\ & \|d \cdot x - x\| = \beta \\
\Leftrightarrow\ & \|(d-1) \cdot x\| = \beta \\
\Leftrightarrow\ & |d-1| \cdot \|x\| = \beta \\
\Leftrightarrow\ & |1-d| \cdot \|x\| = \beta \\
\Leftrightarrow\ & (1-d) \cdot \|x\| = \beta \\
\Leftrightarrow\ & 1 - d = \frac{\beta}{\|x\|} \\
\Leftrightarrow\ & 1 - d = \frac{\beta}{\|x\|} \\
\Leftrightarrow\ & d = 1 - \frac{\beta}{\|x\|} = 1 - \frac{\epsilon}{\eta \cdot \|x\|}
\end{aligned}
$$

Abbildung 5 zeigt die Skalierung der fraktionalen Lösung x, welcher am Rand liegt. x_{sc} wurde insgesamt um Faktor d kleiner skaliert und liegt abseits von x und insbesondere vom Rand mit Abstand β. Somit kann $\delta > 0$ so gewählt werden. Besonders zu beachten ist, dass angenäherte Lösungen, die durch die α-Umgebung akzeptiert werden, ebenfalls von der ursprünglichen ϵ-Umgebung angenommen worden wären.

3.3.3 Lotterie-Algorithmus

Es wurde festgestellt, dass eine fraktionale Lösung stets von integralen Lösungen umschlossen ist. Insbesondere zeigen Nguyen, Peivandi und Vohra, dass es keine Hyperebene gibt, die x von der Menge der integralen Lösungen trennt [21]. Diese Voraussetzung kann genutzt werden, um schrittweise eine Teilmenge integraler Lösungen zu finden, sodass es eine Lösung innerhalb der konvexen Hülle dieser Teilmenge gibt, die sich ebenfalls in der ϵ-Umgebung der fraktionalen Lösung befindet.

x sei hier die—wenn nötig—bereits passend skalierte fraktionale Lösung, andernfalls die ursprüngliche fraktionale Lösung. Die (angepasste) ϵ-Umgebung erlaube Annäherungen an x mit einer Distanz von $\epsilon > 0$. δ sei weiterhin der minimale Abstand zum Rand der Menge von Lösungen, die die Nachfrage erfüllen. Es gilt, dass $\delta > 0$, da sonst die fraktionale Lösung nach vorigem Kapitel skaliert worden wäre. $\mathcal{B}(x, \delta)$ sei eine nicht-leere Menge fraktionaler Lösungen mit Abstand δ um x, die die Nachfrage erfüllen. Z bezeichne die Lösungsmenge, also die Teilmenge ausgewählter integraler Lösungen. Weiterhin sei $conv(Z)$ die konvexe Hülle der Lösungsmenge.

$$\delta = min\{1 - \sum_{b \in B} x_{ib} \mid \forall i \in S\}$$

$$\mathcal{B}(x, \delta) = \{z \mid |x - z| \leq \delta\} \subseteq \{z \in \mathbb{R}^{|S \times B|} \mid (Nachfrage), z \geq 0\}$$

Algorithm 4 Bestimmen des initialen Kostenvektors

```
1: function InitialCostVector(x, (≻_i)_{i∈S})
2:     maxAmountBundles ← max{|{x_ib}| | ∀i ∈ S}
3:     for all i in x.indices do
4:         x_i ← sort(x_i, ≻_i)
5:         value ← maxAmountBundles
6:         for all b in x_i.indices do
7:             u_ib ← value
8:             value ← value − 1
9:         end for
10:    end for
11: end function
```

Algorithm 5 Lotterie-Algorithmus

```
1: function Lottery(x, δ, ϵ, k)
2:     u ← InitialCostVector(x, (≻_i)_{i∈S})
3:     Z ← {IRA(u, x, k)}
4:     y ← argmin{|x − y| | y ∈ conv(Z)}
5:     while |x − y| < ϵ do
6:         z ← x + δ (x−y)/|x−y|
7:         z̄ ← IRA(x − y, z, k)
8:         Wähle Z' ⊂ Z der Größe |Z'| ≤ dim(x) und y ∈ conv(Z')
9:         Z ← Z' ∪ {z̄}
10:        y ← argmin{|x − y| | y ∈ conv(Z)}
11:    end while
12:    return y und die Konvexkombination von y
13: end function
```

Die Funktionsweise des Lotterie-Algorithmus (Algorithmus 5) wird im folgenden beschrieben:

Für die Lösungsmenge Z wird eine erste integrale Lösung mithilfe des Iterativen Rundungsverfahrens nach Kapitel 2.1 gesucht. Dabei ist der Kostenvektor u initial so gewählt, dass dieser den Präferenzen der Studierenden entspricht. Das am meisten präferierte Bündel im Zuteilungsvektor x eines jeden Studierenden erhält als Koeffizient die maximale Anzahl von Veranstaltungsbündel eines Studierenden im Zuteilungsvektor. Die jeweils folgenden präferierten Bündel erhalten

einen um eins erniedrigten Wert. Es wird versucht einen maximalen Extrempunkt zu finden, dabei sollen Komponenten x_{ib} mit hohen Koeffizienten möglichst gleich eins gesetzt werden. Das bedeutet, dass eine integrale Lösung gefunden werden soll, die die Präferenzen der Studierenden möglichst gut erfüllt. Eine Implementierung der Bestimmung des initialen Kostenvektors wird in Algorithmus 4 gezeigt.

Diese gefundene erste integrale Lösung ist jedoch nicht zwingend ideal, denn es muss eine Annäherung von x gefunden werden, sodass die Voraussetzungen der Neidfreiheit und schwacher Schutz gegen strategisches Handeln ebenfalls erfüllt sind. Weiter ist die gefundene Lösung nicht ideal, wenn sie außerhalb der ϵ-Umgebung ist. Die konvexe Hülle eines einzelnen Punktes ist der Punkt selber. Daher wird der Abstand zwischen der ersten integralen Lösung und der fraktionalen Lösung bestimmt.

Nun wird der nächste Punkt der konvexen Hülle zur fraktionalen Lösung y (hier: erste integrale Lösung) bestimmt und auf die Gegenseite von x projiziert. Das bedeutet, dass in $\mathcal{B}(x, \delta)$ ein (fraktionaler) Punkt $z = \delta \frac{x-y}{\|x-y\|}$ gesucht wird. Ebenfalls in der entgegengesetzten Richtung zu y wird mithilfe des Iterativen Rundungsverfahren eine weitere integrale Lösung gesucht, sodass gilt $(x-y) \cdot \bar{z} \geq (x-y) \cdot z$. Der Kostenvektor sei nun $u = x - y$. Die neue integrale Lösung $\bar{z}$ werde der Lösungsmenge Z hinzugefügt.

Insgesamt werden schrittweise konvexe Hüllen gebildet, die geometrisch gesehen konvexe Polytope sind. Zunächst ist das konvexe Polytop nur ein Punkt, dann eine Gerade, anschließend ein Fläche bis zur einem $dim(x) + 1$-dimensionalen Polytop. In dem gebildeten Polytop wird der Punkt y mit kürzestem Abstand zu x gesucht. Der nächste Punkt y wird dann wieder mit Abstand δ auf die Gegenseite von x projiziert. Die Berechnung des nächsten Punkts $y = \sum_{p=1}^{q} \lambda_p \cdot z_p$ mit $z_p \in Z$ der konvexen Hülle zu x kann als quadratisches Optimierungsproblem formalisiert werden:

$$\begin{aligned} \min_{(\lambda_1,\ldots,\lambda_q)} \quad & \|x-y\|^2 = \|x - \sum_{p=1}^{q} \lambda_p z_p\|^2 \\ \text{unter} \quad & 0 \geq \lambda_p \geq 1 \qquad\qquad p = 1 \ldots q \\ & \sum_{p=1}^{q} \lambda_p = 1 \end{aligned} \tag{4}$$

Es ist zu beachten, dass y stets auf der Facette (beispielsweise Kante eines Polygons oder Seitenfläche eines Polyeders, et cetera) liegt. Das bedeutet, dass die Lösungsmenge auf die Menge der Punkte, die die Facette beschreiben, reduziert werden kann. Insbesondere kann dann $Z' \subset Z$ mit $|Z'| \leq dim(x)$ und $y \in conv(Z')$ gewählt werden.

Der Algorithmus terminiert dann, wenn ein Punkt y gefunden wurde, dessen Abstand zu x kleiner als ϵ ist. Andernfalls wird das Verfahren fortgeführt.

In Abbildung 6 wird vor allem der letzte Iterationsschritt des Lotterie-Algorithmus gezeigt. Zunächst wurde bis zu diesem Zeitpunkt eine konvexe Hülle $conv(Z)$ aufgespannt, dessen nächster Punkt y zur fraktionaler Lösung x jedoch außerhalb der ϵ-Umgebung ist. Punkt z resultierte aus der Projektion von y auf die Gegenseite und es wurde eine integrale Lösung z' gesucht. z' befindet sich auf der Grafik links von der roten gestrichelten Linie, denn mithilfe des Iterativen Rundungsverfahrens soll eine integrale Lösung gefunden, sodass $(x-y) \cdot z' \geq (x-y) \cdot z$ gilt. Da y auf der Facette der konvexen Hülle liegt, reicht es für die Teilmenge von Z nur die Punkte der Facette auszuwählen. Die neue integrale Lösung wird ebenfalls zur Lösungsmenge Z hinzugefügt und es wird neue neue konvexe Hülle gespannt.

Jetzt befindet sich der nächste Punkt y zur fraktionalen Lösung innerhalb der ϵ-Umgebung und somit terminiert der Algorithmus. Der resultierende Punkt y und seine Konvexkombination kann

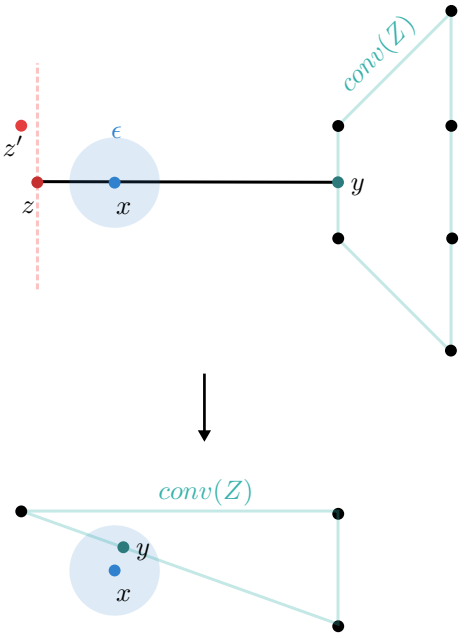


Abbildung 6: Letzter Iterationsschritt des Lotterie-Algorithmus. Adaptiert nach [19].

als Wahrscheinlichkeitsverteilung über die integralen Punkte der Lösungsmenge genutzt werden. Es wird zufällig eine integrale Lösung aus dieser Wahrscheinlichkeitsverteilung bestimmt.

3.4 Implementierung des Bundled Probabilistic Serial Mechanismus

3.4.1 Python-Paket `bundle_allocation`

Die Algorithmen des Bundled Probabilistic Serial sind in Python 3.7 implementiert worden und werden als Python-Paket (Bibliothek) `bundle_allocation` bereitgestellt. Das Python-Paket besteht aus vier Modulen:

- `bundle_allocation.entities`: Datenstrukturen zu Studierenden und ihre Präferenzlisten, Kursen sowie Zuteilungen
- `bundle_allocation.bundled_prob_serial`: Implementierung der Erzeugung einer zufälligen Zuteilung mit BPS (Algorithmus 2)
- `bundle_allocation.iterative_rounding`: Iteratives Rundungsverfahren zur Bestimmung einer deterministischen Zuteilung (Algorithmus 3)
- `bundle_allocation.lottery`: Zerlegung der zufälligen Zuteilung in eine Konvexkombination deterministischer Zuteilungen (Algorithmus 4 & 5)

Für die Implementierung werden die externen Pakete NumPy und gurobipy eingesetzt. NumPy stellt Datenstrukturen für Vektoren, Matrizen beziehungsweise generell n-dimensionale Felder bereit und bietet weiterhin effizient implementierte Funktionen für den Umgang mit solchen

Datenstrukturen [26]. Die Bibliothek steht unter der BSD-Lizenz und ist somit eine quelloffene Software, die frei verwendet werden kann [25]. Gurobi ist ein Solver, welcher unter anderem lineare sowie quadratische Optimierungsprobleme lösen kann [12]. Insbesondere stellt Gurobi ihre Python-Schnittstelle gurobipy zur Verfügung und kann in der Implementierung des Iterativen Rundungsverfahren und in der Bestimmung des nächsten Punktes der konvexen Hülle eingesetzt werden. Obwohl Gurobi proprietäre Software ist und im Allgemeinen ein Bezahlmodell für die Verwendung ihrer Software verfolgt, werden zum Zeitpunkt des Verfassens kostenfreie Lizenzen an akademische Institutionen ausgestellt [2].

Kurse sind als `Course`-Datenklassen repräsentiert, welche die Kurs-ID, den Kursnamen sowie ihre Kapazität als Attribute enthalten. Weiterhin wird ein Kurs nach abgeschlossener Zuteilung mit den zugeteilten Studierenden asoziiert.

`BundleItems` dienen der Zusammenfassung parallel stattfindender Kurse einer gleichen Lehrveranstaltung. Studierende können somit Prioritäten zu verfügbaren Zeitslots abgeben. Die Berechnung einer Zuteilung geschieht auf `BundleItems` und zugeteilte Studierende werden randomisiert und proportional zu den Kapazitäten auf die Kurse des `BundleItems` verteilt.

Die Klasse der Studierenden `Student` enthält als Attribute die ID des Studierenden, die Prioritätenlisten zu seinen Lehrveranstaltungen und die daraus erzeugte sortierte Liste von Veranstaltungsbündel. `Student` hat weiterhin die Methode `generate_bundles` nach Algorithmus 1, um eine Präferenzordnung über Veranstaltungsbündel nach seinen Prioritätenlisten zu erzeugen.

Abschließend entspricht eine Zuteilung der Datenklasse `Allocation`, welche aus einer Liste von `(Student, BundleItem)`-Tupel sowie einem NumPy-Array besteht. Das NumPy-Array ist eine effiziente Datenstruktur für Vektoren, Matrizen und n-dimensionaler Felder. In diesem Fall wird das NumPy-Array als Zuteilungsvektor verwendet und über die Indizes können `(Student, BundleItem)` mit richtigen Komponente im Vektor asoziiert werden.

3.4.2 Microservice

Microservices sind ein Softwarearchitekturmuster, bei dem unterschiedliche Aufgaben durch unabhängige Prozesse erledigt werden, die untereinander über leichtgewichtige und sprachunabhängige Mechanismen kommunizieren. [10]

Die Entscheidung zur Realisierung des Bundled Probabilistic Serial Mechanismus als Microservice wurde im Wesentlichen wegen zweierlei Gründe getroffen: Zum einen soll der Webserver des Digicampus nicht durch ressourcenintensive Berechnungen unnötig belastet werden. Zugriffe auf Seiten des Digicampus sollen weiterhin zügig bearbeitet werden und die Berechnung einer Zuteilung soll im Hintergrund geschehen. Zum anderen bietet die Programmiersprache PHP, auf die der Digicampus basiert, keine Schnittstellen oder Bibliotheken zur einfachen und effizienten Handhabung von Vektoren beziehungsweise komplexeren mathematischen Rechenoperationen.

Daher soll auf der Lehr- und Lernplattform Digicampus lediglich in geeigneter Weise die Präferenzen zu Übungen verschiedener Lehrveranstaltungen der Studierenden gesammelt und in eine sprachunabhängige Form gebracht werden. Die Berechnung einer Zuteilung liegt in der Verantwortung des Microservices, anschließend soll das Ergebnis wieder dem Digicampus übermittelt werden. Die Lehr- und Lernplattform übernimmt abschließend die Eintragung in die jeweiligen Kurse und das Benachrichtigen der Studierenden.

Die Implementierung des Microservices basiert ebenfalls auf Python 3.7 und verwendet als Basis das Framework Connexion. SQLite ist die genutzte relationale Datenbank zur Speicherung der Ergebnisse einer berechneten Zuteilung. Connexion ermöglicht es anhand einer API-Spezifikation nach dem OpenAPI Standard REST-API-Endpunkte einzurichten. REST-APIs sind Programmschnittstellen, die auf HTTP (Hypertext Transfer Protocol) basieren und über HTTP-Verben wie beispielsweise `GET` oder `POST` Ressourcen anfordern, erstellen oder verändern. Die in der Spezifi-

kation angegebene Funktion des Endpunkts verarbeitet die Anfrage und gibt eine entsprechende Antwort zurück. Vorteile der Verwendung dieses Frameworks ist der vereinfachte Entwicklungsprozess und die einhergehende Dokumentation der erwarteten Funktionsweise der API [30]. Weiterhin bietet Connexion eine automatische Validierung der Anfrage und der Endpunktparameter sowie eine passende Serialisierung der Rückantwort per Spezifikation.

RQ (Redis Queue) wird zusammen mit der Redis Datenbank als Jobwarteschlange eingesetzt. Redis ist ein In-Memory-Speicher für Datenstrukturen, welcher als Datenbank, Cache und Nachrichtenvermittler (eng. *Message Broker*) eingesetzt werden kann [15]. Dabei wird der Arbeitsspeicher des Rechners als Datenspeicher für höhere Zugriffsgeschwindigkeiten genutzt. Ein Job ist ein Pythonobjekt, welches eine Funktion repräsentiert, die asynchron in einem weiteren Hintergrundprozess ausgeführt werden soll. Worker sind Pythonprozesse, die typischerweise im Hintergrund laufen und ausstehende Jobs erledigen [31]. Je nach Rechenkapazität können ein oder mehrere Worker eingesetzt werden, um Jobs zu verarbeiten.

Die Kommunikation mit dem Microservice geschieht über HTTP und dabei werden JSON-Dokumente (JavaScript Object Notation) übertragen, um Informationen zwischen Digicampus und Microservice auszutauschen. JSON ist ein Dateiformat, welches in menschenlesbarer Form Datenobjekte wie Schlüssel-Werte-Paare, Felder und primitive Datentype serialisiert [9].

Die API-Spezifikation dieses Microservices ist entsprechend dem OpenAPI Standard eine YAML-Datei (YAML Ain't Markup Language), die in einer menschenlesbaren Datenserialisierungssprache die Schnittstellen beschreibt [27]. Es wurde spezifiziert, dass die REST-API des Microservices drei Endpunkte besitzt.

- `POST /allocation`: Neue Anfrage einer Berechnung mit Angabe von Endpunkten für das Beziehen der relevanten Informationen und der Rückübermittlung der berechneten Daten (Callback-Endpunkt)
- `GET /allocation/{allocation_id}`: Anfordern des berechneten Ergebnisses
- `GET /job/{job_id}`: Abfrage des Jobstatus

Zunächst kann über die POST-Anfragemethode an `/allocation` ein neuer Auftrag zur Berechnung einer Zuteilung übermittelt werden. Dem Microservice wird mitgeteilt, wo Informationen zu den (zusammengefassten) Kursen, den Überschneidungen und den Prioritäten der Studierenden im JSON-Format abgefragt werden können sowie an welchen Callback-Endpunkt das Ergebnis der Zuteilung zurückübermittelt werden soll. Der Microservice fordert die Daten von der angegebenen Schnittstelle an. Mithilfe des Python-Pakets Marshmallow werden die angeforderten Daten nach einem im Voraus definierten Schema validiert und anschließend in die entsprechenden Datenstrukturen von `bundle_allocation` deserialisiert. Da die Berechnung einer Zuteilung CPU-lastig ist, soll mithilfe einer Jobwarteschlange eine Überlastung des Servers vermieden werden. Daher wird zunächst ein Job erstellt, welcher durch einen freien Worker bearbeitet werden kann. Es wird unmittelbar nach Eingang der Anfrage die Job-ID zurückgegeben, womit es dem Benutzer der API ermöglicht wird, den Status des Jobs über eine GET-Anfrage an `/job/{job_id}` zu einem späteren Zeitpunkt zu prüfen.

Der auszuführende Job verwendet die Funktionen des Pakets `bundle_allocation`, um jedem Studierenden zunächst seine Veranstaltungsbündel zu generieren und diese in die passende Präferenzordnung zu bringen. Anhanddessen soll eine zufällige Zuteilung erzeugt werden, die wiederum in eine Konvexkombination deterministischer Zuteilungen zerlegt wird. Anschließend wird zufällig eine deterministische Zuteilung aus dieser Wahrscheinlichkeitsverteilung ausgewählt. Das entsprechende Ergebnis wird in die SQLite Datenbank gespeichert.

Ist die Berechnung einer Zuteilung erfolgreich gewesen, kann das Ergebnis an den Callback-Endpunkt, der in der Anfrage angegeben wurde, übermittelt werden. Alternativ kann das Ergebnis über `/allocation/{allocation_id}` aus der Datenbank abgefragt werden. Der Endpunkt `/job/{job_id}` leitet nach erfolgreichen Erledigen des Jobs zum Ergebnis weiter.

Es wird auf Datensparsamkeit geachtet und nur notwendige Informationen übertragen und gespeichert. Die Pseudonymisierung der Studierenden durch Verwendung einer Identifikationsnummer statt der Benutzerkennung schließt eine Reidentifikation der Person ohne Zugriff auf das Quellsystem aus. Ebenfalls gilt das für Veranstaltungs-IDs, denn ohne Abgleich mit dem Digicampus, können keine Rückschlüsse auf die Veranstaltung getroffen werden. Nicht mehr benötigte Informationen wie die abgegebenen Prioritätenlisten der Studierenden et cetera werden unmittelbar nach der Berechnung verworfen. Die berechneten Ergebnisse werden nur für 48 Stunden gespeichert und werden anschließend gelöscht.

Der Microservice soll nur durch das universitätsinterne Netz erreichbar sein und darf nicht mit dem Internet verbunden sein. Der Verbindung zwischen Microservice und Client muss mit TLS (Transport Layer Security) verschlüsselt sein und alle Klartextverbindungen werden durch den Microservice abgewiesen. Clients müssen sich mithilfe eines Tokens authentifizieren, sodass gesichert werden kann, dass keine unbefugten Zugriffe auf Schnittstellen geschehen.

Jeder Token ist ein 16 Byte langer, zufällig erzeugter Schlüssel, welcher in der SQLite-Datenbank in gehashter Form gespeichert wird. Clients müssen bei Verwendung der REST-API stets ihren Token im `Authorization`-Header angeben.

4 Digicampus

Der Digicampus der Universität Augsburg ist das interaktive Portal zur Organisation des Studiums. Hier findet die Anmeldung zu nahezu allen Kursen, Seminaren, Vorlesungen und Übungen statt. Es werden weiter Werkzeuge zur Kommunikation und Kollaboration zur Verfügung gestellt. Basierend auf das Open-Source-Projekt Stud.IP (*Studienbegleitender Internetsupport von Präsenzlehre*) kann der Digicampus durch Plugins an die gegebenfalls speziellen Anforderungen der Universität angepasst werden. Plugins sind optionale Softwarekomponenten, die durch den Administrator installiert werden können und zur Laufzeit eingebunden werden. Es soll zunächst ein kurzer Überblick über die Interna des Systems gegeben werden. Anschließend wird die Implementierung des Plugins für die überschneidungsfreie präferenzbasierte Zuteilung vorgestellt.

4.1 Kernsystem Stud.IP und Plugin-Mechanismus

Stud.IP ist eine PHP-Webapplikation mit MySQL als relationale Datenbank, die im Herbst 1999 als studentische Initiative an der Universität Göttingen entstanden ist [6]. Seit Ende 2007 ist Stud.IP unter dem Namen Digicampus an der Universität Augsburg im Einsatz [24].

Die Stud.IP Entwicklung begann mit PHP 3 und war eine Ansammlung von Skripten, die, serverseitig ausgeführt, verschiedene Webseiten dynamisch erstellen können. PHP 3 hatte keine integrierte Sessionverwaltung, um zusammenhängende Zugriffe eines gleichen Benutzers auf verschiedene Seiten in eine Session (dt. *Sitzung*) zusammenzufassen. Dadurch sind die Seiten zustandslos und werden bei Aufruf stets von neuem erstellt. Erst durch Einbindung der Bibliothek PHPLib wurde es ermöglicht, Login-Mechanismen und sessionbasierte Variablen zu realisieren [23].

Über die Jahre wurde der Quellcode schrittweise durch neue Technologien und Strukturen ausgetauscht oder erweitert. Unter anderem basiert Stud.IP nun auf dem MVC-Paradigma, einem Softwarearchitekturmodell, bei der eine Anwendungskomponente in drei eigenständige Module unterteilt wird: Das Model (dt. *Modell*) dient der Speicherung der Daten und des Zustands der Anwendungskomponente, die View (dt. *Darstellung*) ist zuständig für die graphische Oberfläche und der Controller (dt. *Steuerung*) realisiert die Komponentenlogik sowie verwaltet die Einbeziehungsweise Ausgaben. Die Implementierung des MVC-Paradigma wurde über das Trails-Framework realisiert. Mit PHP 4 wurde auch die Sessionverwaltung von PHPLib durch die neu in der Sprache eingeführten Sessionverwaltung ersetzt. Heute bildet PHP 7 die Basis von Stud.IP.

Der `public`-Ordner des Stud.IP Quellcodes enthält unter anderem PHP-Skripte, die durch den Aufruf über Webclients vom Webserver ausgeführt werden. Hier finden sich die wichtigsten Einstiegspunkte `dispatch.php` und `plugins.php`, welche als sogenannte Dispatcher agieren und die Anfrage des Webbrowsers an den relevanten Controller im Kernsystem beziehungsweise im Plugin weiterreichen. Weiter enthält der `public`-Ordner Einstiegspunkte für die Kommunikation mit externen Systemen und Seiten, die nicht auf das MVC-Paradigma umgestiegen sind.

Es ist zu beachten, dass alle Controller von `StudipController` abgeleitet sind. Dieser Controller startet für den Seitenaufbau eine (wenn nötig neue) Session. Bei der Erstellung einer neuen Session werden über `lib/seminar_open.php` auch Stud.IP spezifische Sessionvariablen initialisiert, wie die Spracheinstellungen, das aktuelle Semester oder die „geöffnete“ Veranstaltung beziehungsweise Einrichtung des Benutzers. Nach dem Starten der Session werden noch die aktivierten Plugins initialisiert. Ab diesem Zeitpunkt können Plugins in den Ablauf eingreifen. Anschließend werden die speziellen Aktionen der jeweiligen Unterklasse ausgeführt und die Session wird gespeichert und geschlossen.

Stud.IP bietet mit der Plugin-Schnittstelle einen Mechanismus an, über den eigene Funktionen zu Stud.IP hinzugefügt werden können, ohne dabei das Kernsystem zu verändern. Das Aktua-

lisieren, Entfernen oder Hinzufügen von Komponenten ist dabei im laufenden Betrieb möglich [33].

Ein Stud.IP-Plugin ist eine ZIP-Datei, welche mindestens aus einer beschreibenden Manifest-Datei und der Pluginklasse mit dem Einstiegspunkt für die Ausführung besteht. Im Administrationsbereich kann die ZIP-Datei zur Installation hochgeladen werden, sodass sie vom Server am entsprechenden Pfad entpackt wird. Die Pluginklasse wird anschließend bei der Pluginverwaltungskomponente registriert.

Die Pluginklasse erbt von `StudIPPlugin` und kann ein oder mehrere Interfaces implementieren, um spezielle Funktionalitäten bereitzustellen.

Ist das Interface `RESTAPIPlugin` in der Pluginklasse implementiert, werden im Kernsystem weitere REST-API-Routen verfügbar gemacht, die über `api.php` angesprochen werden können. Es wird eine `RouteMap` Instanz definiert, welche die REST-API-Endpunkte mit ihren Anfragemethoden (`GET`, `POST`, `PUT`, `DELETE`) einer zugehörigen Funktion zuordnet. Diese RouteMap wird beim Router in `api.php` registriert, sodass der Router die eingehende Anfrage entsprechend weiterreichen kann.

Mit dem Interface `SystemPlugin` wird das Plugin bei jedem Seitenaufbau ausgeführt. Hierbei können neue Navigationspunkte eingehängt werden oder die darzustellende Seite vor der Ausgabe manipuliert werden.

Plugins können eigene Seiten ausliefern, wenn sie die erwartete Ordnerstruktur für das Model, die View und die Controller enthalten. Weiter müssen die Controller Unterklassen von `PluginController` sein, sodass sie über `plugins.php` aufrufbar sind.

Stud.IP bietet noch weitere Interfaces an, die Erweiterungen an verschiedenen Stellen des Kernsystems zulassen. Für diese Bachelorarbeit sind jedoch nur die erwähnten relevant.

Die ZIP-Datei kann weiter auch Migrationsskripte enthalten, die Änderungen am System beziehungsweise am Datenbankschema beschreiben und durchführen. Somit können unter anderem eigene Datenbanktabellen erzeugt werden oder neue Klassen dynamisch zur Laufzeit geladen werden. Ein Plugin kann weitere statische Ressourcen wie Bilder, CSS-Stylesheets oder JavaScript-Dateien bereitstellen. Es können externe PHP-Bibliotheken eingebunden werden, falls diese vom Plugin benötigt werden.

4.2 Anmeldesets

Veranstaltungen in Stud.IP sind standardmäßig für alle Benutzer des Systems offen. Das bedeutet, dass sich jede Person für einen beliebigen Kurs anmelden können. Einige Veranstaltungen erfordern jedoch Zugangsbeschränkungen, beispielsweise wenn nur für Studierende eines gewissen Studiengangs sich anmelden dürfen oder wenn Veranstaltungen nur eine beschränkte Anzahl von Sitzplätzen bereitstellen können. Mithilfe sogenannter **Anmelderegeln** können in Stud.IP verschiedene Anmeldeverfahren modelliert werden. Mehrere Veranstaltungen, die dasselbe Anmeldeverfahren haben, können in ein **Anmeldeset** zusammengefasst werden. Es ist zu beachten, dass nicht alle Regeln zueinander kompatibel sind.

Es werden folgende Regeln mitgeliefert:

Bedingte Anmeldung Nur Studierende, die die angegebenen Bedingungen erfüllen, wie beispielsweise bestimmte Studiengänge oder Fachsemester, können sich zur Veranstaltung anmelden.

Anmeldung zu maximal n Veranstaltungen Innerhalb eines Anmeldesets können Studierende nur an eine festgelegte Maximalanzahl von Veranstaltungen teilnehmen.

Anmeldung gesperrt Studierende können sich nicht zu den Veranstaltungen innerhalb dieses Anmeldesets anmelden.

Anmeldung mit Passwort Bei der Anmeldung zu den Veranstaltungen innerhalb dieses Anmeldesets wird ein Passwort abgefragt. Der Zugang zur Veranstaltung ist nur mit korrektem Passwort möglich.

Zeitgesteuerte Anmeldung Es wird ein Zeitfenster festgelegt, in dem die Anmeldung zu den Veranstaltungen des Anmeldesets möglich ist.

Beschränkte Teilnehmendenanzahl Es werden für die Veranstaltungen des Anmeldesets Kapazitäten festgelegt. Die Sitzplätze werden durch ein Losverfahren an die anfragenden Studierenden verteilt.

Veranstaltungsbezogene Anmeldung Studierende müssen gewisse Veranstaltungen belegen oder dürfen nicht Teilnehmer bestimmter Veranstaltungen sein, um sich zu Veranstaltungen dieses Anmeldesets anzumelden.

Bevorzugte Anmeldung Studierende bestimmter Studiengänge oder Fachsemester werden bei der Platzverteilung mit Losverfahren bevorzugt behandelt.

Im Falle der Übungsgruppenzuteilung ist es üblich, dass ein Lehrender ein Anmeldeset mit allen Übungen der jeweiligen Lehrveranstaltung anlegt. Nun gelten für diese Übungen dasselbe Anmeldeverfahren. Studierende sollen Prioritäten zu den jeweiligen Übungen abgeben und werden zu einem festgelegten Zeitpunkt durch das Losverfahren, welches in der Einleitung beschrieben wurde, einer Übung zugeteilt. Um dieses Anmeldeverfahren durch die gegebenen Anmelderegeln zu realisieren, benötigt es die Anmelderegeln „Anmeldung zu maximal n Veranstaltungen“ mit $n = 1$ und „Beschränkte Teilnehmendenanzahl“.

Jede Anmelderegel ist eine konkrete Implementierung der abstrakten Klasse `AdmissionRule`, welche die Anmeldelogik, die notwendigen Attribute und ihre entsprechenden HTML-Vorlagen für die Beschreibung der Anmelderegel und die Konfiguration der Regel durch Lehrende enthält. Es ist möglich eigene Regeldefinition zu erstellen, in dem die abstrakte Klasse den Anforderungen entsprechend implementiert wird. Instanzen von Anmelderegeln werden in ihrer jeweils eigenen Datenbanktabelle persistent gespeichert.

Zur Laufzeit können neue Regeln geladen werden, da Stud.IP über eine Datenbanktabelle von Anmelderegeln samt Pfad zur Klasse verfügt. Das Plugin fügt über ein Migrationsskript der Tabelle eine neue Zeile für eine neue Anmelderegel hinzu. Im Administrationsbereich können Anmelderegeln aktiviert beziehungsweise deaktiviert werden und weiter können die Kompabilitäten der Anmelderegeln zueinander konfiguriert werden.

Anmeldesets werden im System durch `CourseSet`-Objekte repräsentiert, die eine Liste von Veranstaltungen mit ein oder mehreren Anmelderegeln asoziieren. Jedes Anmeldeset wird von einem Lehrenden erstellt und wird einer Einrichtung (beispielsweise einer Fakultät oder einem Institut) zugewiesen. Der Ersteller kann das Anmeldeset zur gemeinsamen Verwendung innerhalb der Einrichtung freigeben, andernfalls erhalten nur der Ersteller und die Administratoren Zugriff auf das Set. Somit ist die Möglichkeit geschaffen, dass mehrere Lehrende der gleichen Einrichtung dasselbe Anmeldeverfahren für ihre Veranstaltungen anwenden. Die Klasse `CourseSet` bietet unter anderem Methoden zur Überprüfung, ob ein Studierender zur Anmeldung nach den definierten Regeln des Anmeldesets zugelassen ist, und zur Verwaltung der Sitzplatzverteilung. Es kann der Status und der Zeitpunkt der Sitzplatzverteilung sowie die abgegebenen Prioritäten und die Anzahl eingesehen werden. Das Losverfahren kann ebenfalls über eine Methode gestartet werden. Jedoch ist `CourseSet` stark an die konkreten Klassen der Anmelderegel „Beschränkte Teilnehmendenanzahl“ (`ParticipantRestrictedAdmission`), des Losverfahrens (`RandomAlgorithm`) und der Priorität (`AdmissionPriority`) gekoppelt, sodass eine Erweiterung durch neue Anmelderegeln, Zuteilungsmethoden und Prioritätenerhebung schwierig ist.

Studierende melden sich zu Veranstaltungen an, wenn sie über die Informationsseite der Veranstaltung auf *Zugang zur Veranstaltung* klicken. Dieser Link wird über den Dispatcher an den jeweiligen Controller weitergereicht, welcher folgendes ausführt:

Zunächst wird überprüft, ob der Kurs mit der übergebenen ID in einem Anmeldeset enthalten ist und bereits irgendeine Regel des Anmeldesets eine direkte Anmeldung ablehnt. Gegebenfalls werden dann Fehlermeldungen oder Aufforderungen zur Passworteingabe ausgegeben. Nach der Eingabe des Passworts wird wieder geprüft, ob es mindestens eine ablehnende Anmelderegel gibt.

Gibt es keine Anmelderegel, die eine direkte Anmeldung blockiert, wird im nächsten Schritt überprüft, ob für das Anmeldeset eine Sitzplatzverteilung aktiv ist. Falls die Verteilung bereits stattgefunden hat und die Veranstaltung noch freie Plätze hat, wird der Studierende als Teilnehmer eingetragen, andererseits erhält der Studierende einen Platz auf der Warteliste. Die Anmeldung wird abgelehnt, wenn es keine Warteliste gibt oder wenn die Warteliste keine weiteren Listenplätze vorweist.

Für den Fall, dass die Sitzplatzverteilung noch nicht stattgefunden hat, wird im Speziellen auf die Anmelderegel „Anmeldung zu maximal n Veranstaltungen“ geprüft, um die Maske für die Prioritätenabgabe zu laden. Andernfalls kann nur zur Platzverteilung an- beziehungsweise abgemeldet werden.

Wenn es keine blockierende Anmelderegel gibt und keine Sitzplatzverteilung aktiv ist, werden sich anmeldende Studierende in die Veranstaltung eingetragen. Die Eintragung ist je nach konfiguriertem Anmeldemodus durch Lehrende vorläufig oder direkt.

4.3 Implementierung des Plugins

Das Kernsystem Stud.IP soll nun um die Fähigkeit der überschneidungsfreien präferenzbasierten Veranstaltungszuteilung erweitert werden. Es wurde die Anforderung gestellt, dass am Quellcode des Kernsystems möglichst keine Änderungen durchgeführt werden sollen. Die Erweiterung der gewünschten Funktionalität wird vollständig durch ein Plugin realisiert.

Die Aufgabe des Plugins ist es, eine neue Anmelderegel einzuführen, sodass das gewünschte Anmeldeverfahren in Stud.IP durchgeführt werden kann. Es sollen den Lehrenden beziehungsweise den Studierenden über neue Seiten ermöglicht werden, die Anmelderegel zu konfigurieren respektive die Präferenzen zu den Übungen einzureichen. Die eingegebenen Daten müssen persistent gespeichert werden, dazu werden die notwendigen Datenbanktabellen angelegt. REST-API-Routen werden für die Kommunikation zwischen Microservice und Stud.IP eingeführt und abschließend Cronjob, welcher periodisch überprüft, ob ausstehende Berechnungen vorhanden sind und gegebenfalls eine Anfrage an das Microservice sendet oder die Ergebnisse einholt.

4.3.1 Anmelderegel

Anmeldesets beschreiben ein gemeinsames Anmeldeverfahren für alle Veranstaltungen innerhalb des Sets. Das heißt, dass alle Regeln gleichermaßen für die Veranstaltungen des Anmeldesets gelten. Im Falle der Übungsgruppenzuteilung am Institut für Informatik muss eine Gruppierung von Veranstaltungen innerhalb des Anmeldesets ermöglicht werden. Obwohl diese Veranstaltungen am gleichen Anmeldeverfahren mit gleichem Verteilzeitpunkt teilnehmen, werden Prioritäten nicht zu allen Veranstaltungen des Anmeldesets abgegeben, sondern nur zu Übungen innerhalb einer Lehrveranstaltung. Es wird für jede Lehrveranstaltung nur eine Übung zugeteilt, wenn Studierende am Übungsbetrieb dieser Lehrveranstaltung teilnehmen möchten. Weiterhin kann für jede Lehrveranstaltung eine andere Mindestanzahl von abzugebenden Prioritäten gefordert werden.

Es gibt eine Menge von Veranstaltungen, die dem Anmeldeset zugeordnet werden. Diese Veranstaltungen entsprechen den verschiedenen Übungsgruppen aller teilnehmenden Lehrveranstal-

tungen. Parallel stattfindende Übungsgruppen können als Bündelelement zusammengefasst werden, sodass nur zum Übungstermin Prioritäten abgegeben werden. Übungsgruppen derselben Lehrveranstaltung werden in eine Zuteilungsgruppe gruppiert.

In anderen Anwendungsfällen kann eine Gruppierung von Veranstaltungen anders aussehen. Allgemein werden Prioritäten nur innerhalb einer Zuteilungsgruppe abgegeben und dem Studierenden wird nur eine Veranstaltung dieser Gruppe zugeteilt. Für die Prioritätenerhebung können mehrere Veranstaltungen zusammengefasst werden, sodass für diese zusammengefassten Veranstaltungen dieselbe Priorität gilt.

Das Plugin führt die neue Klasse `BundleAllocationAdmission` für die Anmelderegel „Präferenzbasierte überschneidungsfreie Anmeldung“ ein, welche von der Oberklasse `AdmissionRule` erbt. Instanzen dieser Anmelderegel werden in der Datenbanktabelle `bpsadmissions` gespeichert. `BundleAllocationAdmission` enthält als Attribute den Verteilzeitpunkt, die Job-ID sowie die Information darüber, ob die Verteilung abgeschlossen ist. Der Anmelderegel sind in der Datenbank ihre Zuteilungsgruppen `bps_rankinggroup` zugeordnet, die wiederum Bündelelemente `bps_bundleitem` enthalten. Ein Bündelelement umfasst ein oder mehrere Veranstaltungen (`bps_bundleitem_course`). Abgegebene Prioritäten zu Bündelelemente der Studierenden werden in `bps_bundleitem_ranking` gespeichert und vorläufige Zuteilungsergebnisse, die vom Microservice berechnet wurden, werden in `bps_prelim_alloc`.

Innerhalb der Klasse `BundleAllocationAdmission` müssen die zwei notwendigen HTML-Vorlagen spezifiziert werden. Zum einen soll die Konfigurationsoberfläche der Anmelderegel für Lehrende bereitgestellt werden und zum anderen der beschreibende Text des Anmeldeverfahrens.

Die Konfigurationsoberfläche (Abbildung 7) ermöglicht es den Lehrenden den Verteilzeitpunkt und die maximale Teilnehmendenanzahl der jeweiligen Veranstaltungen festzulegen, Zuteilungsgruppen zu erstellen und die Veranstaltungen den Gruppen zuzuordnen sowie Veranstaltungen zu einem Bündelelement zusammenzufassen.

Die HTML-Vorlage zur Beschreibung der Anmelderegel wird verwendet, um auf der Informationsseite einer Veranstaltung und auf der Bearbeitungsseite eines Anmeldesets das eingestellte Anmeldeverfahren zu erklären (Abbildung 8). Bei der Ausgabe der Vorlage wird überprüft, ob der momentane Benutzer die notwendigen Rechte eines Dozierenden beziehungsweise eines Administrators hat und sich auf der Bearbeitungsseite des Anmeldesets befindet, um den Button zur Verwaltung von Anmeldungen darzustellen. Dieser Button leitet auf eine Seite (Abbildung 9), die es Lehrenden und Administratoren ermöglicht, die abgegebenen Prioritäten und das Zuteilungsergebnis einzusehen sowie bei Bedarf Zuteilungen von Studierenden manuell zu ändern. Alle hier zu sehenden Informationen können ebenfalls als CSV-Datei für Tabellenkalkulationsprogramme exportiert werden. Weiterhin soll eine neue Berechnung der Sitzplatzverteilung angefordert werden können, falls sich Änderungen bezüglich der Kapazitäten ergeben haben. Ist das Zuteilungsergebnis für alle beteiligten Personen zufriedenstellend, kann das Ergebnis bestätigt werden. Dabei werden die Studierenden entsprechend in die Veranstaltungen beziehungsweise Wartelisten eingetragen und über Stud.IP-Nachrichten mit E-Mail-Weiterleitung benachrichtigt.

Bei einer Sitzplatzverteilung über das Stud.IP-eigene Losverfahren wird dem Benutzer bei der Bearbeitung des Anmeldesets zwar ebenfalls ermöglicht, unter anderem die maximale Teilnehmendenanzahl festzulegen oder die Liste der Anmeldungen einzusehen, jedoch werden diese Buttons nur dann angezeigt, wenn dem Anmeldeset im Speziellen die Regel „Beschränkte Teilnehmendenanzahl“ hinzugefügt wurde. Um diese Einschränkung zu umgehen, wurde die Vorlage zur Beschreibung der Anmelderegel um diesen Button erweitert.

Ein ähnliches Problem besteht bei der Prioritätenerhebung: Bei der Anmeldung zur Veranstaltung wird ebenfalls auf die Anmelderegeln „Anmeldung zu maximal n Veranstaltungen“ und „Beschränkte Teilnehmendenanzahl“ geprüft, um die Maske zur Prioritätenerhebung zu laden. Die Maske wird nicht durch die Anmelderegel bereitgestellt, sondern ist Teil der Komponente,

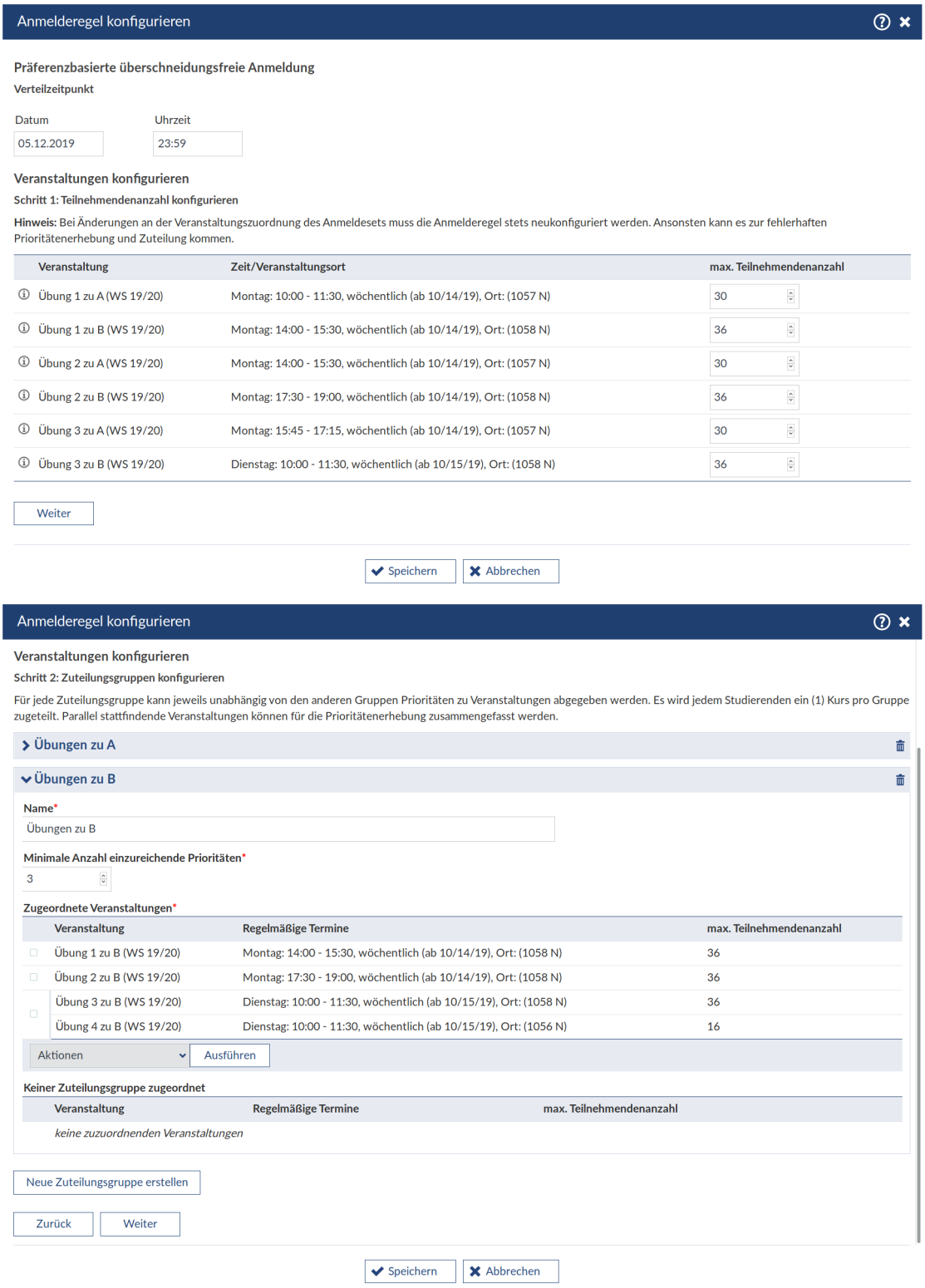


Abbildung 7: Konfigurationsoberfläche der Anmelderegel

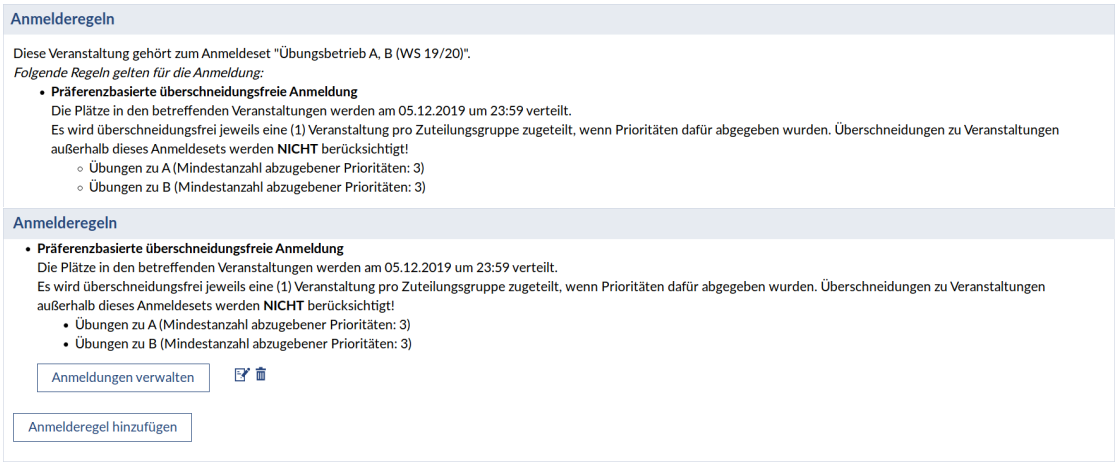


Abbildung 8: Beschreibung der Anmelderegel auf der Informationsseite einer Veranstaltung (oben) sowie auf der Bearbeitungsseite des Anmeldesets (unten)

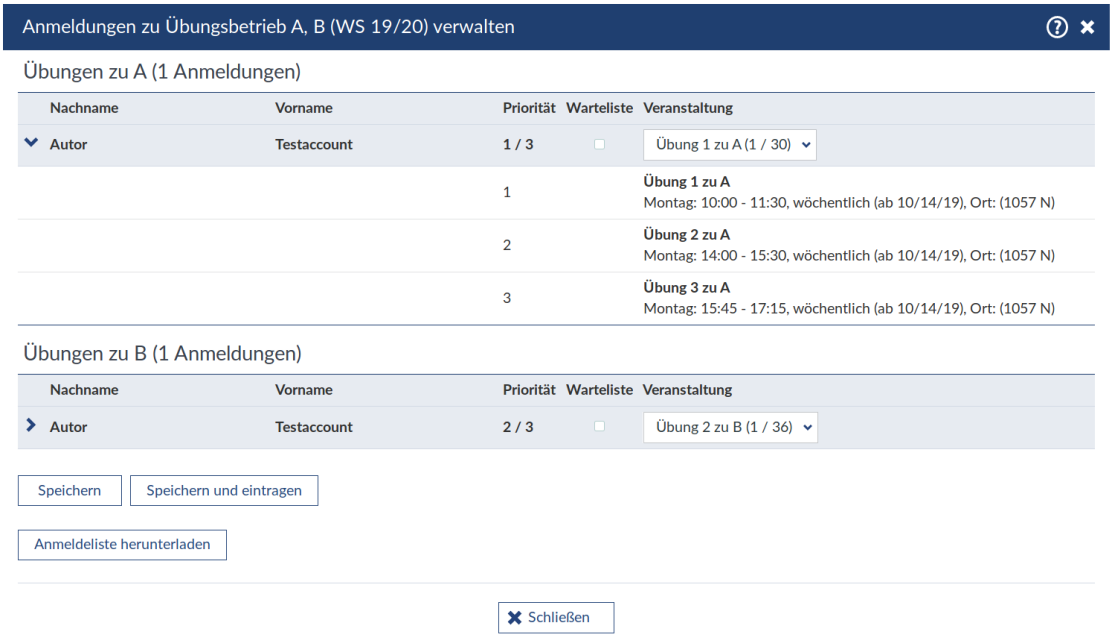


Abbildung 9: Verwaltungsseite für Lehrende und Administratoren über abgegebene Prioritäten und vorläufige Zuteilungsergebnisse

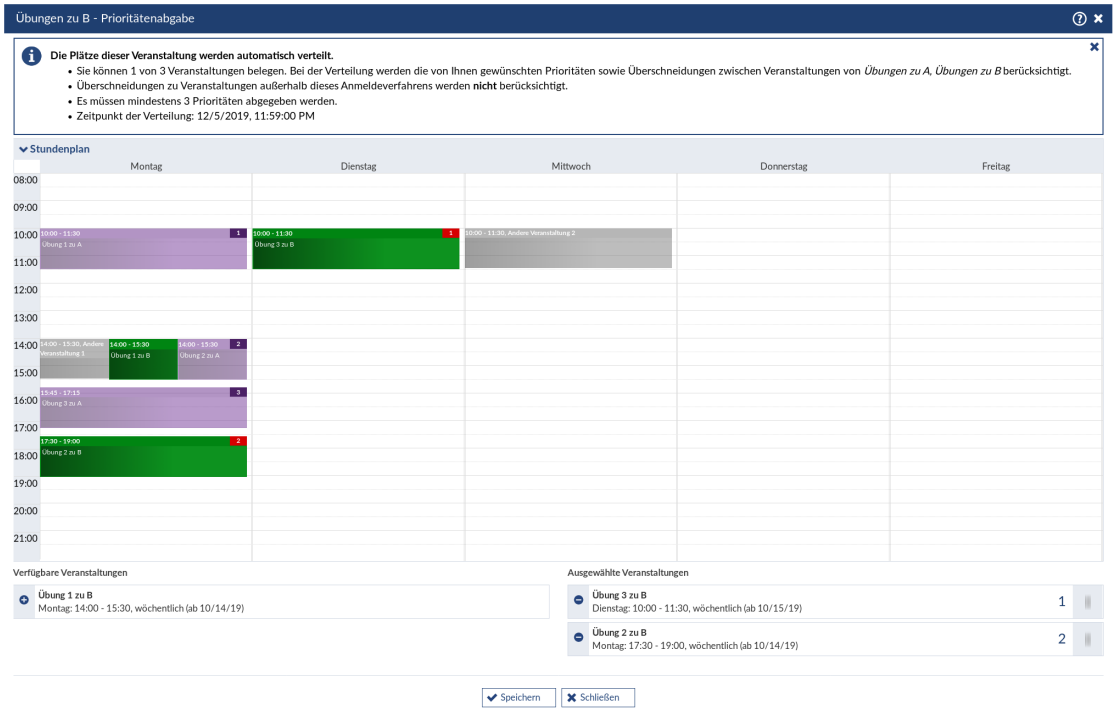


Abbildung 10: Präferenzerhebung durch Studierende

die die Anmeldung durchführt. Diese Komponente hat eine starke Abhängigkeit zu den speziellen Klassen des Losverfahrens. Es ist daher nicht möglich, dass eigene Anmelderegeln eine andere Art von Prioritätenerhebung durchführen.

Hierfür wurde eine Workaround-Lösung entwickelt, die die Einschränkungen möglichst ohne unerwartete Änderungen in der Benutzererfahrung umgeht: Zunächst werden Anmeldungen über den Stud.IP-eigenen Mechanismus gesperrt. Die Anmelderegel agiert hier ähnlich wie „Anmeldung gesperrt (global)“. Das Interface `SystemPlugin` dieses Plugins wurde so implementiert, dass nur Informationsseiten von Veranstaltungen modifiziert werden. Der Link *Zugang zur Veranstaltung* wird ausgetauscht, wenn der Kurs in einem Anmeldeset enthalten ist, welches eine überschneidungsfreie präferenzbasierte Anmeldung vorsieht. Die Benutzer werden nicht zur nativen Anmeldekomponente geleitet, sondern auf die Anmeldeseite, die vom Plugin bereitgestellt wird (Abbildung 10). Die Pluginseite kann nun auf ihre eigenen Datenstrukturen zugreifen und stellt den momentanen Stundenplan mit bereits eingetragenen Veranstaltungen und gegebenenfalls bereits abgebene Prioritäten anderer Zuteilungsgruppen des gleichen Anmeldesets dar. Weiter werden die Bündelelemente der Zuteilungsgruppe der momentan geöffneten Veranstaltung geladen, sodass Prioritäten über diese Elemente abgegeben werden kann. Falls bereits Prioritäten eingereicht wurden, wird der zuletzt gespeicherte Zustand angezeigt. Die Prioritäten der Studierenden zu den jeweiligen Bündelelementen der Zuteilungsgruppe werden in der Tabelle `bps_bundleitem_ranking` gespeichert.

Studierende sehen auf der Seite „Meine Veranstaltung“ eine Übersicht über ihre Wartelistenplätze, ihre vorläufigen Anmeldungen und ihre abgegebenen Prioritäten zu Veranstaltungen im Losverfahren. Auch bei der überschneidungfreien präferenzbasierten Anmeldung zu Veranstaltungen sollen Studierende einen Überblick über ihre eingereichten Prioritäten bekom-

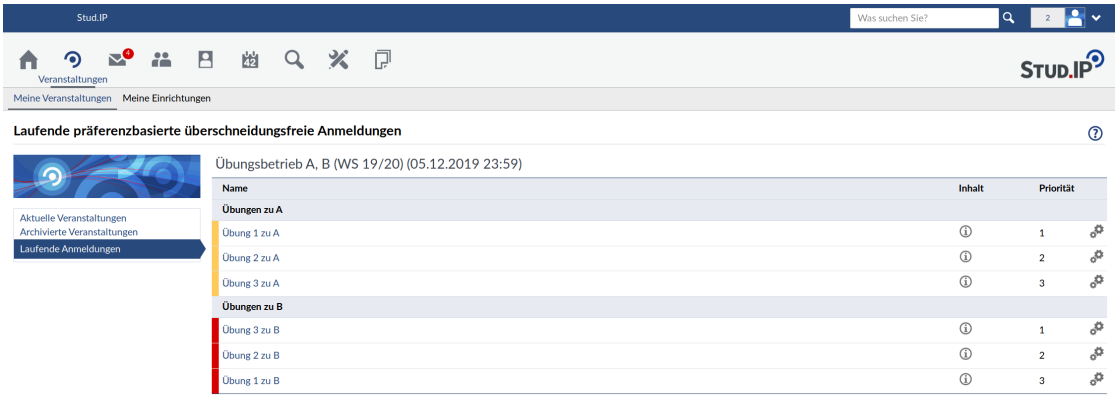


Abbildung 11: Übersichtsseite für Studierende über ihre abgegebene Prioritäten

men. Bei der Stud.IP-nativen Übersicht besteht wieder eine starke Abhängigkeit zur Klasse `AdmissionPriority` und der Anmelderegel „Beschränkte Teilnehmendenanzahl“. Daher ist es nicht möglich von seiten des Plugins Einträge der Übersicht hinzuzufügen. Das Plugin stellt eine eigene Übersichtsseite (Abbildung 11) bereit, die über Navigationspunkt „Laufende Anmeldungen“ auf „Meine Veranstaltungen“ erreichbar ist.

4.3.2 Kommunikation zwischen Stud.IP und Microservice

Wie in Kapitel 3.4.2 erwähnt, soll die Berechnung einer Zuteilung mithilfe des Bundled Probabilistic Serial Mechanimus außerhalb des Stud.IP-Systems geschehen. Dazu müssen von seiten Stud.IP Endpunkte eingerichtet werden, sodass über die REST-API mit dem Microservice kommuniziert werden kann. Einerseits müssen die benötigten Informationen zu den Prioritäten der Studierenden, zu den Zuteilungsgruppen und den zusammengefassten Veranstaltungen sowie zu Überschneidungen unter den Veranstaltungen vom Microservice angefordert werden können. Andererseits benötigt es eine Schnittstelle, welche es ermöglicht, dass die berechneten Zuteilungsergebnisse wieder zurückübermittelt werden.

Das Plugin implementiert das Interface `RESTAPIPlugin` und stellt eine entsprechende `RouteMap`-Klasse bereit. In dieser Klasse sind die Endpunkte folgendermaßen definiert:

- `GET /bundleallocation/courseset/{set_id}/preferences`: Über eine `GET`-Anfrage an diese Ressource können gespeicherte Informationen zur Sitzplatzverteilung des Anmeldesets `set_id` angefordert werden. Es werden Zuteilungsgruppen, Bündelelemente und ihre Veranstaltungs-IDs sowie Informationen über Überschneidungen und pseudonymisierte Prioritäten der Studierenden im JSON-Format übermittelt. Die Hashtabelle bezüglich Überschneidungen unter Veranstaltungen wird hier erzeugt.
- `POST /bundleallocation/courseset/{set_id}/allocations`: Die Ergebnisse der Berechnung können Stud.IP über diesen Endpunkt übermittelt werden. Die Ergebnisse werden in die Datenbanktabelle `bps_prelim_alloc` gespeichert, sodass sie später von Lehrenden und Administratoren geprüft werden können. Weiter wird der Besitzer der Anmeldesets über eine Systemnachricht benachrichtigt, dass die Ergebnisse eingesehen werden können. Abschließend wird die Zuteilung als abgeschlossen markiert.

Zugriff auf REST-API-Endpunkte wird im Digicampus durch einen speziellen Tokenmechanismus geschützt. Dieser Token wird jedoch nicht über den `Authorization`-Header übergeben,

sondern über einen Digicampus-eigenen Header.

Cronjobs sind wiederkehrende Aufgaben, die zeitbasiert ausgeführt werden sollen. Stud.IP setzt unter anderem Cronjobs ein, um regelmäßig zu prüfen, ob es Anmeldesets mit ausstehenden Losverfahren gibt. Falls solche Anmeldesets gefunden wurden, wird für jedes Anmeldeset das Losverfahren ausgeführt. Studierende werden anschließend in die Veranstaltungen eingetragen und benachrichtigt, die Prioritäten werden aus der Datenbank gelöscht sowie das Losverfahren im Anmeldeset als abgeschlossen markiert.

Das Plugin registriert im Stud.IP-System einen neuen Cronjob, der halbstündig ausgeführt wird. Wie im obigen Cronjob, werden Anmeldesets gesucht, die die Anmelderegel „Präferenzbasierte und überschneidungsfreie Anmeldung“ enthalten und dessen Zuteilung noch nicht abgeschlossen ist. Falls der Verteilzeitpunkt in der Vergangenheit liegt und keine Job-ID hinterlegt ist, wird eine neue Berechnungsanfrage dem Microservice übermittelt. Ist die Job-ID bekannt, wird der Status der Berechnung und bei abgeschlossener Berechnung das Zuteilungsergebnis angefragt. Dies dient der Absicherung für den Fall, dass die Rückübermittlung des Ergebnis nicht erfolgreich gewesen ist. Das Ergebnis wird wieder gespeichert und der Besitzer des Anmeldesets benachrichtigt.

5 Fallbeispiel

Anhand von zwei anonymisierten Datensätzen vom Sommersemester 2018 und Wintersemester 2019/2020 sollen Zuteilungen mithilfe des Bundled Probabilistic Serial Mechanismus berechnet und evaluiert werden. Zunächst werden die Datensätze und die erhobenen Prioritäten kurz beschrieben und im weiteren Verlauf die Ergebnisse der Zuteilung diskutiert.

5.1 Datensätze

Der erste Datensatz stammt aus dem Vorlesungsverwaltungssystem des Lehrstuhls von Prof. Dr. Hagerup. Studierende haben im Sommersemesters 2018 Prioritäten zu den Lehrveranstaltungen *Informatik 2*, *Systemnahe Informatik* und *Theoretische Informatik* abgegeben. Dieser Datensatz wurde zum Testen des Mechanismus verwendet. Im Sommersemester 2018 wollten 659 Studierende am Übungsbetrieb dieser drei Lehrveranstaltungen teilnehmen. Dabei haben 437 Studierende Prioritäten zu Übungen der Lehrveranstaltung *Informatik 2* abgegeben, 325 Studierende zu *Theoretische Informatik* sowie 200 Studierende zu *Systemnahe Informatik*. Etwa 40% der Studierenden (265) besuchen mindestens zwei Lehrveranstaltungen und erfordern daher eine überschneidungsfreie Zuteilung. Die folgende Tabelle zeigt die Kapazitäten der angebotenen Übungstermine auf:

Informatik 2		**Systemnahe Informatik**		**Theoretische Informatik**	
Mo 08:15 - 09:45 Uhr	30	Mo 10:00 - 11:30 Uhr	30	Mo 08:15 - 09:45 Uhr	24
Mo 14:00 - 15:30 Uhr	18	Di 10:00 - 11:30 Uhr	30	Mo 14:00 - 15:30 Uhr	30
Mo 15:45 - 17:15 Uhr	30	Di 14:00 - 15:30 Uhr	36	Mo 17:30 - 19:00 Uhr	24
Mo 17:30 - 19:00 Uhr	24	Mi 17:30 - 19:00 Uhr	36	Di 12:15 - 13:45 Uhr	30
Mo 17:30 - 19:00 Uhr	30	Do 10:00 - 11:30 Uhr	30	Mi 12:15 - 13:45 Uhr	24
Di 12:15 - 13:45 Uhr	24	Fr 12:15 - 13:45 Uhr	30	Mi 14:00 - 15:30 Uhr	24
Di 17:30 - 19:00 Uhr	24	Mo 14:00 - 15:30 Uhr	36	Mi 15:45 - 17:15 Uhr	24
Di 17:30 - 19:00 Uhr	30			Do 08:15 - 09:45 Uhr	36
Mi 08:15 - 09:45 Uhr	24			Do 10:00 - 11:30 Uhr	24
Mi 08:15 - 09:45 Uhr	24			Do 12:15 - 13:45 Uhr	30
Mi 10:00 - 11:30 Uhr	24			Do 17:30 - 19:00 Uhr	24
Do 08:15 - 09:45 Uhr	18			Fr 08:15 - 09:45 Uhr	36
Do 15:45 - 17:15 Uhr	24			Fr 10:00 - 11:30 Uhr	24
Do 17:30 - 19:00 Uhr	24				
Do 17:30 - 19:00 Uhr	30				
Fr 08:15 - 09:45 Uhr	30				
Fr 10:00 - 11:30 Uhr	24				
Fr 12:15 - 13:45 Uhr	24				
Fr 12:15 - 13:45 Uhr	18				
Fr 14:00 - 15:30 Uhr	24				

Tabelle 3: Übersicht über Kapazitäten der jeweiligen Übungstermine des Sommersemesters 2018

Abbildung 12 zeigt den Stundenplan mit den angebotenen Übungsterminen des Sommersemesters und ihre durchschnittliche erhaltene Priorität. Übungstermine von *Informatik 2* werden rot dargestellt, Übungen der *Systemnahen Informatik* sind lila und Übungen der *Theoretischen Informatik* sind blau. Es ist leicht zu sehen, dass einige Übungstermine verschiedener Lehrveranstaltungen zur gleichen Zeit stattfinden. Eine stärkere Farbsättigung signalisiert eine niedrigere durchschnittliche Priorität und zeigt Tendenzen zum Beliebtheitsgrad eines Übungstermins auf.

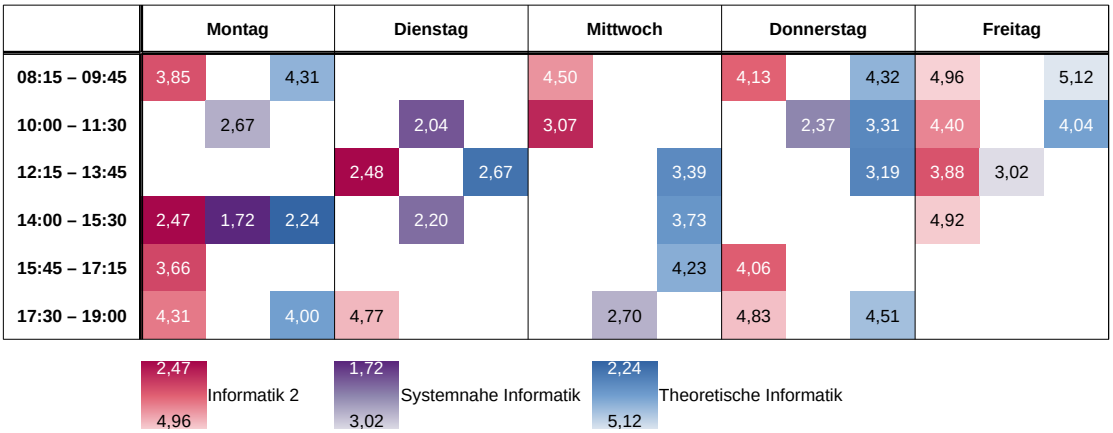


Abbildung 12: Durchschnittliche Priorität zu Übungsterminen der Lehrveranstaltungen aus dem Sommersemester 2018

Es ist zu beachten, dass der Durchschnittswert nur innerhalb der gleichen Lehrveranstaltung vergleichbar ist, da unterschiedlich viele Studierende am Übungsbetrieb der jeweiligen Lehrveranstaltungen teilnehmen. Auffällig ist, dass Übungstermine am Montag oder am Dienstag in der Mittagszeit besonders beliebt sind. Termine am Freitag werden im Allgemeinen schlecht besucht, insbesondere wenn sie in der Früh sind. Abendtermine sind ebenfalls eher unbeliebt bei den Studierenden.

Für das Wintersemester 2019/2020 wurden für die Lehrveranstaltungen *Informatik 1* sowie *Diskrete Strukturen und Logik* Studierende mithilfe des Bundled Probabilistic Serial Mechanismus ihren Übungsgruppen zugeteilt. Hierbei haben insgesamt 629 Studierende Prioritäten abgegeben, davon besuchen 200 Studierende beide Lehrveranstaltungen und erfordern eine überschneidungsfreie Zuteilung. Es gibt 539 Teilnehmende in *Informatik 1* sowie 290 Teilnehmende in *Diskrete Strukturen und Logik*. Die Kapazitäten der Übungstermine in *Informatik 1* richten sich nach der Anzahl von Sitzplätzen der gebuchten Räume. Für *Diskrete Strukturen und Logik* reichte dies jedoch nicht aus, daher wurden alle Termine außer Montag, 17:30 Uhr und Mittwoch, 10:00 Uhr um 4 Plätze überbucht. Die Kapazität des Termins am Mittwoch, 10:00 Uhr wurde um 6 vergrößert. In Tabelle 4 kann eine Übersicht über die verfügbaren Übungen und ihre Kapazitäten entnommen werden.

Die Präferenzerhebung erfolgte zwar über Digicampus, jedoch nicht unter Verwendung des entwickelten Plugins. Hierzu wurde die Präferenzerhebung des Digicampus genutzt. Die Losverfahren wurden vorzeitig händisch gestoppt und die Prioritäten mehrerer Anmeldesets exportiert, sodass eine Zuteilung mithilfe des BPS-Mechanismus geschehen kann. Erzeugte Teilnehmendenlisten wurden wieder manuell von den Lehrenden in die jeweiligen Veranstaltungen importiert. Zu diesem Datensatz wurde ebenfalls eine Analyse über die Beliebtheit von Übungsterminen durchgeführt. Abbildung 13 zeigt ähnlich zum Sommersemester 2018 eine starke Tendenz zu Übungsterminen um 10:00 Uhr beziehungsweise um 12:15 Uhr in der ersten Hälfte der Woche. Es scheint, dass Studierende lieber Übungen in der Früh im Gegensatz zu Übungen um 17:30 Uhr besuchen würden. Obwohl im Allgemeinen Termine am Freitag von Studierenden vermieden werden, ist die Übung zu *Diskrete Strukturen und Logik* am Freitag um 10:00 Uhr die zweitbeliebteste.

Informatik 1		**Diskrete Strukturen und Logik**	
Mo 08:15 – 09:45 Uhr	54	Mo 14:00 – 15:30 Uhr	28
Mo 12:15 – 13:45 Uhr	30	Mo 17:30 – 19:00 Uhr	120
Mo 14:00 – 15:30 Uhr	30	Di 15:45 – 17:15 Uhr	28
Mo 17:30 – 19:00 Uhr	54	Di 17:30 – 19:00 Uhr	28
Di 08:15 – 09:45 Uhr	54	Mi 10:00 – 11:30 Uhr	42
Di 12:15 – 13:45 Uhr	24	Do 12:15 – 13:45 Uhr	28
Di 14:00 – 15:30 Uhr	78	Fr 10:00 – 11:30 Uhr	28
Di 15:45 – 17:15 Uhr	30		
Di 17:30 – 19:00 Uhr	54		
Mi 08:15 – 09:45 Uhr	24		
Mi 10:00 – 11:30 Uhr	30		
Do 12:15 – 13:45 Uhr	24		
Do 17:30 – 19:00 Uhr	30		
Fr 08:15 – 09:45 Uhr	30		
Fr 10:00 – 11:30 Uhr	30		
Fr 14:00 – 15:30 Uhr	30		

Tabelle 4: Übersicht über Kapazitäten der jeweiligen Übungstermine des Wintersemesters 2019/2020

5.2 Ergebnisse

Für die Berechnung einer Zuteilung müssen zunächst die Präferenzordnungen der Studierenden über Veranstaltungsbündel erzeugt werden. Sowohl für das Sommersemester 2018 als auch für das Wintersemester 2019/2020 dauerte das Erzeugen und Sortieren der Veranstaltungsbündel für 659 beziehungsweise 629 Studierende nur etwa 150 bis 170 ms. Die Berechnung einer zufälligen Zuteilung nach Algorithmus 2 war ebenfalls relativ zügig mit ungefähr 340 ms beziehungsweise 370 ms. Einzig der Lotterie-Algorithmus brauchte aufgrund des Lösens mehrerer linearer und quadratischer Optimierungsprobleme länger. Die Zerlegung der zufälligen Zuteilung in eine Wahrscheinlichkeitsverteilung über deterministische Zuteilungen benötigte etwa 1,5 Minuten. Dabei wurde für das Sommersemester 2018 $\epsilon = 1$ und $\eta = 5$ sowie für das Wintersemester 2019/2020 $\epsilon = 0.5$ und $\eta = 3$ gewählt. Insgesamt wurde gezeigt, dass Bundled Probabilistic Serial von seiten der Berechnungsdauer ein praktikabler Mechanismus für die Übungsgruppenzuteilung ist.

Es wird nun der erwartete Rang und die erwartete Größe der zufälligen Zuteilung betrachtet. Unter Rang versteht man die Position eines Veranstaltungsbündels innerhalb der Präferenzordnung und die Größe beschreibt die Anzahl zugeteilter Studierende. Im Sommersemester 2018 beträgt der erwartete Rang 3.463 bei einer erwarteten Größe von 647.522. Das bedeutet, dass etwa 647 Studierende mit einer Zuteilung des dritten beziehungsweise vierten Bündel in ihrer Präferenzordnung rechnen können. Einige Studierende können bei einer zufälligen Zuteilung natürlich Pech haben. In der deterministischen Zuteilung entspricht die Größe jedoch der Anzahl aller teilnehmenden Studierenden, sofern die Kapazitäten dafür ausreichen. Insgesamt erhalten um die 93,474% der Studierende im Sommersemester 2018 ein Bündel aus den ersten 10 der Präferenzordnung. Für das Wintersemester beträgt der erwartete Rang 5.884 und die erwartete Größe 617.744. Der schlechtere Rang im Wintersemester 2019/2020 gegenüber des Rangs im Sommersemester 2018 lässt sich dadurch erklären, dass bezüglich den Übungen der *Diskreten Strukturen und Logik* eine Knappheit von Plätzen herrschte. Ein Großteil der angebotenen Plätze wird durch eine Übung am Montag, um 17:30 Uhr abgedeckt, die eher unbeliebt ist. Im Wintersemester 2019/2020 erhalten daher nur 78,373% ein Bündel aus den Top 10 ihrer Präfe-

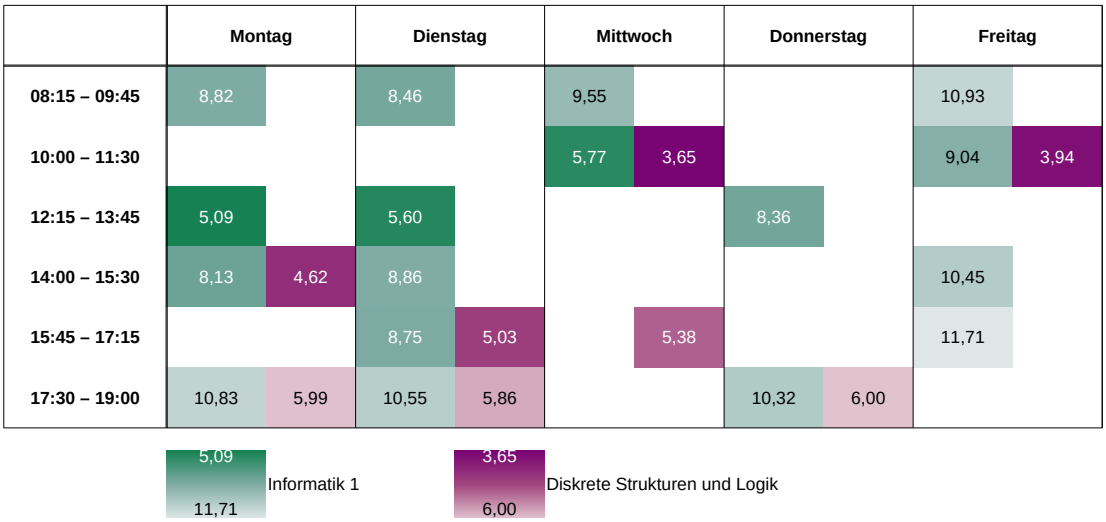


Abbildung 13: Durchschnittliche Priorität zu Übungsterminen der Lehrveranstaltungen aus dem Wintersemester 2019/2020

renzordnung.

Die Präferenzordnung über Veranstaltungsbündel eines Studierenden wurde über Prioritätenlisten ein oder mehrer Lehrveranstaltungen gebildet. Es ist daher nützlich nicht nur das zu erwartende Bündel zu betrachten, sondern auch die Wahrscheinlichkeitsverteilungen über die Prioritäten der Lehrveranstaltungen. Tabellen 5 und 6 zeigen mit welcher Wahrscheinlichkeit Studierende ihre Priorität in der Lehrveranstaltung erhalten. Mit 81,21% wird dem Studierenden eine seiner drei meist präferiertesten Übungen zu *Informatik 2* zugeteilt. In der *Systemnahen Informatik* und *Theoretischen Informatik* sind es sogar 97,59% beziehungsweise 85,48%. Im Wintersemester 2019/2020 liegt die Wahrscheinlichkeit, eine seiner drei meist präferierten Übungen zu *Informatik 1* zu erhalten, bei 75,82%. Bei Übungen zu *Diskrete Strukturen und Logik* beträgt die Wahrscheinlichkeit nur bei 65,76%.

Obwohl die Chance besteht eine „schlechte“ oder gar seine letzte Priorität zu erhalten, muss beachtet werden, dass zufällige Zuteilungen wiederum in Wahrscheinlichkeitsverteilungen deterministischer Zuteilungen zerlegt werden. Die schlechteste Priorität der ausgewählten deterministischen Zuteilung für *Informatik 1* und *Diskrete Strukturen und Logik* im Wintersemester 2019/2020 war die 7. Priorität. Insbesondere wird mithilfe des Iterativen Rundungsverfahrens zunächst die Nachfrage der Studierenden erfüllt, wobei die Übungen um maximal $k - 1$ Sitzplätze überbelegt werden können. Der Lotterie-Algorithmus hat für die zufällige Zuteilung des Sommersemesters 2018 eine Zerlegung in 33 deterministische Zuteilungen gefunden. Für das Wintersemester 2019/2020 wurde die zufällige Zuteilung durch eine Linearkombination von 35 deterministischen Zuteilungen angenähert.

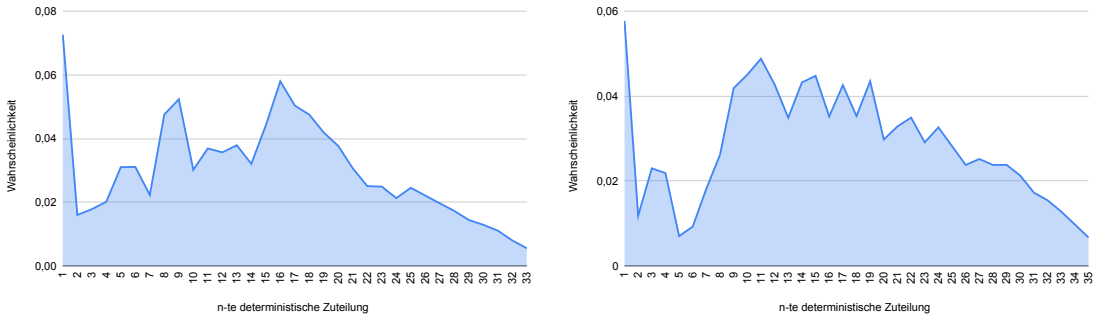
Abbildung 14 zeigt die Wahrscheinlichkeitsverteilung der zerlegten zufälligen Zuteilung des Sommersemesters 2018 sowie des Wintersemesters 2019/2020. Der Lotterie-Algorithmus fügt schrittweise deterministische Zuteilungen der Lösungsmenge hinzu. Der Koeffizient der integralen Lösung in der Konvexkombination entspricht der Wahrscheinlichkeit, dass diese deterministische Lösung ausgewählt wird. Interessanterweise hat die erste Zuteilung der Lösungsmenge die höchste Wahrscheinlichkeit und die zweite Zuteilung eine recht niedrige Wahrscheinlichkeit. Dies liegt daran, dass in der Gegenrichtung der ersten „guten“ deterministischen Zuteilung eine integrale

Informatik 2		**Systemnahe Informatik**		**Theoretische Informatik**	
Priorität	*Wahrscheinlichkeit*	*Priorität*	*Wahrscheinlichkeit*	*Priorität*	*Wahrscheinlichkeit*
1	0,4760	1	0,7626	1	0,5272
2	0,1991	2	0,1828	2	0,2206
3	0,1370	3	0,0305	3	0,1070
4	0,1060	4	0,0056	4	0,0628
5	0,0429	5	0,0002	5	0,0327
6	0,0204			6	0,0242
7	0,0019			7	0,0014
				8	0,0001
				9	0,0002
				10	0,0001

Tabelle 5: Wahrscheinlichkeitsverteilung der Prioritäten der Lehrveranstaltungen im Sommersemester 2018

Informatik 1		**Diskrete Strukturen und Logik**	
Priorität	*Wahrscheinlichkeit*	*Priorität*	*Wahrscheinlichkeit*
1	0,4667	1	0,4621
2	0,1782	2	0,1302
3	0,1133	3	0,0635
4	0,0924	4	0,0909
5	0,0520	5	0,0983
6	0,0361	6	0,0774
7	0,0180	7	0,0509
8	0,0100		
9	0,0099		
10	0,0052		
11	0,0030		
12	0,0016		
13	0,0006		
14	0,0001		

Tabelle 6: Wahrscheinlichkeitsverteilung der Prioritäten der Lehrveranstaltungen im Wintersemester 2019/2020



(a) Sommersemester 2018

(b) Wintersemester 2019/2020

Abbildung 14: Wahrscheinlichkeitsverteilung deterministischer Zuteilungen der Annäherung der zufälligen Zuteilung

Lösung gesucht wird. Anschließend steigt die Wahrscheinlichkeit wieder an. Die deterministischen Zuteilungen, die gegen Ende der Lösungsmenge hinzugefügt wurde, haben eine vergleichsweise niedrige Wahrscheinlichkeit, da sie zur Annäherung der zufälligen Zuteilung nur wenig beitragen.

Insgesamt lässt sich zur „automatischen“ Überbelegung von Übungen im Iterativen Rundungsverfahren sagen, dass im Allgemeinen die Kapazitäten der Übungen eingehalten werden. Beim Lotterie-Algorithmus treten in 9 von 32 möglichen deterministischen Zuteilungen im Sommersemester 2018 sowie in 8 von 35 deterministischen Zuteilungen Überbelegungen auf. Es wurde bis auf eine Zuteilung stets um einen Sitzplatz erweitert, was für die Durchführung der Übung in der Regel kein Problem darstellt. Das Iterative Verfahren erlaubt eine Lockerung der Kapazität um maximal $k - 1$ Sitzplätze. Im Sommersemester 2018 sind das zwei Plätze und im Wintersemester 2019/2020 ist das nur ein Platz. Es ist aufgefallen, dass bei Zuteilungen mit Überbelegungen meist mehrere Übungen gleichzeitig betroffen sind. In einer Zuteilung des Sommersemesters 2018 werden insgesamt acht Übungen überbelegt, meistens sind es jedoch nur drei bis fünf.

6 Zusammenfassung und Ausblick

Diese Bachelorarbeit stellte den Bundled Probabilistic Serial Mechanismus vor, welcher eine ordinal effiziente, neidfreie und gegen strategisches Handeln geschwach geschützte Zuteilung von mehreren Objekten als Bündel an Agenten ermöglicht. Der Mechanismus wurde im Speziellen auf die überschneidungsfreie und präferenzbasierte Übungsgruppenzuteilung für das Institut für Informatik an der Universität Augsburg angewendet.

Dabei wurde zunächst das Zuteilungsproblem sowie wünschenswerte Eigenschaften einer Zuteilung erläutert. Es wurde das Unmöglichkeitstheorem gezeigt, welches besagt, dass Effizienz und Schutz gegen strategisches Handeln in Konflikt zueinander stehen. Jeder Anwendungsfall muss einen passenden Kompromiss finden.

Der nächste Teil der Bachelorarbeit hat den Bundled Probabilistic Serial Mechanismus im Detail betrachtet. Studierende geben Prioritäten zu Übungen innerhalb einer Lehrveranstaltung ab, jedoch nicht zu Veranstaltungsbündel. Daher müssen zunächst Veranstaltungsbündel erzeugt werden, die überschneidungsfrei sind. Weiterhin müssen die Bündel in eine Reihung gebracht werden, die im Wesentlichen den Prioritäten der Studierenden entspricht. Anhand den Reihungen der Studierenden wird eine zufällige Zuteilung erzeugt, die die Wahrscheinlichkeiten beschreibt, dass ein Studierender ein bestimmtes Bündel erhält. Mithilfe des Birkhoff-von-Neumann-Theorems kann diese zufällige Zuteilung in eine Wahrscheinlichkeitsverteilung deterministischer Zuteilungen zerlegt werden. Eine deterministische Zuteilung entscheidet, welches Bündel welchem Studierenden zugeordnet wird. Die Problematik von zufälligen Zuteilungen, die am Rand der Nachfrage-Menge liegen, sowie die Lösung durch Skalierung wird erläutert. Mithilfe des Iterativen Rundungsverfahrens können deterministische Zuteilungen gefunden werden. Letztlich wird der Lotterie-Algorithmus vorgestellt, welcher es ermöglicht, eine Wahrscheinlichkeitsverteilung über deterministische Zuteilungen zu finden, die der zufälligen Zuteilung ungefähr entspricht.

Nachdem der Mechanismus nun besprochen wurde, liegt der Fokus in der Implementierung und Integration des Bundled Probabilistic Serial Mechanismus in die Lehr- und Lernplattform Digicampus der Universität Augsburg. Die Berechnung geschieht in einem Microservice, welches in Python unter Verwendung von Bibliotheken zur effizienten Vektorrechnung und Solvern für lineare und quadratische Optimierungsprobleme entwickelt wurde. Weiter wird kurz die Integration in Digicampus beschrieben: Es wurde ein Plugin entwickelt, welches das Kernsystem Stud.IP um eine neue Anmelderegel, Prioritätenerhebung und REST-API-Schnittstellen für die Kommunikation mit dem Microservice erweitert.

Abschließend wurde der Bundled Probabilistic Serial Mechanismus auf zwei anonymisierte Datensätze für die Übungsgruppenzuteilung aus dem Sommersemester 2018 sowie aus dem Wintersemester 2019/2020 angewendet. Es wurde ein Überblick über die abgegebenen Prioritäten der Studierenden gegeben und erörtert inwiefern die berechnete mithilfe von BPS berechnete Zuteilung den Wünschen der Studierenden gerecht wurde.

Mit dieser Bachelorarbeit wurde gezeigt, dass der Bundled Probabilistic Serial Mechanismus für die Zuteilung von Studierenden zu ein oder mehreren Übungsterminen am Institut für Informatik der Universität Augsburg gut geeignet ist. Es wurden weiterhin die technischen Voraussetzungen geschaffen für eine Integration des Mechanismus in die Lehr- und Lernplattform Digicampus.

Für die Zukunft gibt es noch Herausforderungen, die gemeistert werden können. Im Moment wird für jede Zuteilungsgruppe nur eine Veranstaltung dem Studierenden zugeteilt. Die Einschränkung kann gelockert werden, sodass das Anmeldeverfahren auf andere Anwendungsfälle von Zuteilungen mehrerer Veranstaltung pro Zuteilungsgruppe angewendet werden kann. Hierzu benötigt es gegebenenfalls andere Formen von Präferenzerhebung sowie andere Datenstrukturen. Von seiten des Bundled Probabilistic Serial Mechanismus kann theoretisch eine beliebige Kombination von Veranstaltungen den Studierenden zugeteilt werden.

Studierende ohne zugeteiltem Platz stehen momentan auf der jeweiligen Warteliste ihrer gewünschten Übung. Gegebenenfalls ist es sinnvoll über eine zweite Runde eine Verteilung auf Restplätze zu erzeugen. Das Plugin kann entsprechend erweitert werden, sodass Lehrende eine solche Zuteilung anfordern kann.

Weiterhin gab es auf der ehemaligen Vorlesungsverwaltungsplattform des Lehrstuhls von Prof. Dr. Hagerup die Möglichkeit der Freundesoption. Die berechnete Zuteilung soll es ermöglichen, dass befreundete Studierende die gleiche Übung besuchen, wenn es ihre Prioritäten sowie die Kapazitäten der Übungen erlauben. Hierbei kann das Iterative Rundungsverfahren um „weiche“ Restriktionen im Optimierungsproblem erweitert werden, die möglichst erfüllt werden sollen, aber verletzt werden können, falls sonst keine deterministische Zuteilung gefunden werden kann.

Nach einer Zuteilung sollen die Prioritäten der Studierenden nicht automatisch gelöscht werden, sondern gegebenenfalls anonymisiert gespeichert werden, sodass Statistiken über Angebot und Nachfrage sowie die Effizienz der Zuteilung erhoben werden können. Diese Informationen helfen Lehrende dabei ihren Übungsbetrieb besser planen und organisieren zu können. Insbesondere kann anhand von historischen Daten die notwendigen Kapazitäten und gewünschten Termine abgeschätzt werden. Digicampus soll um die Funktionalität des Exports der Daten und Statistiken beziehungsweise um Werkzeuge für die Planung von Veranstaltungen entsprechend erweitert werden.

Die Implementierungen dieser Bachelorarbeit soll zukünftig als Open-Source-Software öffentlich zur Verfügung gestellt werden. Es wäre weiterhin wünschenswert, wenn das Plugin durch Mitarbeiter der Universität, studentische Hilfskräfte oder freiwillige Studierende gewartet und weiterentwickelt wird. Insbesondere soll das Plugin barrierefrei verwendbar sein sowie für englischsprache Studierende internationalisiert werden. Weiterhin ist die Robustheit des Plugins und des Microservices durch Komponenten- sowie Integrationstests zu prüfen.

Abschließend soll nicht nur das Plugin, sondern auch das Kernsystem weiterentwickelt werden. Insbesondere soll die Anmeldekomponente in Stud.IP von den speziellen Klassen der Anmeldung über das Losverfahren entkoppelt werden. Weiterhin soll die Anmeldekomponente leicht durch Plugins ohne Verwendung von Workaround-Lösungen erweiterbar sein.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Atila Abdulkadiroğlu und Tayfun Sönmez. „Ordinal efficiency and dominated sets of assignments“. In: *Journal of Economic Theory* 112.1 (1. Sep. 2003), S. 157–172. ISSN: 0022-0531. DOI: 10.1016/S0022-0531(03)00091-7. URL: http://www.sciencedirect.com/science/article/pii/S0022053103000917 (besucht am 29. 10. 2019).

[2] *Academic Program and Licenses*. Gurobi. URL: https://www.gurobi.com/academia/academic-program-and-licenses/ (besucht am 18. 11. 2019).

[3] Martin Bichler, Sören Merting und Aykut Uzunoglu. „Assigning Course Schedules: About Preference Elicitation, Fairness, and Truthfulness“. In: (6. Dez. 2018). URL: https://arxiv.org/abs/1812.02630v2 (besucht am 08. 04. 2019).

[4] Garrett Birkhoff. „Three observations on linear algebra“. In: *Univ. Nac. Tucumán. Revista A.* 5 (1946), S. 147–151.

[5] Anna Bogomolnaia und Hervé Moulin. „A New Solution to the Random Assignment Problem“. In: *Journal of Economic Theory* 100.2 (1. Okt. 2001), S. 295–328. ISSN: 0022-0531. DOI: 10.1006/jeth.2000.2710. URL: http://www.sciencedirect.com/science/article/pii/S0022053100927108 (besucht am 08. 04. 2019).

[6] Marco Bohnsack. *Es war einmal: Stud.IP 0.1 – Eine Arbeitserleichterung muss her!* Das Stud.IP-Blog. 30. Mai 2007. URL: https://blog.studip.de/beitrag/studip-01-eine-arbeitserleichterung-muss-her (besucht am 22. 11. 2019).

[7] Eric Budish u. a. „Course Match: A Large-Scale Implementation of Approximate Competitive Equilibrium from Equal Incomes for Combinatorial Allocation“. In: *Operations Research* 65.2 (28. Okt. 2016), S. 314–336. ISSN: 0030-364X. DOI: 10.1287/opre.2016.1544. URL: https://pubsonline.informs.org/doi/10.1287/opre.2016.1544 (besucht am 06. 10. 2019).

[8] *Dokumentation zum Vorlesungsverwaltungssystem*. 15. Feb. 2013. URL: https://thi-vv.informatik.uni-augsburg.de/vv/Dokumentation_VV_student.pdf.

[9] ECMA. *ECMA-404: The JSON Data Interchange Format*. Geneva, Switzerland: ECMA (European Association for Standardizing Information und Communication Systems), 2017. URL: http://www.ecma-international.org/publications/standards/Ecma-404.htm.

[10] Martin Fowler und James Lewis. *Microservices - a definition of this new architectural term*. martinfowler.com. 25. März 2014. URL: https://martinfowler.com/articles/microservices.html (besucht am 15. 11. 2019).

[11] Allan Gibbard u. a. „Manipulation of voting schemes: a general result“. In: *Econometrica* 41.4 (1973), S. 587–601.

[12] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2019. URL: http://www.gurobi.com.

[13] J. Honer, G. Lach und E. Zorn. „An IP-based Model for the Post-Enrollment-based Course Timetabling Problem at TU Berlin“. In: *In proceedings of the 7th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2015), 25 - 28 Aug 2015, Prague, Czech Republic*. Hrsg. von Z. Hanzalek u. a. 2015, S. 331–344.

[14] David Hugh-Jones, Morimitsu Kurino und Christoph Vanberg. „An experimental study on the incentives of the probabilistic serial mechanism“. In: *Games and Economic Behavior* 87 (1. Sep. 2014), S. 367–380. ISSN: 0899-8256. DOI: 10.1016/j.geb.2014.06.001. URL: http://www.sciencedirect.com/science/article/pii/S0899825614001055 (besucht am 30. 09. 2019).

[15] *Introduction to Redis – Redis*. URL: https://redis.io/topics/introduction (besucht am 22. 11. 2019).

[16] Onur Kesten. „School Choice with Consent“. In: *The Quarterly Journal of Economics* 125.3 (1. Aug. 2010), S. 1297–1348. ISSN: 0033-5533. DOI: 10.1162/qjec.2010.125.3.1297. URL: https://academic.oup.com/qje/article/125/3/1297/1903670 (besucht am 29. 10. 2019).

[17] *Lehr- und Lernplattform Digicampus der Universität Augsburg*. URL: https://digicampus.uni-augsburg.de/ (besucht am 29. 10. 2019).

[18] *Matching plattform der Technischen Universität München*. URL: https://matching.in.tum.de/ (besucht am 29. 10. 2019).

[19] Sören Merting, Martin Bichler und Aykut Uzunoglu. „Course Assignment with Complementarities“. Matching in Practice Workshop. Cologne, Germany, 16. März 2018. URL: http://www.matching-in-practice.eu/wp-content/uploads/2017/11/Matching_MiP_3_18.pdf (besucht am 08. 04. 2019).

[20] Sören Merting, Paul Karaenke und Martin Bichler. „Strategy-Proof Assignment of Bundles with Ordinal Preferences: An Application in Retail Logistics“. In: *Multikonferenz Wirtschaftsinformatik (MKWI) 2016*. Hrsg. von Volker Nissen u. a. Technische Universität Ilmenau: Universitätsverlag, März 2016. ISBN: 978-3-86360-132-4.

[21] Thành Nguyen, Ahmad Peivandi und Rakesh Vohra. „Assignment problems with complementarities“. In: *Journal of Economic Theory* 165 (1. Sep. 2016), S. 209–241. ISSN: 0022-0531. DOI: 10.1016/j.jet.2016.04.006. URL: http://www.sciencedirect.com/science/article/pii/S0022053116300151 (besucht am 08. 04. 2019).

[22] André Noack. *Kommentar eines Entwicklers zur Funktionsweise des Losverfahrens unter Prioritäten in Stud.IP*. Arbeitsgruppe: Developer-Board - Forum - Stud.IP Entwicklungs- und Anwendungsforum. 25. Okt. 2019. URL: https://develop.studip.de/studip/plugins.php/coreforum/index/index/de70b10163fc8a917d8d656746301cc9?cid=a70c45ca747f0ab2ea4acbb17398d370#de70b10163fc8a917d8d656746301cc9 (besucht am 29. 10. 2019).

[23] André Noack. *Stud.IP Programmierung*. 22. Dez. 2011. URL: http://develop.studip.de/studip/download/force_download/0/8217c5e9c3b82ab83e388d8aa2ce339f/studip_programmierung_20111222.pdf.

[24] Patrick Noack, Peter Rosina und Bernhard Strehl. „Digicampus: Integration von E-Learning-Werkzeugen und Realisierung einer campusweiten Lehr-/Lernplattform“. In: (), S. 12.

[25] *NumPy license — NumPy*. URL: https://numpy.org/license.html (besucht am 18. 11. 2019).

[26] Travis Oliphant. *NumPy: A guide to NumPy*. Published: USA: Trelgol Publishing. 2006. URL: http://www.numpy.org/ (besucht am 27. 07. 2019).

[27] *OpenAPI Spezifikation 3.0.2*. OpenAPI Specification | Swagger. 8. Okt. 2018. URL: https://swagger.io/specification/ (besucht am 22. 11. 2019).

[28] Abraham Othman, Christos Papadimitriou und Aviad Rubinstein. „The Complexity of Fairness through Equilibrium“. In: *EC 2014 - Proceedings of the 15th ACM Conference on Economics and Computation* (21. Dez. 2013). DOI: 10.1145/2600057.2602855.

[29] *Plattform zur Vorlesungsverwaltung des Lehrstuhls für Theoretische Informatik*. Vorlesungsverwaltung. URL: https://thi-vv.informatik.uni-augsburg.de/vv/ (besucht am 29. 10. 2019).

[30] *Readme des Frameworks Connexion*. GitHub. URL: https://raw.githubusercontent.com/zalando/connexion/ec37d03902cf471e00dc7f5e19da99d3c652b94c/README.rst (besucht am 22. 11. 2019).

[31] *RQ: Documentation Overview*. URL: https://python-rq.org/docs/ (besucht am 22. 11. 2019).

[32] Mark Allen Satterthwaite. „Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions". In: *Journal of Economic Theory* 10.2 (Apr. 1975), S. 187–217. ISSN: 00220531. DOI: 10.1016/0022-0531(75)90050-2. URL: https://linkinghub.elsevier.com/retrieve/pii/0022053175900502 (besucht am 29. 10. 2019).

[33] *Stud.IP-Entwicklerdokumentation: Die Stud.IP-Plugin-Schnittstelle*. URL: https://hilfe.studip.de/develop/Entwickler/PluginSchnittstelle (besucht am 22. 11. 2019).

[34] Aykut Uzunoglu. *Analysis of Algorithms for Matchings with Bundled Preferences*. 2017.

[35] John Von Neumann. „A certain zero-sum two-person game equivalent to the optimal assignment problem". In: *Contributions to the Theory of Games* 2.0 (1953), S. 5–12.