Goethe-Universität Frankfurt

Bachelorarbeit

Flexibel einsetzbares Gruppeneinteilungssystem

eingereicht bei

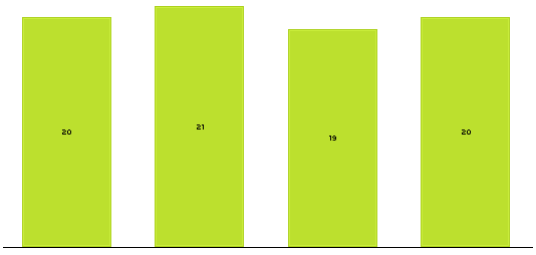
Prof. Dr.-Ing. D. Krömker

Dipl.-Inf. David Weiß

Professur für Graphische Datenverarbeitung

von

Pavel Safre



Eingereicht am: 10.09.2012

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Frankfurt am Main, den 10. September 2012

_____

(Pavel Safre)

Danksagung

Zunächst möchte ich mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben.

Vielen Dank an Herrn Krömker, der bei der Erarbeitung des Konzepts stark geholfen hat, an Herrn Bosse, der mich bei der Ausarbeitung vom theoretischen Konzept, Verbesserung der Formulierungen und Sammlung von Anforderungen unterstützt hat und an Herrn Weiß für seine Hinweise zur Weiterentwicklung. Ich bedanke mich außerdem bei allen Mitarbeitern der Professur für Graphische Datenverarbeitung für die Kooperation und konstruktive Kritik.

Mein besonderer Dank geht an Frau Sabrina Brandt für die Prüfung der Ausarbeitung sowie auch die wertvollen Verbesserungsvorschläge.

Inhaltsverzeichnis

Zusammenfassung

Diese Arbeit zeigt, dass die optimale Verteilung von Individuen in Gruppen unter Beachtung ihrer Zeitpräferenzen ein NP-schweres Problem ist. Daher liegt es Nahe, dass für große Teilnehmerzahlen eine optimale Lösung nicht in praxistauglicher Zeit berechnet werden kann. Hier kann eine geeignete Heuristik Abhilfe schaffen.

Da dieses Problem in Universitäten für Studierende bei der Zuteilung der Übungsgruppen für Hunderte von Teilnehmern jedes Semester aufs Neue gelöst werden muss, macht es Sinn, dabei eine rechnergestützte Lösung einzusetzen.

In dieser Arbeit werden die gängigsten in Deutschland und insbesondere an der Goethe-Universität Frankfurt am Main verwendeten Gruppeneinteilungssysteme untersucht. Alle aktuell eingesetzten Lösungen weisen offensichtliche Mängel auf. In dieser Arbeit wird analysiert, weshalb es dazu kommt und gezeigt, wie diese Mängel vermieden werden können. Außerdem werden Kriterien entwickelt und diskutiert, die ein gutes Gruppeneinteilungssystem erfüllen sollte.

Es wird beschrieben, inwiefern eine gute mit einer Heuristik schnell berechenbare approximative Lösung des Gruppeneinteilungsproblems besser als eine perfekte Lösung sein könnte. Mehrere Heuristiken werden verglichen und eine für dieses Problem gut passende wird entwickelt und implementiert. Mithilfe der Beispielimplementierung und anhand anonymisierter Anmeldedaten für die Veranstaltungen aus vergangenen Jahren wird gezeigt, welche Ergebnisse bei dem Wechsel zu einem solchen System erreicht werden können.

Weiterhin wird analysiert, wie ein solches Gruppeneinteilungssystem an die anderen an Universitäten eingesetzten digitalen Systeme angekoppelt werden kann. Das ist notwendig, um zu vermeiden, dass die redundanten Studierendendaten doppelt gepflegt werden müssen. Somit werden Konsistenz und Korrektheit der Daten bei dem Einsatz eines neuen Systems gefördert.

Abschließend wird ein Ausblick in die Zukunft der Gruppeneinteilungssysteme gegeben und beschrieben, welche Aspekte in diesem Bereich weiterhin wichtig sein könnten.

Abstract

This thesis shows that the optimal distribution of individuals in groups under consideration of their temporal preferences is a NP-hard problem. This suggests that an optimal solution cannot be computed in practical time. A suitable heuristic can provide thereby a remedy.

Due to the fact that this problem has to be solved for hundreds of course participants, when a university has to distribute students in groups for the group study, it makes sense to use thereby a PC-aided solution.

This thesis examines the group distribution systems, which are most commonly used in Germany and especially in Goethe University Frankfurt. All these solutions have obvious flaws. This thesis analyzes why these problems were encountered and how they may be prevented in the future implementations. Furthermore criteria is developed and discussed, that are essential for a good group distribution system.

Here is described, in what way a good approximate result, gained with a quick heuristic approach, may be better than a perfect result. Multiple heuristics were compared with each other and a new one was developed and implemented, that suits best for this particular problem. With the aid of the implemented prototype and the anonymized registration data of the past years it is demonstrated, what kind of results may be achieved, if such a system is used for this task.

In addition it is analyzed, how such group distribution system may be connected with the other computer systems in a university. This is necessary to avoid the need of the double maintenance of the redundant student data. Consequently the consistency and correctness of the user data is facilitated if an additional system is to be deployed.

In conclusion the forecasts in this branch are provided and it is shown, which aspects could be the most relevant in the future.

Kapitel 1

Einleitung

Diese Arbeit untersucht mögliche Lösungen für die konfliktfreie Vergabe von Plätzen bei der Verteilung von Teilnehmern einer Veranstaltung in Gruppen.

1.1 Motivation

Jeder kennt das Problem der Überschneidung von vorgegebenen Terminen, bei dem man eine Auswahl zwischen mehreren Angeboten treffen muss. Selbst gut organisierte Studierende können dieses Problem manchmal nur unzureichend lösen: Oft werden Übungstermine einer Veranstaltung parallel zu den Pflichtterminen einer anderen Veranstaltung angeboten.

Üblicherweise werden Übungstermine in großen Mengen für jeweils kleine Teilnehmerzahlen angeboten. Dies soll dann zumindest in der Theorie eine konfliktfreie Wahl der Termine für alle Studierende ermöglichen.

Bei ähnlichen Terminvorlieben der Studierenden, begrenzten Raummöglichkeiten und suboptimaler Gruppenverteilung führt ein Überangebot der Terminen allerdings zum Verkomplizieren der Auswahl, zur Konkurrenz zwischen den Mitgliedern um die Plätze sowie zur Überfüllung einzelner besonders populärer Gruppen. All dies steigert keinesfalls die Übungsqualität.

Als ich mit meinem Informatikstudium angefangen habe, wurden für die Übungseinteilung Gruppenlisten im Gebäude ausgehängt, wo Interessierte sich eintragen konnten. Diese Listen wurden nach Eintragungen abgearbeitet. Diese Lösung ist offensichtlich nicht perfekt, da die Studierenden, die nicht schnell genug am Aushang sind, keine Chance haben, die gewünschten Plätze zu bekommen.

Danach wurden in der Goethe-Universität Frankfurt verschiedene digitale Systeme der Gruppeneinteilung eingeführt und teilweise parallel benutzt. Jedes hat ebenfalls einige Schwachstellen, die ein *gutes* Gruppeneinteilungssystem vermeiden könnte.

So begann ich noch vor Beginn der Bachelorarbeit darüber nachzudenken, wie ein perfektes Gruppen-

einteilungssystem aussehen sollte und wie ein gutes Gruppeneinteilungssystem implementiert werden müsste. Während des Studiums habe ich bei der Kommunikation mit Kommilitonen sowie auch Dozenten viele Anforderungen gesammelt und dabei auch vieles über die Optimierungsverfahren gelernt.

1.2 Zielsetzung

In dieser Arbeit wird eine detaillierte Analyse des Gebiets der Gruppeneinteilung durchgeführt. Das Ziel dabei ist es, die Schwachstellen der wichtigsten in Deutschland verwendeten Systeme herauszukristallisieren, festzustellen, wie diese vermieden werden können und mittels einer Beispielimplementierung ein gutes und flexibles Gruppeneinteilungssystem zu entwickeln.

Ein Schwerpunkt dieser Arbeit ist das Erarbeiten von Kriterien, die für ein gutes Gruppeneinteilungssystem relevant sind. Ein weiterer Schwerpunkt ist die Entwicklung der Verfahren und Modelle, mit denen eine entsprechende Implementierung erfolgen kann.

1.3 Aufbau

Im Kapitel 2 dieser Arbeit werden die theoretischen Grundlagen und Notationen vorgestellt, die im Folgenden verwendet werden: Die Komplexitätsklassen, das allgemeine Zuordnungsproblem sowie auch das Gruppeneinteilungsproblem aus theoretischer Sicht.

Kapitel 3 beschreibt die aktuell existierenden Algorithmen und nennt ihre Vor- und Nachteile im Bezug auf die Gruppeneinteilung. Außerdem werden in diesem Kapitel die existierenden Gruppeneinteilungssysteme vorgestellt und beschrieben welchen Anforderungen sie nicht gerecht werden. Es wird außerdem gezeigt, welche Rolle Schnittstellen bei der praktischen Verwendung solcher Systeme spielen.

Im Kapitel 4 wird die neue Heuristik demonstriert und die Anforderungen eines guten Gruppeneinteilungssystems klargelegt.

Kapitel 5 erläutert die Gründe der Plattformauswahl für den Prototyp und zeigt welche Stellen bei der Implementierung in der Praxis schwieriger sind, als das sich aus der theoretischen Sicht erkennen ließ. Das Laufzeitverhalten des vorgeschlagenen Algorithmus wird analysiert, um eine Aussage über die Praxistauglichkeit ermöglichen zu können. Im Anschluss werden die Ergebnisse praktischer Tests mit anonymisierten Anmeldedaten aus vergangenen Jahren gezeigt, die die praktische Verwendbarkeit des vorgestellten Algorithmus bestätigen.

Das Kapitel 6 setzt sich damit auseinander, welche Entwicklungen im Bereich der Gruppeneinteilung vorstellbar sind und wozu sie führen würden.

Im abschließenden Kapitel 7 kommt eine kurze Zusammenfassung der beschriebenen Ideen und Konzepte.

Kapitel 2

Grundlagen

In diesem Kapitel wird beschrieben, wie genau das Gruppeneinteilungsproblem aus mathematischer Sicht formuliert werden kann und warum das NP-schwer ist. Die für das Verständnis davon notwendigen theoretischen Konzepte werden eingeführt.

2.1 Theoretische Konzepte

So wie hinter jeder Verwirklichung immer eine Idee steht, steht hinter jeder Implementierung ein Algorithmus. Einige Algorithmen wurden in der Theorie bereits vor Hunderten von Jahren beschrieben, aber aus diversen Gründen nicht umgesetzt. Auf der anderen Seite existieren auch viele Probleme, für die nachgewiesen wurde, dass keine Lösungen existieren können, die in akzeptabler Zeit berechnet werden können, wenn die gestellten Aufgaben schwer genug sind.

Die Theoretische Informatik beschreibt unter anderem die sogenannte NP-Klasse der Probleme. Dazu gehören die Entscheidungsprobleme, die in der Polynomialzeit (abhängig von der Eingabelänge) nichtdeterministisch gelöst werden können. Bisher existieren dafür keine deterministische Algorithmen mit polynomialer Laufzeit.

Ein Problem gehört zur NP-Klasse, wenn eine (*geratene*) Lösung deterministisch in Polynomialzeit verifiziert werden kann. Die schwierigsten Probleme dieser Klasse heißen *NP-vollständig*. Sie zählen zu den Schwierigsten, da 1971 von Stephen Cook nachgewiesen wurde, dass alle anderen Probleme aus der NP-Klasse in Polynomialzeit auf diese reduziert werden können.

Diejenigen Probleme, die mindestens genauso schwer wie die NP-vollständige sind, heißen *NP-schwer*. Diese Probleme müssen nicht unbedingt in der NP-Klasse liegen. Sie müssen also auch nicht unbedingt Entscheidungsprobleme sein.

Ein Beispiel der NP-schweren Probleme ist das bekannte Problem des Handlungsreisenden (engl.: *Traveling Salesman Problem, TSP*):

> Der Handlungsreisende muss n Städte besuchen, um dort zu versuchen, seine Waren zu verkaufen. Dabei darf keine Stadt mehrmals besucht werden. Da der Handlungsreisende nicht pro Stunde bezahlt wird, ist es für ihn sinnvoll, möglichst wenig Zeit für die Reise von einer Stadt in die andere zu verbrauchen. Dafür muss die gesamte Reisestrecke möglichst kurz sein.
> Der Handlungsreisende muss außerdem am Ende seiner Reise in der gleichen Stadt landen, wo er am Anfang der Reise war (seine Heimatstadt).

Das Problem scheint auf den ersten Blick nicht sehr schwierig zu sein. Bei dem Versuch, es für eine gegebene Instanz zu lösen sieht man allerdings, dass es unmöglich ist, selbst die erste zu besuchende Stadt der Reise zu bestimmen, ohne dass man den kürsesten Restweg für die restlichen $n-1$ Städte berechnet.

Löst man das Problem also per Brute Force[1], folglich mit einer vollständigen Enumeration aller möglichen Belegungen, muss man erstmal die Strecken zu n potenziellen Städten beachten, dann jeweils $n-1$ Strecken zur zweiten Stadt u.s.w. bis am Ende nur eine Stadt bleibt. Die Summen aller solcher Rundreisen müssen miteinander verglichen werden, und die kürzeste davon wird ausgewählt.

Was ist dabei der gesamte Rechenaufwand? Man benötigt

$$n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 = n!$$

Rechenschritte zur Feststellung der Längen aller Wege. Selbst für eine relativ kleine Problemgröße von 25 Städten werden somit $25! = 1.5 \cdot 10^{25}$ Rechenschritte benötigt. Sollte unser Computer Zehn Milliarden Berechnungen pro Sekunde durchführen können, wären für die Berechnung ca. $1.5 \cdot 10^{15}$ Mio. Sekunden oder 1.2 Mrd. Jahre benötigt. Zum Vergleich: Selbst der aktuell schnellste Supercomputer (Sequoia[2]) mit $1.632 \cdot 10^{16}$ Operationen pro Sekunde wird dafür 950 Mio. Sekunden oder 723 Jahren brauchen.

In der Praxis müssen solche Probleme allerdings auch für Hunderte Orte täglich gelöst werden (beispielsweise von den Paketdiensten, die Sendungen möglichst schnell an alle Empfänger liefern müssen).

Aus diesem Grund werden in der Praxis für die Berechnung der Lösung solcher Probleme bei hinreichend großen Eingaben *Heuristiken* eingesetzt. Ein heuristischer Algorithmus verwendet bestimmte erfahrungsbasierte Annahmen, die die Lösung stark vereinfachen. Damit lässt sich eine gute Lösung (vergleichsmäßig) sehr schnell finden.

[1] engl.: *rohe Gewalt*: Alle Möglichkeiten ausprobieren
[2] Stand 05.09.12 gemäß http://www.top500.org/

Der Nachteil solcher Lösungen besteht darin, das sie nur lokal optimale Lösungen finden, die in den meisten Fällen deutlich von der optimalen (globalen) Lösung abweichen. Der Postmann, der sonst 182 Jahre auf den kürzesten möglichen Weg warten müsste, wäre allerdings für eine selbst anderthalb so lange Alternative, die sofort zur Verfügung stehen könnte, viel dankbarer.

In [1] wird beispielsweise für bis zu 13 Städte ein Algorithmus eingesetzt, der eine perfekte Lösung findet. Für das Lösen des Problems für 13 Städte werden dabei 1.75 Sekunden benötigt (IBM 7094 II). Wenn als Eingabe allerdings 14 bis 145 Städte kommen, wird aufgrund der zu langen Dauer der Ausführung des ersten Algorithmus anstatt dessen ein heuristischer Algorithmus verwendet, der eine gute Lösung innerhalb von $30 \cdot n^3$ µs auf dem gleichen Rechner liefert. Für 14 Städte sind das $30 \cdot 14^3 = 82320\,\mu s \approx 0.082\,s$ - nur ein Bruchteil der Zeit, die der erste Algorithmus für eine kleinere Aufgabe benötigt hat. Innerhalb von 1.78 s kann man mit dem heuristischen Algorithmus das Problem für 39 Städte lösen (genau drei mal größer als mit dem ersten Algorithmus).

2.2 Verallgemeinertes Zuordnungsproblem

Im Rahmen der Gruppeneinteilung beschäftigen wir uns mit einer Variante des *allgemeinen Zuordnungsproblems* (engl.: *generalized assignment problem, GAP*). Dieses ist bewiesen NP-schwer [2] und ist damit mindestens genau so schwer wie das oben beschriebene TSP.

Die Problemstellung ist:

> Es existieren eine Menge von Mitarbeitern und eine Menge von Aufgaben. Jeder Mitarbeiter kann eine beliebige Aufgabe durchführen. Dabei entstehen ihm bestimmte Kosten.
> Diese Kosten können variieren, abhängig davon, welcher Mitarbeiter welche Aufgabe durchführt (Erfahrungsunterschiede). Außerdem hat jeder Mitarbeiter ein Budget, das die Summe der Kosten der von ihm durchgeführten Aufgaben nicht übertreffen darf.
> Das Ziel besteht darin, alle vorhandenen Aufgaben zwischen den Mitarbeitern so zu verteilen, dass die gesamten Kosten minimiert werden bzw. dass der gesamte Profit maximiert wird.

Mathematisch gesehen kann das Problem so beschrieben werden: [3]

$$
\begin{aligned}
&\text{(GAP) } min \sum_i \sum_j c_{ij} \cdot x_{ij} \\
&sodass \quad \sum_j a_{ij} \cdot x_{ij} \leq b_i && i \in \{1,\dots,m\} \\
&\qquad \sum_i x_{ij} = 1 && j \in \{1,\dots,n\} \\
&\qquad x_{ij} \in \{0,1\} && i \in \{1,\dots,m\}, \quad j \in \{1,\dots,n\}
\end{aligned}
$$

Dabei bezeichnen:

- m die Mitarbeiteranzahl
- n die Aufgabenanzahl
- c_{ij} die Kosten der Zuweisung der Aufgabe j an den Mitarbeiter i
- a_{ij} den Kapazitätsverlust, wenn die Aufgabe j dem Mitarbeiter i gegeben wurde
- b_i die Kapazität des Mitarbeiters i
- x_{ij} ist gleich 1, falls der Mitarbeiter i die Aufgabe j durchführt und sonst 0

Um eine garantiert perfekte Lösung zu finden, müssen wir bei der Lösung dieses Problems alle möglichen Zusammensetzungen der Aufgaben bei jedem Mitarbeiter analysieren und die dabei entstehenden Kosten aufsummieren. Wenn jede Aufgabe an einen der m Mitarbeiter gesendet wird und es insgesamt n Aufgaben gibt, benötigen wir dafür mindestens n^m Rechenschritte. Für 100 Aufgaben und 10 Mitarbeitern wären das $100^{10} = 10^{20}$ Rechenschritte. Bei Zehn Milliarden Berechnungen pro Sekunde würden wir für die Berechnung vom optimalen Ergebnis 10^{10} s oder ca. 7610 Jahren benötigen.

Da das in der Praxis nicht akzeptabel ist, werden auch hier diverse Heuristiken verwendet.

2.3 Problemstellung der Gruppeneinteilung

Bei meiner Recherche habe ich leider keine Literatur gefunden, die sich direkt mit der Gruppeneinteilung beschäftigt. Daher bringe ich hier mein Konzept ein, das auf der Betrachtung der Eigenschaften des Problems und der Ähnlichkeit der Problemstellung zum verallgemeinerten Zuordnungsproblem basiert.

Bei der Einteilung von Teilnehmern in Gruppen ist das Ziel, unter Beachtung der Terminwünsche der Teilnehmer, möglichst gleiche Gruppen zu bauen. Die Gruppen müssen möglich gleich sein, damit der Aufwand aller Gruppenbetreuer vergleichbar ist und damit keine Gruppen entstehen, deren Teilnehmer nicht in die für die Übungen vorgesehene Räume passen.

Da das Problem empirisch bestätigt bei nur einem Wunsch pro Person wegen ähnlicher Zeitpräferenzen der Teilnehmer zu großen Diskrepanzen führt, müssen dabei von Teilnehmern mehrere Wunschtermine ausgewählt werden. Aufgrund der Tatsache, dass dabei einige Termine weniger passend und dementsprechend weniger wünschenswert für die Teilnehmer sind, sollen die Präferenzen beispielsweise im Form der Prioritäten angegeben werden. Ein weiteres dabei entstehendes Ziel ist es, möglichst viele hohe und möglichst wenige niedrige Prioritäten zu erfüllen. Mindestens ein Wunsch eines Teilnehmers muss erfüllt werden, damit dieser Teilnehmer zur Übung auch erscheinen kann.

Dabei müssen (möglichst) alle Teilnehmer eingeteilt werden. Die Einteilung muss innerhalb einer praxistauglichen Zeit geschehen, damit die Ergebnisse in der Praxis auch verwendet werden können.

Wenn wir diese Formulierung analog zur Definition des verallgemeinerten Zuordnungsproblems umformulieren, bekommen wir die folgende Aufgabenstellung:

> Es existieren eine Menge von *Gruppen* und eine Menge von *Kursteilnehmern*. Jede *Gruppe* kann einen beliebigen *Teilnehmer* beherbergen. Dabei entstehen bestimmte Kosten.
> Diese Kosten können variieren, abhängig davon, in welche *Gruppe* welcher *Teilnehmer* eingeteilt wird (*im Form von erfüllten oder nicht erfüllten Prioritäten*). Außerdem existiert eine *durchschnittliche Größe*, von der die *Anzahl der eingeteilten Mitglieder* möglichst wenig abweichen sollte.
> Die Aufgabe besteht darin, alle vorhandenen *Teilnehmer* zwischen den *Gruppen* so zu verteilen, dass die gesamten Kosten minimiert werden *und* das die Gruppen *möglichst gleiche Größen* haben.

Wie man anhand der Definition sehen kann, ist damit dieses Problem dem verallgemeinerten Zuordnungsproblem sehr ähnlich. Außer der Umbenennungen beziehen sich die Unterschiede auf zwei Vereinfachungen:

1. Die Kosten sind nicht mehr beliebig groß, sondern können nur die Werte annehmen, die der Erfüllung eines Wunsches entsprechen (z.B. 1,2,3 für den 1., 2. und 3. Wunschtermin und ∞ für die Nichterfüllung).

2. Der Kapazitätsverlust bei der Aufnahme eines Teilnehmers ist immer gleich und kann somit gleich 1 genommen werden.

Diese Vereinfachungen können allerdings die allgemeine Problemkomplexität leider nicht senken. Außerdem kommt ein weiterer Parameter ins Spiel, der die Schwierigkeit weiter erhöht:

Beim verallgemeinerten Zuordnungsproblem werden maximale Gruppengrößen b_i vorgegeben, die eingehalten werden müssen. Unser Ziel ist jedoch, die Abweichung aller Gruppengrößen von der durchschnittlichen Gruppengröße zu minimieren. Das führt dazu, dass wir nicht eine, sondern zwei verschiedene Zielfunktionen haben, nach den optimiert werden muss.

Mathematisch gesehen kann somit das Problem der Gruppeneinteilung (GEP) so beschrieben werden:

Gegeben:	$c_{ij} \in \{1,2,3,\infty\}$	$i \in \{1,\ldots,m\}, \quad j \in \{1,\ldots,n\}$	
Bestimme:	$x_{ij} \in \{0,1\}$	$i \in \{1,\ldots,m\}, \quad j \in \{1,\ldots,n\}$	
Wobei:	$\sum_i x_{ij} = 1$	$j \in \{1,\ldots,n\}$	(G1)
	$c_{ij} = \infty \quad \Rightarrow \quad x_{ij} = 0$		(G2)

Zielfunktionen:

$$min \sum_i \sum_j c_{ij} \cdot x_{ij} \tag{B1}$$

$$min \sqrt{\sum_i \left(\sum_j x_{ij} - \frac{n}{m} \right)^2} \tag{B2}$$

Dabei bezeichnen:

- m die Gruppenanzahl
- n die Teilnehmeranzahl
- c_{ij} $\in \{1,2,3,\infty\}$ die Kosten der Einteilung der Person j in die Gruppe i
- x_{ij} ist gleich 1, falls der Teilnehmer j in die Gruppe i eingeteilt wird und sonst 0
- $\frac{n}{m}$ die mittlere Gruppengröße i

Die Einteilungsrichtlinien:

(G1) besagt, dass jeder Teilnehmer genau in eine Gruppe eingeteilt wird.
(G2) garantiert die Einteilung der Teilnehmer nur in die Gruppen, deren Besuch für diese Teilnehmer gemäß ihrer Prioritäten möglich ist.

Die Zielfunktionen:

(B1) minimiert die insgesamt erfüllten Wünsche, um möglichst die höchsten Prioritäten zu erfüllen.
(B2) minimiert die Standardabweichung einzelner Gruppengrößen, damit die Gruppen möglichst gleichmäßig gefüllt werden.

Eine Lösung zu finden, die beide Zielfunktionen gleichzeitig minimiert, wird im Allgemeinen nicht möglich sein, da sie sich auf unterschiedliche Größen beziehen und voneinander unabhängig ablaufen. Daher muss die Entscheidung getroffen werden, wie die beiden Zielfunktionen bei der Berechnung der optimalen Einteilungen berücksichtigt werden können.

Die Möglichkeiten sind:

1. Einen der Werte als führenden Wert auszuwählen und danach zu optimieren.
   Nachteil: Das Ergebnis kann nicht praxistauglich sein, da ein wichtiges Kriterium unberücksichtigt bleibt.
   Beispiel 1: Gleichgroße Gruppen, wo Teilnehmer eingeteilt wurden, die zu den Übungsterminen nicht erscheinen können.
   Beispiel 2: Erfüllung aller Wünsche der ersten Priorität, die dazu führt, dass einige überfüllte Gruppen entstehen, wo Teilnehmer keine Sitzplätze haben und die Übungsleiter über nicht ausreichend Zeit für die einzelnen Teilnehmer verfügen.

2. Den beiden Werten Gewichte zuweisen
   Nachteil: Es ist nicht klar, was von den beiden Faktoren wie viel wichtiger ist. In Einzelfällen kann einer der beiden Faktoren keine Rolle spielen.
   Beispiel: Eine Gruppe wird von einem wissenschaftlichen Mitarbeiter geleitet. Die andere - von einem Student. Der erste könnte auch eine größere Gruppe übernehmen.

3. Bei einem der Faktoren Güteklassen definieren. In jeder Güteklasse nach dem zweiten Parameter optimieren. Sollten im Hinsicht auf den zweiten Parameter äquivalente Ergebnisse auftauchen, nach dem ersten Faktor entscheiden, welche Lösung ausgewählt wird. Im Anschluss den Kursleitern die Möglichkeit geben, eine Auswahl zwischen den gelieferten Lösungsvorschlägen zu treffen.
   Nachteil: Hoher Organisationsaufwand bei der Analyse von vielen Vorschlägen.
   Lösung: Angabe von wenigen Vorschlägen und rechnergestützte Unterstützung im Form von Graphen.

Die dritte Möglichkeit ist sinnvoll und flexibel und könnte daher bei der Lösung bevorzugt werden.

Praxistauglich könnte für dieses Problem auf den ersten Blick aufgrund der Komplexität höchstwahrscheinlich nur ein heuristischer Algorithmus sein.

Kapitel 3

State of the Art

Dieses Kapitel beschreibt die aktuell existierenden Algorithmen, die für die Lösung des Gruppeneinteilungsproblems verwendet werden können. Anschließend werden die heutzutage verwendeten Gruppeneinteilungsverfahren analysiert und die existierende Gruppeneinteilungssysteme verglichen.

3.1 Algorithmen

Ein gutes Gruppeneinteilungssystem muss auf einem Algorithmus basieren, der eine gute Lösung in akzeptabler Zeit findet. Die heutzutage für das verallgemeinertes Zuordnungsproblem verwendeten Algorithmen müssen aus diesem Grund im Aspekt der Tauglichkeit für eine praktische Anwendung für die Gruppeneinteilung analysiert werden.

3.1.1 Exakte Methoden

Brute-Force

Unter der sogenannten Brute-Force-Methode wird gemeint, dass alle möglichen Belegungen durchgesucht werden, um die optimale Lösung zu finden. Wie wir bei der Problemstellung des Gruppeneinteilungsproblems gesehen haben, werden dafür exponentiell viele Schritte benötigt, was dieses Verfahren bei den praxisrelevanten Problemgrößen (z.B. 600 Teilnehmer; 15 Gruppen) untauglich macht.

Branch-and-Bound-Methoden

Die *Branch-and-Bound*[1]-basierten Methoden können als eine verbesserte Version der Brute-Force-Methode betrachtet werden: Damit nicht alle möglichen Belegungen überprüft werden müssen, wird

[1] engl.: *Verzweigung und Schranke*

die Lösungsmenge aufgrund bestimmter Entscheidungen in mehrere kleinere Mengen geteilt (*Verzweigung*), die kleinere Probleme repräsentieren und daher einfacher gelöst werden können und dann rekursiv ihre Lösungen übergeben. Damit nicht alle Belegungen wie bei der Brute-Force-Methode untersucht werden müssen, werden geeignete *Schranken* dafür eingesetzt, dass die Zweige, bei denen im Voraus deutlich wird, dass sie auf jeden Fall nicht das optimale Ergebnis liefern, sofort abgebrochen und somit nicht weiter berechnet werden.

Das Laufzeitproblem bleibt bei dieser Methode leider erhalten: Im Worst Case ist mit der Aufzählung aller Belegungen zu rechen (exponentieller Rechenaufwand), im Regelfall - mit Bruchteilen davon, die ebenfalls exponentiell groß sind, wie in [4] gezeigt wird.

Lineare Programmierung

Für die einzelnen Fälle des allgemeinen Zuordnungsproblems wird oft Lineare Optimierung angewandt. Ein Beispiel hierzu (die Simplex-Methode) wurde in [5] beschrieben. Bei diesem Algorithmus wird durch die Gleichungen und Ungleichungen der Aufgabenstellung ein Polyeder[2] definiert. Die optimale Lösung befindet sich in einer seiner Ecken. Der Algorithmus durchläuft die Kanten des Polyeders in die Richtungen, wo bessere Ergebnisse (bei der Verwendung der dieser Stelle im Graph entsprechenden Belegung) geliefert werden. Falls von einer Ecke ausgehend keine Nachbarn mit einem besseren Wert existieren, ist die Belegung, die dieser Ecke entspricht, das globale Optimum bzw. einer der globalen Optima falls mehrere gleichwertige perfekte Lösungen existieren.

Für das Gruppeneinteilungsproblem eignet sich diese Vorgehensweise ebenfalls nicht, da sie nur für Probleme mit kontinuierlichen Variablen angewandt werden kann [6], und bei dem Problem der Gruppenzuteilung mit diskreten Werten gearbeitet wird.

3.1.2 Approximationsalgorithmen

Wenn eine perfekte Lösung nicht innerhalb der akzeptablen Zeit gefunden werden kann, muss ein Approximationsalgorithmus eingesetzt werden, der eine *gute* Lösung anstatt dessen liefern kann.

Branch-and-Cut

Die *Branch-and-Cut*-Methode verwendet die Vorgehensweise der Linearen Programmierung, um eine reelle Lösung der gestellten Problems zu finden. Da eine ganzzahlige Lösung erwartet wird, wird bei einem nicht-ganzzahligen Ergebnis das sogenannte *Schnittebenenverfahren* (engl.: *cutting plane algorithm*) dafür eingesetzt, um weitere Ungleichungen zu finden, die solche Ergebnisse verbietet. Mit der neuen Ungleichung kann das Ungleichungssystem wieder gelöst werden. Dieser Prozess wird

[2] ein dreidimensionales Vieleck

Abbildung 3.1: Branch and Cut: Eine nicht ganzzahlige Lösung

iteriert, bis entweder eine ganzzahlige Lösung gefunden wird oder das Schnittebenenverfahren keine weitere Ungleichungen liefern kann.

Im ersten Fall ist das Endergebnis gefunden. Im zweiten Fall wird eine Verzweigung (*Branch*) durchgeführt, und das gesamte Problem wird in zwei Unterprobleme geteilt:

1. Mit der zusätzlichen Bedingung, dass das Ergebnis kleiner oder gleich der vom gefundenen Wert aus nächst kleineren ganzen Zahl ist.
2. Mit der zusätzlichen Bedingung, dass das Ergebnis größer oder gleich der vom gefundenen Wert aus nächst größeren ganzen Zahl ist.

Wenn beispielsweise als Ergebnis 1.8 geliefert wurde, ist die erste zusätzliche Bedingung: $Ergebnis \leq 1$ und die zweite: $Ergebnis \geq 2$. Diese Unterprobleme schließen den gefundenen Wert und alle anderen nicht ganzzahligen Werte aus, die in seiner unmittelbaren Umgebung stehen, wie man auf der Abbildung 3.1 sieht. Außerdem operieren sie mit zweimal kleineren Wertebereichen und können daher deutlich einfacher zum Ergebnis kommen. Dann wird der ganze Prozess nochmals wiederholt.

Branch-and-Cut ist mehrmals schneller als die exakten Methoden beim Finden einer Lösung. Für die Praxis eignet sich dieser Algorithmus nur begrenzt, da er bei erheblichen Problemgrößen immer noch mehrmals langsamer als heuristische Methoden zum Ergebnis führt, was zu Schwierigkeiten führen könnte, wenn die Berechnung eine Woche lang laufen muss. Der Lösungsweg zum Endergebnis ist außerdem schlecht verfolgbar.

Eine effiziente Implementierung dieser Methode ist das Open Source Programm *lp_solve*[3], die eine gute, garantiert sehr nah an dem Optimum liegende Lösung für die Gruppengrößen bis 100 Teilnehmer innerhalb von einigen Minuten berechnen kann. Aus diesem Grund wird sie vom Gruppeneinteilungssystem der TU Kaiserslautern verwendet, das in dieser Arbeit beim Vergleich der existierenden Gruppeneinteilungssysteme analysiert wird.

Genetische Algorithmen

Genetische Algorithmen sind eine der Metaheuristiken, mit denen die Optimierung beliebiger Probleme möglich ist. Sie sind computerbasierte Problemlösungssysteme, die berechenbare Modelle von natürlichen, evolutionären Prozessen als Schlüsselelemente verwenden.[6] Eine Belegung aller unbekannten Variablen (beispielsweise der Inhalt der Ergebnisgruppen bei der Gruppeneinteilung) wird

[3] http://lpsolve.sourceforge.net/5.5/

bei dem Ansatz der genetischen Optimierung als Genom bezeichnet und entspricht der biologischen DNA. Wie im biologischen Vorbild, sind hierbei *Mutationen* und *Rekombinationen* möglich.

Bei der Mutation können einzelne Werte eines Genoms zufällig geändert werden (*Flipmutation*). Eine andere Art der Mutation (*Swapmutation*) vertauscht die Werte zweier Variablen innerhalb eines Genoms.

Bei der Rekombination werden zwei oder mehr Genome (Elterngenome) zu einem Genom zusammengefasst, indem ihre einzelnen Anteile an die entsprechenden Plätze des Ergebnis- bzw. Kindgenoms kopiert werden.

Für die Bestimmung der Güte eines Genoms bzw. einer Belegung der Variablen wird die sogenannte *Fitnessfunktion* eingesetzt. Alle aktuell betrachteten Genome werden mit einer Wahrscheinlichkeit, die von ihrem Fitnesswert abhängt, ausgewählt.

Bei jedem Evolutionsschritt (entspricht einer Optimierungsiteration) werden Mutationen bzw. Rekombinationen auf die ausgewählten Genome angewandt. Damit die besten Lösungen nicht verloren gehen, werden die Genome mit dem besten Fitnesswert *gerettet*, d.h. eine Kopie von ihnen wird unverändert in die neue Generation übertragen.

Eine *Straffunktion* beschreibt die Beschränkungen des für die Modellparameter zulässigen Bereiches. Falls ein Genom nicht im zulässigen Bereich liegt, wird dieses entweder gelöscht oder in eine zulässige Belegung mittels einer dafür extra erstellten Funktion überführt.

Da bei Genetischen Algorithmen beliebige Genome zur beliebigen Zeit auftauchen können, wird nicht nach einer lokalen, sondern nach einer globalen optimalen Belegung gesucht. Dieses Verfahren ist als eine Metaheuristik außerdem auch sehr einfach bei der Implementierung, da das Verfahren unabhängig von der Problemstellung ist und dementsprechend bereits existierende Programmausschnitte der eigentlichen Optimierung bei der Implementierung benutzt werden können.

Zu den Nachteilen des Verfahrens zählt, dass dieses relativ große Speicherkapazität für die aktuell betrachteten Genome braucht. Da das Verfahren blind nach Lösungen sucht, kann außerdem sehr viel Rechenzeit benötigt werden, um ein gutes Ergebnis zu erzielen.

Das optimale Ergebnis kann außerdem damit nicht garantiert werden; die Abweichung davon kann beliebig groß sein, abhängig von den Anfangsbelegungen. Genauso wie beim Branch-and-Cut kann außerdem hierbei der Lösungsweg nicht verfolgt werden.

Andere Metaheuristiken

Alle anderen Metaheuristiken (wie beispielsweise Tabusuche, Simulated Annealing, Variable Neighborhood Search und Random Search) zeigen im Bezug auf die Einsatzmöglichkeit für die Gruppeneinteilung ähnliche Nachteile wie Genetische Algorithmen auf:

- Die Berechnung ist zeitintensiv. Mit einem ausreichend guten Ergebnis ist nur nach einigen Tagen der Kalkulation zu rechnen.
- Das optimale Ergebnis bzw. eine Lösung, die sich dem optimalem Ergebnis nachweisbar angleicht, kann nicht garantiert werden.

Allerdings benötigen sie viel weniger Arbeitsspeicher für die Optimierung, da zu jedem Zeitpunkt jeweils mit einer Belegung aller Variablen und ihren Nachbarn[4] gearbeitet wird.

Spezielle Heuristik der Megadigitale

Das Gruppeneinteilungsskript von Megadigitale[5] verwendet für die Einteilung der Studierenden in Übungsgruppen eine eigene Heuristik:

1. Bei der Anmeldung bekommt jeder Teilnehmer eine Losnummer.
2. Die Bewerbungen werden nach Losnummer und dann nach Priorität (von 1 bis 3 aufsteigend) sortiert und im Array `bewerbungen` gespeichert.
3. Die Arrays `gruppenGrossen` und `maximaleGrossen` enthalten aktuelle und maximale Gruppengrößen für alle Gruppen.
4. Im Anschluss wird die Prozedur ausgeführt, die in dem Listing 3.1 gezeigt wird.

```
1  lastmtknr = -1;
2  foreach(k = 0; k < count(bewerbungen); k++) {
3    mtknr = bewerbungen[k]['mtknr'];
4    if( mtknr == lastmtknr )
5      continue; // bereits eingeteilt: weiter
6    wunschtermin = bewerbungen[k]['termin'];
7    if(gruppenGrossen[wunschtermin] > maximaleGrossen[wunschtermin])
8      // Gruppe voll:
9      if(isset(bewerbungen[k+1]) && mtnr==bewerbungen[k+1]['mtknr'])
10       continue; // Ausweichtermine vorhanden
11     else // nicht einteilbar
12       teilnehmer_eintragen_in_gruppe("NICHT␣EINGETEILT");
13 else
14   teilnehmer_eintragen_in_gruppe( wunschtermin ); // In Gruppe einteilen
15   lastmtknr = mtnr; // Erfolgreich eingeteilt: Weiter
16 }
```

Listing 3.1: Heuristik der Megadigitale

[4] Nachbarn einer Belegung mehrerer Variablen sind alle Belegungen gleicher Variablen, die sich höchstens in einem Wert von dem Original unterscheiden und dieser Wert bei ganzzahligen Variablen entweder um eins größer oder um eins kleiner ist.

[5] Ein Projekt zur Förderung von E-Learning an der Goethe-Universität Frankfurt

Die Anzahl der vom System bei der Einteilung betrachtenden Prioritäten sowie auch die maximalen Gruppengrößen sind vom Kursverwalter einstellbar.

Das Skript durchläuft in seiner Schleife (Zeilen 2-16) alle vorhandenen Bewerbungen. Falls der aktuelle Student noch nicht eingeteilt wurde, wird überprüft, ob in der Gruppe, auf die sich seine aktuelle Bewerbung bezieht, freie Plätze zur Verfügung stehen. Wenn ja, wird er in diese Gruppe eingeteilt. Sonst kommt er in die Liste der nicht eingeteilten Interessierten.

Der Hauptvorteil dieser Heuristik ist ihre Geschwindigkeit: Mit nur einer Schleife und atomaren Befehlen darin hat er die Komplexität von $\mathcal{O}(n)$ und kann daher rasant Einteilungen aller denkbaren Teilnehmerzahlen durchführen. Außerdem ist das Verfahren sehr übersichtlich und das Ergebnis jeder Einteilung kann einfach rückwärts verfolgt werden.

Der Nachteil besteht in der Qualität der erreichten Lösung:

- Die Gruppenverteilung ist sehr ungleichmäßig, da potenzielle Teilnehmer unpopulärer Gruppen in die ersten in Frage kommenden Gruppen gelangen; die populären Gruppen werden aus diesem Grund bis zum Anschlag gefüllt.
- Einige Teilnehmer werden aufgrund von festen Grenzen der Gruppengrößen nicht eingeteilt und müssen dann manuell in die sinnvollen Gruppen eingetragen werden.

Betrachten wir das Beispiel von acht Teilnehmern, die in vier Gruppen eingeteilt werden müssen. Ihre Bewerbungen sind in der Tabelle 3.1 zu sehen.

Damit die Tabelle übersichtlicher wird, wurden keine Benutzerkennungen angegeben: Die einzigartigen Losnummern können als Identifikatoren verwendet werden.

Losnummer	Priorität	Wunsch
1	1	1
1	2	4
1	3	3
2	1	1
2	2	4
2	3	2
3	1	2
3	2	3
3	3	4
4	1	1
4	2	2
4	3	3

Losnummer	Priorität	Wunsch
5	1	1
5	2	2
5	3	4
6	1	2
6	2	1
6	3	4
7	1	4
7	2	2
7	3	1
8	1	4
8	2	2
8	3	1

Tabelle 3.1: Beispielbewerbungen für eine Einteilung

Eine gleichmäßige Verteilung der Teilnehmer in Gruppen ist in diesem Fall möglich. Es gibt mehrere gleichwertige optimale Einteilungen. Eine davon ist:

$$1 \to \{2,5\} \quad 2 \to \{6,4\} \quad 3 \to \{3,1\} \quad 4 \to \{7,8\}$$

Dabei werden die ersten Wünsche der Mitglieder mit den Losnummern 2 und 5 mit der Einteilung in die Gruppe 1 erfüllt. Die Mitglieder mit den Losnummern 6 und 4 kommen in die Gruppe 2 (1. Wunsch + 2. Wunsch), usw.

Die Heuristik der Megadigitale erreicht eine solche Verteilung leider nicht:

Bei der Vorgabe der durchschnittlichen Gruppengröße als Gruppengrößenmaximum und unter Verwendung von drei Wünschen für die Einteilung wird der Teilnehmer mit der Losnummer 1 gemäß seinem Wunsch in die erste Gruppe eingeteilt. Dann wird der Teilnehmer 2 betrachtet, der ebenfalls seinem ersten Wunsch nach in die erste Gruppe kommt... Zu der Zeit, wenn die 7. und 8. Teilnehmer dran kommen, werden alle für sie in Frage kommenden Gruppen bereits je zwei Teilnehmer beherbergen. Daher werden sie gar nicht eingeteilt. Die Gruppe 3 bleibt zudem leer:

$$1 \to \{1,2\} \quad 2 \to \{3,4\} \quad 3 \to \emptyset \quad 4 \to \{5,6\} \quad \lightning \to \{7,8\}$$

Um zu vermeiden, dass einige Interessenten nicht eingeteilt werden, könnte man die maximale Teilnehmeranzahl für jede Gruppe von 2 auf 4 erhöhen. Das Problem dabei ist die riesige Diskrepanz zwischen den Gruppengrößen: Die Gruppe 1 ist doppelt so groß wie die Gruppen 2 und 4.
Die Gruppe 3 ist immer noch leer:

$$1 \to \{1,2,4,5\} \quad 2 \to \{3,6\} \quad 3 \to \emptyset \quad 4 \to \{7,8\}$$

Bei der praktischen Verwendung der Heuristik von Megadigitale treffen die Kursleiter oft die Entscheidung, dem Algorithmus zu verbieten, die 2. und 3. Wünsche zu verwenden. Das wird aus dem Grund gemacht, dass sonst die Teilnehmer mit kleinen Losnummern ihre letzten Wünsche mit den Plätzen erfüllt bekommen, die andere Teilnehmer als ihre ersten Wünsche angegeben haben.

Bei Verordnung geringerer Gruppenkapazitäten werden einige Interessenten in diesem Fall gar nicht eingeteilt und bei größeren Gruppenkapazitäten werden die Diskrepanzen zwischen den Gruppen riesig. Das Beispiel vom ersten (Verteilung nach 1. Wünschen, maximale Gruppengröße 2) sehen wir hier:

$$1 \to \{1,2\} \quad 2 \to \{3,6\} \quad 3 \to \emptyset \quad 4 \to \{7,8\} \quad \lightning \to \{4,5\}$$

Um die Unterschiede zu veranschaulichen, wurden die erwähnten Einteilungen zum Vergleich auf der Abbildung 3.2 nebeneinander geplottet:

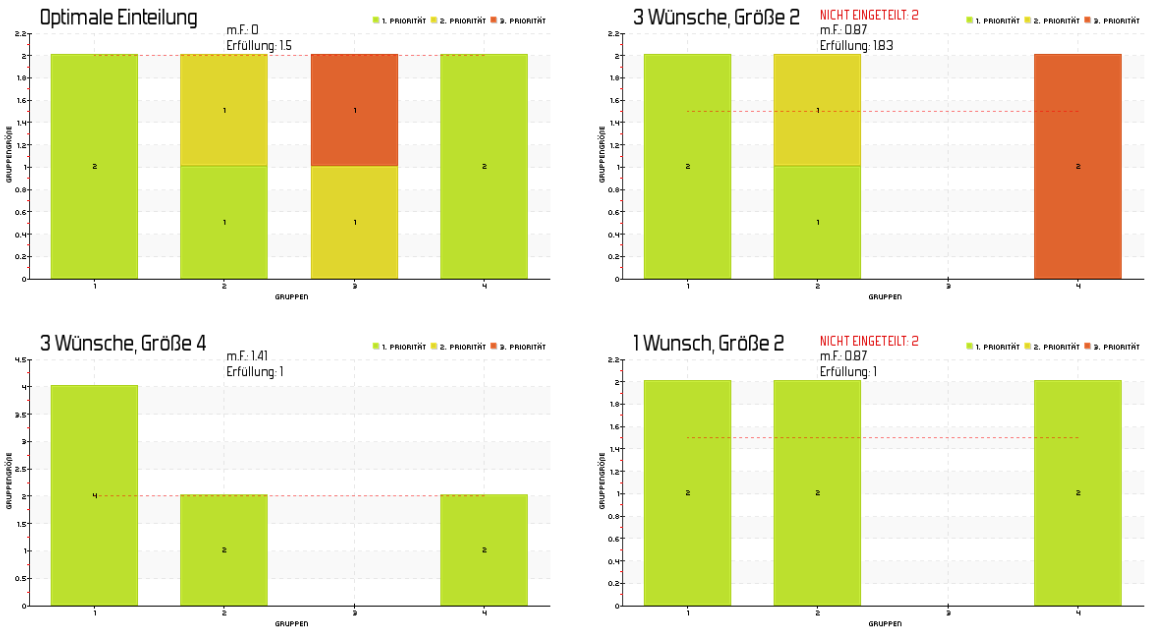


Abbildung 3.2: Unterschiedliche Einteilungen der Heuristik von Megadigitale im Vergleich zur optimalen Lösung

- Die optimale Einteilung oben links.
- Die Lösung mit der Verwendung von drei Wünschen und der maximalen Gruppengröße zwei oben rechts.
- Bei drei betrachteten Wünschen und maximaler Gruppengröße vier unten links.
- Bei einem Wunsch und der maximalen Gruppengröße von zwei unten rechts.

Auf allen drei Graphen der heuristischen Einteilung sind die merkbaren Abweichungen einzelner Gruppengrößen vom durchschnittlichen Wert (mit der rot gestrichelten Linie gezeichnet) ersichtlich. Solche Abweichungen bedeuten in der Praxis überfüllte Räume, in denen Teilnehmer keine Sitzplätze bekommen, sowie auch die Existenz fast leerer Gruppen.

Die nicht automatisch eingeteilten Teilnehmer müssen im Falle der Verwendung dieses Systems anschließend manuell eingeteilt werden, was insbesondere bei großen Veranstaltungen einen enormen Aufwand für die Kursleiter bedeutet, der in einer geringen Zeitspanne auf sie zukommt.

Die vorliegenden Ergebnisse zeigen, dass diese schnelle Heuristik zwar in der Kombination mit dem menschlichen Einsatz, mehrmaliger Neuberechnung der Einteilung mit dem Einsatz diverser Parameter und der nachfolgender manuellen Bearbeitung der Einteilung verwendet werden kann, aber keine Lösung zum automatisierten Finden guter Einteilungen ist.

3.2 Gruppeneinteilungsverfahren

Der Algorithmus ist bei einem Gruppeneinteilungssystem auch nicht alles: Es gibt viele weitere Punkte, die bei der Entwicklung eines solchen Systems beachtet werden müssen, um ein gutes Ergebnis zu erzielen. Um die Wichtigsten davon herauszukristallisieren, müssen die aktuell in Verwendung gezogenen Lösungen betrachtet, analysiert und miteinander verglichen werden.

Da Informationen bezüglich der Gruppeneinteilungssysteme aller Universitäten nicht immer veröffentlicht werden, habe ich mich bei der Analyse auf die Gruppeneinteilungssysteme beschränkt, zu denen ich Zugriff hatte bzw. zu deren Entwicklern ich Kontakt aufnehmen konnte.

Bei der Gruppeneinteilung für die Tutorien werden mehrere Lösungen eingesetzt:

3.2.1 Anmeldelisten (Papierform)

Anmeldelisten waren historisch gesehen die erste Lösung des Gruppeneinteilungsproblems für die Universitäten. Dabei wird für jede Gruppe einer Veranstaltung jeweils eine Tabelle erstellt. In dieser Tabelle gibt es Zeilen, wo Studierende, die in eine jeweilige Gruppe kommen möchten, ihre Namen eintragen müssen. Wenn alle Zeilen mit Namen versehen sind, kann sich in diese Gruppe keiner mehr eintragen. Das entspricht dem sogenannten *First-come, first-served*-Prinzip: Wer als Erster sich in die Gruppe einträgt, bekommt einen Platz.

Der Vorteil einer solchen Einteilung besteht darin, dass die Organisation relativ einfach ist.

Die offensichtlichsten Nachteile sind:

- Man muss persönlich am Ort der Einteilung anwesend sein, um seinen Namen einzutragen.
  Die Studierenden, die während des Studiums arbeiten, können nicht bei allen Terminen erscheinen; falls die Gruppeneinteilung nicht extra (z.B. im Papier- oder Digitalform) angekündigt wurde, ist damit zu rechnen, dass Einige von ihnen sie verpassen.
- Bei den Terminen, wo die Anzahl der Interessenten größer als die Anzahl der Plätze in der Gruppe ist, kommen diejenige in die Gruppe, die sich zuerst eintragen.
  Dementsprechend bekommt nicht derjenige einen Platz in der überfüllten Gruppe, der zu einem anderen Zeitpunkt nicht kommen kann, sondern derjenige, der den anderen Interessierten körperlich überlegen ist.
- Man kann schlecht seine Gruppe ändern, wenn man feststellt, das in Eile der falsche Termin gewählt wurde.
- Wenn alle passenden Gruppen überfüllt sind, bekommen die Teilnehmer, die zu keinem sonstigem Termin kommen können, keinen Platz und können somit an der Veranstaltung nicht ordentlich teilnehmen.

- Die Einteilung erfolgt dezentral. Falls man vergessen hat, in welche Gruppe man sich eingetragen hat, ist das ein Problem.
- Fremde Menschen können sich als Studierende eintragen.
- Die verfügbaren Gruppen der anderen Kurse werden nicht zum Vergleich dargestellt. Es kann daher vorkommen, dass man mit einer „blinden“ Auswahl einer Gruppe des Kurses A sich alle Möglichkeiten gesperrt hat, am Übungsbetrieb des Kurses B teilzunehmen.

Aufgrund vieler Probleme wurden anstatt der Anmeldelisten in Papierform in einigen Veranstaltungen und Universitäten andere Lösungen eingesetzt.

3.2.2 Anmeldelisten (Digital)

Da rechnergestützte Lösungen in der letzten Zeit populär geworden sind, erschienen digitale Anmeldelisten als eine Alternative zum veralteten Prototyp. Diese sind normalerweise über Internet für die Teilnehmer und Kursleiter erreichbar und werden in der Regel mehrere Tage für die Wahl des Termins angeboten, damit alle, die aus irgendwelchen Gründen Internet- oder Stromausfall haben sollten, trotzdem die Möglichkeit haben, in eine Gruppe zu kommen.

Die digitalen Anmeldelisten bieten viele Vorteile gegenüber der Papierform:

- Sie werden oft zentral und nicht nur für einzelne Kurse angeboten. Damit bekommt man einen Überblick über alle im Anmeldesystem aufgelisteten Kurse und kann sich in Ruhe überlegen, welche Termine sich schneiden könnten, falls man eine bestimmte Auswahl trifft.
- Sie bieten oft die Möglichkeit der Wunschänderung, falls man sich nach einer Überlegung für eine andere Gruppe entschieden hat.
- Damit kann man recht einfach online anschauen, in welcher Gruppe man ist, wenn man das vergessen hat.

Allerdings basieren sie immer noch auf dem First-Come First-Served-Prinzip und bieten somit denjenigen, die beispielsweise aufgrund schlechter Internetverbindung oder der Abwesenheit eines guten Smartphones zum Zeitpunkt der Freischaltung der Anmeldung sich nicht in den ersten Minuten eingetragen haben, wegen der Überfüllung nur die unpopulären - frei gebliebenen - Gruppen an.

Dieses Gruppeneinteilungsverfahren wird aufgrund von Einfachheit der Implementierung von vielen Systemen verwendet. Das sind beispielsweise:

1. Das Gruppeneinteilungssystem des Instituts für Mathematik der Goethe Universität Frankfurt am Main.
2. OLAT[6]: Die Open Source Lernplattform, welche von vielen Universitäten verwendet wird.

[6] steht für **O**nline **L**earning **a**nd **T**raining

3. QIS/LSF[7]: Ein Hochschulverwaltungssystem, das von vielen Universitäten verwendet wird und Anmeldelisten als eine der drei Gruppeneinteilungsarten anbietet.

3.2.3 Losverfahren

Bei einem Losverfahren melden sich Interessenten, genauso wie beim Verfahren der Anmeldelisten, jeweils für die Wunschgruppe ihrer Wahl. Am Ende der Anmeldezeit (in der Regel sind das eine bis zwei Wochen) werden die Gruppen mit allen Interessenten gefüllt, wo die Anzahl der Bewerbungen nicht größer als die Anzahl der Plätze ist. Bei den Gruppen, wo die Anzahl der Plätze es nicht erlauben würde, alle aufzunehmen, wird auf Grundlage des Zufallsprinzips entschieden, wer in die Gruppe kommt und wer nicht.

Der wichtigste Vorteil dieses Verfahrens gegenüber den Anmeldelisten ist die Unabhängigkeit der Ergebnissen von den die Gerechtigkeit der Platzvergabe verletzenden Faktoren (wie z.B. ein besseres Smartphone).

Zu der Liste der Nachteile kommt ein weiterer hinzu: Bei diesem Verfahren wird einem potenziellen Teilnehmer erst am Ende der Anmeldezeit bekannt, ob er in die Wunschgruppe kommt. Wenn nicht, bekommt er keinen Platz und kann daher nicht am Kurs weiter ordentlich teilnehmen.

Dieses Gruppeneinteilungsverfahren ist wegen diesem Nachteil nicht sehr beliebt, wird allerdings beispielsweise vom Hochschulverwaltungssystem QIS/LSF als eine der drei Gruppeneinteilungsarten angeboten.

3.2.4 Priorisierte Einteilungsverfahren

Die priorisierten Einteilungsverfahren verwenden das System von mehreren Wünschen und dazugehörigen Prioritäten, um möglichst alle Interessenten in Gruppen zu verteilen.

Dabei wählt jeder potenzielle Teilnehmer nicht nur einen am besten passenden Termin, sondern mehrere (in der Regel drei) Termine aus. Jeder der Wunschtermine muss priorisiert werden. Die Anzahl der Prioritäten ist gleich der Anzahl der auszuwählenden Termine. Höhere Prioritäten werden nach Möglichkeit zuerst erfüllt, erst dann kommen die anderen.

Betrachten wir das Beispiel eines Teilnehmers, dem aus den angebotenen zehn Terminen nur vier passen könnten. Daraus darf er drei auswählen. Der Teilnehmer würde am liebsten in die Gruppe 3 kommen. Wenn das nicht klappt, würde er aus allen anderen die Gruppe 2 auswählen. Sollte auch das nicht klappen, nimmt er die Gruppe 1. In diesem Fall muss dieser Teilnehmer für die Gruppe 3 die Priorität 1 (hoher Vorrang) auswählen, für die Gruppe 2 die Priorität 2 (mittlerer Vorrang) und für die Gruppe 1 die Priorität 3 (niedriger Vorrang).

[7] steht für: **Q**ualitätssteigerung der Hochschulverwaltung im **I**nternet durch **S**elbstbedienung / **L**ehre, **S**tudium, **F**orschung

Falls die Anzahl der Bewerbungen für eine Gruppe ihre Größe nicht übertrifft, werden in diese Gruppe alle eingeteilt, die sie als ihren ersten Wunsch angegeben haben. Im anderen Fall ermöglichen mehrere weitere von Teilnehmern angegebene für sie in Frage kommende Termine dem System, einen anderen Platz für diejenige zu finden, die bei den beiden oben erwähnten Gruppeneinteilungsverfahren in gar keine Gruppen eingeteilt wären.

Die Vorteile dieses Verfahrens sind:

1. Unabhängigkeit der Ergebnisse von den die Gerechtigkeit der Platzvergabe verletzenden Faktoren (wie z.B. ein besseres Smartphone).
2. Die Möglichkeit der gleichmäßigeren Verteilung als bei Anmeldelisten- und Losverfahren.

Der einzige Nachteil ist die Schwierigkeit der Umsetzung, die unmittelbar mit der Komplexität der Aufgabe der Gruppeneinteilung mit mehreren Wünschen, die bei den theoretischen Grundlagen beschrieben wurde, im Zusammenhang steht.

Aus diesen Gründen ist dieses Verfahren am besten geeignet, um die Teilnehmer eines Kurses in möglichst gleich große Gruppen einzuteilen und dabei jedem einen Termin zu geben, der ihm zeitlich passen würde.

Priorisierte Einteilungsverfahren werden aufgrund der Ermöglichung gleichmäßigerer Verteilung in neueren rechnergestützten Gruppeneinteilungssystemen eingesetzt. Manuelle Einteilung nach diesem Prinzip ist wegen der Komplexität des Verfahrens leider kaum möglich. Das Gruppeneinteilungsskript von Megadigitale, das Campus-Verwaltungssystem CampusNet, das heutzutage von 33 Hochschulen verwendet wird sowie auch das Gruppeneinteilungssystem der TU Kaiserslautern verwenden dieses Verfahren für die Berechnung der guten Einteilungen. Das Hochschulverwaltungssystem QIS/LSF bietet dieses Verfahren außerdem als eine der drei Gruppeneinteilungsarten an.

3.3 Aktuelle Gruppeneinteilungssysteme im Vergleich

Ein gutes Gruppeneinteilungssystem muss die Stärken aller bereits existierenden Lösungen übernehmen und ihre Schwächen nicht erben.

Für die Analyse der Stärken und Schwächen können wir die erwähnten Gruppeneinteilungssysteme miteinander vergleichen. Einige Vergleichskriterien können wir dabei der Analyse der Einteilungsverfahren entnehmen. Einige andere aus der Kritik, die an diese Systeme oft geäußert wird.

Die folgenden Vergleichskriterien werden betrachtet:

- *Gleiche Chancen* gibt an, ob alle, die das System verwenden, gleiche Chancen auf die Plätze in überpopulären Gruppen haben. Im einem System, wo diese Eigenschaft fehlt, hingegen kann man auf eine bestimmte Art und Weise ohne Beachtung der Interessen von anderen Teilnehmern und existierenden Härtefälle von eigener Seite einen Platz sich sichern: z.B. durch ein schnelleres Smartphone und die folgende schnellere Anmeldung in einem First-Come-First-Serve-basierten System.
  Dieses Kriterium ist äußerst wichtig, um allen Teilnehmern bei der Zuteilung möglichst gleiche Chancen zu gewährleisten.
- *Einteilung Aller* zeigt, ob bei der Einteilung einige Teilnehmer ohne Plätze bleiben können.
  Dieses Kriterium ist ebenfalls eins der wichtigsten, da z.B. in den Pflichtveranstaltungen allen Teilnehmern die Möglichkeit der Teilnahme an der Veranstaltung garantiert werden muss.
- *Wunschänderung* steht für die Möglichkeit, den bereits eingetragenen Wunschtermin vor der Einteilung zu ändern.
  Diese Eigenschaft ist sehr wichtig, da damit falsche Einteilungen anhand veralteter bzw. falsch eingetragener Zeitangaben zum großen Teil vermieden werden.
- *Zuteilungsänderung* bedeutet für die Kursleiter die Möglichkeit, die Ergebnisse der automatischen Einteilung zu verändern, wenn sie nicht zufriedenstellend sind oder wenn von der Seite der Teilnehmer Härtefälle bekannt werden, aufgrund dessen Verschiebungen der Eingeteilten von einer Gruppe in die andere notwendig sind.
- Der Parameter *aktuelle Situation* bezeichnet die Funktion, bei der Studierende bzw. Kursleiter den aktuellen Stand der Anmeldung und die potenziellen Gruppengrößen sehen können, damit sie nach einer kurzen Analyse sich beispielsweise für eine im Voraus bekannt überfüllte Gruppe gar nicht bewerben.
- *Vorläufige Einteilung* ermöglicht dem Kursleiter, die Einteilung vor dem Ende der Bewerbungsfrist vorläufig durchzuführen, sodass die Ergebnisse nur für ihn und nicht für die Teilnehmer sichtbar sind.
  Das kann in den Fällen sehr hilfreich sein, wo einige angebotene Termine den Teilnehmern sehr

schlecht passen, um in Voraus erkennen zu können, dass bestimmte Gruppen in einem solchen Fall entweder einen anderen Termin bekommen sollten oder bereits vor der Einteilung gelöscht werden können, da sie sowieso nicht im sinngemäßen Umfang genutzt würden.

- Die vom System beachtete Eventualität *paralleler Veranstaltungen* ermöglicht die Vermeidung der Vergabe mehrerer gleichzeitiger Termine für eine Person, die selbstverständlich nicht gleichzeitig besucht werden können.
- Die Rücksichtnahme auf die *parallelen Gruppen* einer Veranstaltung bietet einen am gleichen Campus zur gleichen Zeit angebotenen Termin als eine Alternative zum ausgewählten Termin.
- Die *Referenzfunktion* ermöglicht die Angabe einer Referenz auf einen anderen Teilnehmer mit dem man in der gleichen Gruppen landen möchte.
  Bei der Einteilung wird der Referent gemäß seinen Bewerbungen einer Gruppe zugewiesen. Dann wird das Programm den Referierenden in die gleiche Gruppe platzieren.
- Die *Überscheidungsdarstellung* ist eine wünschenswerte Funktion, die zeigt, welche aktuell im System von einer Person eingetragene Termine sich überschneiden, damit die daraus folgenden Probleme für diese Person noch vor der Einteilung ersichtlich sind.
  Sie ist allerdings weniger wichtig, als alle oben erwähnten Kriterien.

Die Gruppeneinteilungssysteme, die betrachtet und verglichen werden, sind:

- Das Campus-Verwaltungssystem *CampusNet* von Datenlotsen[8], als eine *FCFS*-Lösung und als ein System, das *Prioritäten*verfahren unterstützt.
- Die Lösung des Instituts für Mathematik (*IfM*) der *Goethe Uni*versität Frankfurt.
- Das Skript der *Megadigitale*.
- Die Lernplattform *OLAT*.
- Das Gruppeneinteilungssystem des Instituts für Mathematik der *TU Kaiserslautern*.
- Das Hochschulverwaltungssystem *QIS/LSF* in drei Aspekten: Als eine First-Come-First-Serve (*FCFS*)-Lösung, als eine Losverfahrenlösung und als ein System, das *Prioritäten*verfahren unterstützt.

Die Tabelle 3.2 zeigt die Zusammenfassung der Ergebnisse vom Vergleich der Einteilungsverfahren aller genannten Gruppeneinteilungssysteme. Falls ein Kriterium erfüllt wird, steht ein ✓-Zeichen. Sollte eine Eigenschaft nur im Falle eines erheblichen manuellen Einsatzes seitens der Kursverwalter erfüllt werden können, steht ein Ⓜ-Zeichen. Im sonstigen Fall wird die der Eigenschaft entsprechende Zelle zwecks bessere Lesbarkeit leer gelassen.

Die Kandidaten des Vergleiches werden in drei Kategorien mit einer horizontalen Linie geteilt. In der

[8] http://www.datenlotsen.de

	gleiche Chancen	alle eingeteilt	Wunschänderung	Zuteilungsänderung	aktuelle Situation	vorläufige Einteilung	parallele Veranstaltungen	parallele Gruppen	Referenzfunktion	Überschneiddarstellung
CampusNet (FCFS)			✓	✓	✓	✓		✓		
IfM Goethe Uni			✓		✓	✓				
OLAT			✓		✓	✓				
QIS/LSF (FCFS)			✓		✓	✓				
QIS/LSF (Losverfahren)	✓		✓		✓					
CampusNet (Prioritäten)	✓	Ⓜ	✓	✓	✓			✓		
Megadigitale	✓	Ⓜ	Ⓜ	✓		✓				
TU Kaiserslautern	✓	Ⓜ	✓	✓	✓		✓	✓	✓	
QIS/LSF (Prioritäten)	✓		✓							

Tabelle 3.2: Gruppeneinteilungssysteme im Vergleich
✓ steht für *vorhanden*, Ⓜ für *manuell möglich*

obersten Kategorie finden sich alle FCFS-basierten Gruppeneinteilungssysteme. In der zweiten Kategorie ist die einzige betrachtete Lösung, die die Einteilung via Losverfahren verwendet. Die unterste Kategorie enthält die Systeme, die ein priorisiertes Gruppeneinteilungsverfahren verwenden.

Wie sich anhand der Tabelle erkennen lässt, weisen alle First-Come-First-Serve-basierten Lösungen die gleichen Schwächen bei den ersten (und relevantesten) zwei Kriterien auf.

Die Losverfahrenlösung von QIS/LSF ist um einiges besser, da im Gegenteil zu den FCFS-Lösungen *gleiche Chancen* garantiert werden. Diese Lösung kann allerdings verfahrenstechnisch die Einteilung aller Interessenten nicht gewährleisten.

Die Prioritätenlösung von QIS/LSF ist mit Abstand die schlechteste von allen Prioritätenlösungen, vor allem aufgrund der Tatsache, dass nicht alle eingeteilt werden und die Anmeldungen der Nicht-Eingeteilten verfallen, sodass der Kursleiter keine Möglichkeit hat, den Prozess im Nachhinein zu verfolgen, um den Versuch zu tätigen manuell eine vernünftige Lösung zu finden.

Parallele Gruppen der Veranstaltungen werden bei der Einteilung nur von drei Systemen in Betracht gezogen, was gute und sinnvolle Lösungen bei anderen Systemen unmöglich macht: Die parallelen Gruppen werden von den Veranstaltern normalerweise extra zu der für die Teilnehmer am besten passenden Zeit angeboten, damit insgesamt mehr Plätze zu dieser Zeit zur Verfügung stehen. Die beiden Gruppen werden in solchem Fall vom Veranstalter als Äquivalente empfunden, was ein gutes Gruppeneinteilungssystem gleich empfinden muss.

Die Darstellungsmöglichkeiten bei der Terminüberschneidung sind bei allen betrachteten Systemen

nicht vertreten. Obwohl solch eine Funktionalität nicht unbedingt essentiell für die eigentliche Gruppeneinteilung ist, hilft sie enorm bei der Terminauswahl und sollte aus diesem Grund von einem guten Gruppeneinteilungssystem angeboten werden.

Anhand der Tabelle 3.2 sieht man, dass das Gruppeneinteilungssystem *TU Kaiserslautern* am meisten punktet. Auch hier wurden allerdings leider nicht alle in Frage kommenden Anforderungen erfüllt.

Die *vorläufige Einteilungen* können von diesem System verfahrenstechnisch (aufgrund der Verwendung eines langsamen Algorithmus) nicht angeboten werden, was die Analyse der Notwendigkeit der Terminänderung vor der Einteilung sehr schwierig macht.

Dieses Gruppeneinteilungssystem bietet außerdem bei mehreren parallelen Gruppen jeder Veranstaltung die Termine und nicht die Gruppen selbst für die Auswahl bei der Anmeldung an. Der Vorteil dieser Vorgehensweise gegenüber der Gruppenauswahl ist eine bessere Übersicht bei einer kleineren Anzahl der Wahlmöglichkeiten, die die Auswahl für die Teilnehmer vereinfacht und damit Usability verbessert. Der Nachteil wird offensichtlich, wenn man darüber nachdenkt, dass unterschiedliche Übungen nicht nur in den unmittelbar in der Nähe voneinander stehenden Räumen angeboten werden, sondern auch - bei großen Hochschulen - in unterschiedlicher Campus angeboten werden. Dann können diese nicht als Alternativen zueinander betrachtet werden, da die Dauer der Reise zwischen den Campus erheblich sein kann und somit für den Anfahrt zusätzliche Zeit eingeplant werden muss.

Kapitel 4

Neue Lösungsansätze

Unter Beachtung der existierenden Algorithmen für das Problem der Gruppeneinteilung muss für ein gutes Gruppeneinteilungssystem die bestmögliche Option gewählt, oder eine neue Lösung gefunden werden, die eventuell bessere Ergebnisse erzielt.

Außerdem muss bedacht werden, in welcher Form sich die Standardanforderungen an ein solches System, die von den bereits existierenden Systemen erfüllt werden, umsetzen lassen. Zusätzliche Anforderungen und Entwicklungswünsche müssen gesammelt werden, um auch die momentan vorstellbaren Schwachstellen in Zukunft vermeiden zu können.

Alle bereits im Einsatzgebiet verwendeten Verwaltungssysteme, welche die Kursverwaltung koordinieren, müssen auf die Möglichkeiten der Synchronisierung, Export und Import der Kurs- und Teilnehmerinformationen untersucht werden, um die Redundanz der Daten zu vermeiden und die gleichzeitige Bedienung mehrerer Systeme zu vereinfachen.

4.1 Algorithmus

4.1.1 Optimierungskriterien

Wie in Kapitel 2.3 bereits erwähnt wurde, existieren bei der Gruppeneinteilung zwei Optimierungskriterien: gleichmäßige Verteilung der Teilnehmer in die Gruppen und die Erfüllung der Wünsche der Teilnehmer mit möglichst hohen Prioritäten.

Da nicht nach beiden Kriterien gleichzeitig optimiert werden kann, werden in dieser Arbeit bei einem der Faktoren Güteklassen definiert.

Die Definition von Güteklassen für die Gleichmäßigkeit der Verteilung ist problematisch, da abhängig von der Problemstellung die Abweichungen der Gruppengrößen von Durchschnitt bei unterschiedlichen Einteilungen sehr stark variieren können. Die Gleichmäßigkeit als eine kontinuierliche Größe

kann man als die **Standardabweichung der Gruppengrößen** definieren:

$$\sigma := \sqrt{\frac{1}{n}\sum_{i=1}^{n}(g_i - \bar{g})^2}$$

Dabei bezeichnen:

- n die Gruppenanzahl
- g_i die Größe der Gruppe i
- $\bar{g}$ die durchschnittliche Gruppengröße

Damit werden größere Abweichungen, die in der Praxis auf jeden Fall vermieden werden sollten, deutlicher im Endergebnis ausgeprägt als die kleineren Abweichungen von einem oder zwei Teilnehmern, die bei den Gruppengrößen ab zehn Teilnehmer akzeptabel sind. Bei diesem Maß für Gleichmäßigkeit sind überdurchschnittlich große Gruppen für die Qualität der Verteilung genauso schlecht wie unterdurchschnittlich kleine. Solche Größenabweichungen kompensieren einander auf keinen Fall. Dies spiegelt sich in der Formel wieder, indem Quadrierung die entsprechenden Vorzeichen entfernt.

Die bestmögliche Gleichmäßigkeit wird genau dann erreicht, wenn die Abweichung $\sigma = 0$ beträgt. Je größer σ wird, desto ungleichmäßiger ist die Verteilung der Teilnehmer auf die Gruppen und desto schlechter wird das Ergebnis.

Im Bereich der Erfüllung der Wünsche ist die Aufgabe der Güteklassendefinition deutlich einfacher als bei der Abweichung:

Definieren wir zuerst den Maß für die Erfüllung der Terminwünsche bei einer Einteilung als den **Durchschnitt aller erfüllten Wünsche**:

$$W := \frac{1}{n}\sum_{i=1}^{n} w_i$$

Dabei bezeichnen:

- n die Teilnehmeranzahl
- w_i den durch die Einteilung erfüllten Wunsch des Teilnehmers i (1, 2 oder 3)

Bei 154 Teilnehmern mit dem erfüllten 1. Wunsch, 45 Teilnehmern mit dem erfüllten 2. Wunsch und 13 Teilnehmern mit dem erfüllten 3. Wunsch ergibt die Formel beispielsweise:

$$\frac{154 \cdot 1 + 45 \cdot 2 + 13 \cdot 3}{154 + 45 + 13} \approx 1.33$$

Je näher die *Erfüllung* an eins ist, desto besser die Qualität der Erfüllung. Die schlechteste mögliche *Erfüllung* $W = 3$ bedeutet, dass mit der aktuellen Einteilung ausschließlich dritte Wünsche erfüllt wurden.

Es ist im Voraus bekannt, dass bei der Einteilung nur anhand der ersten Wünsche die *Erfüllung* im-

mer gleich 1 wird, da ein Durchschnitt von Einsen berechnet wird. Dabei ist allerdings zu erwarten, dass aufgrund weniger beachteten Bewerbungen die beste *Gleichmäßigkeit* niedriger wird, als bei der Einteilung anhand der ersten und der zweiten Wünsche. Bei der Einteilung mittels aller drei Wünsche kann man auf Kosten einer noch schlechteren Erfüllung die beste *Gleichmäßigkeit* erreichen.

Daher können wir drei Güteklassen definieren: *1 Wunsch*, *2 Wünsche* und *3 Wünsche* und dann in jeder Klasse nach Gleichmäßigkeit optimieren.

4.1.2 Anforderungen an den Algorithmus

Bei der Analyse der existierenden Algorithmenarten, die bei der Lösung des Gruppeneinteilungsproblems in Frage kommen haben wir das Folgende festgestellt:

- Die exakten Methoden können aufgrund zu langer Rechendauer für die relevanten Problemgrößen leider nicht verwendet werden.
  Das bedeutet, dass wir nicht die optimalen Lösungen, sondern die guten Lösungen als Ergebnis erwarten müssen.
- Die Verwendung der allgemeinen Approximationsmethode Branch-and-Cut wird bereits in der Praxis verwendet und liefert gute Ergebnisse. Die Berechnungszeiten sind für kleine und mittelgroße Bewerbungszahlen akzeptabel, für die größeren Aufgaben müsste ein schnellerer Algorithmus verwendet werden.
- Metaheuristiken wie Genetische Algorithmen oder Tabusuche können zwar bei der Lösung des Problems verwendet werden, erzielen jedoch gegenüber Branch-and-Cut keine Vorteile, liefern dabei die Lösungen, die stark vom Optimum abweichen können.
- Die für das Problem der Gruppeneinteilung entwickelte Heuristik von Megadigitale hat zwar eine sehr kurze Laufzeit und gute Einstellbarkeit, eignet sich allerdings aufgrund von suboptimalen Ergebnissen nur für die vorläufige Einteilung, die durch manuelle Bearbeitung komplettiert werden muss, damit ein Ergebnis, das nah am Optimum ist, erreicht werden kann.
- Fazit: Entwicklung einer neuen speziellen Heuristik ist sinnvoll.

Welche Anforderungen werden an den Algorithmus gestellt?

- Für die ersten n Wünsche muss eine Gruppenbelegung gefunden werden, bei der die angebotenen Gruppen möglichst gleichmäßig befüllt werden, und kein Teilnehmer einer Gruppe zugeteilt wird, die ihm nicht passt.
- Ein *priorisierter Einteilungsverfahren* muss verwendet werden.
- Schnelle Einteilung, damit die *vorläufigen Einteilungen* mehrmals auf Wunsch berechnet werden können.

- Transparenz und Einfachheit des Einteilungsverfahrens. Dies soll den Teilnehmern die Analyse des richtigen Verhaltens bei der Auswahl der Termine ermöglichen. So wird das Prozedere der Anmeldung von den Teilnehmern gern durchlaufen, da ihnen ein Ausblick auf den Ausgang offengelegt wird.

4.1.3 Ideen

Die folgenden Ideen sind dabei zielführend und führen in die Richtung des optimalen Ergebnisses:

- Wenn eine Gruppe so wenige Bewerbungen hat, dass im Falle der Erfüllung aller vorliegenden Bewerbungen unter Beachtung aller Prioritäten, diese die Durchschnittsgröße dennoch nicht übersteigt, wird sie mit allen vorliegenden Bewerbungen gefüllt und dann nicht mehr betrachtet.

  Dies kann in linearer Zeit erledigt werden, abhängig von der Bewerbungsanzahl. Eine solche Füllung verkleinert die Anzahl der Gruppen um eins und die Anzahl der noch zu betrachtenden Bewerbungen um die Größe der auf solche Art und Weise entstandenen Gruppen. Damit wird das Gesamtproblem deutlich vereinfacht und die betrachteten Gruppen möglichst groß aber nicht zu groß.

- Alle Gruppen sollen *parallel* mit Teilnehmern gefüllt werden, damit sie möglichst gleich sind. Dafür sollen Teilnehmer einzeln in die Gruppen eingeteilt werden.

- Der nächste Teilnehmer kommt in eine der Gruppen mit der aktuell geringsten Größe, damit die Gruppengrößen parallel steigen.

- Dabei muss ein Wunsch mit möglichst hoher Priorität erfüllt werden, damit die *Erfüllung* möglichst nah an eins liegt.

4.1.4 Neue Heuristik

Diese Ideen werden mit dem folgenden Algorithmus umgesetzt:

1. Bei der Anmeldung bekommt jeder Teilnehmer eine zufällig generierte Losnummer.

   Damit die Anordnung eindeutig ist, verhindert der Algorithmus, dass gleiche Losnummern doppelt vergeben werden, indem vor der Vergabe einer neuen Nummer, zunächst die Liste der bereits existierenden Losnummern nach ihr durchforstet wird.

2. Die Kennung des aktuellen Kurses wird in der Variable `$this->kurs_id` gespeichert.

3. Alle Bewerbungen für den aktuellen Kurs werden aus der Datenbank geladen, nach der Priorität und dann nach der Losnummer sortiert.

```
1  // get a part reference copy of applications
2  $registrations = array_slice($this->applications, 0, $maxPriority, true);
3  $divisions = array();
4  $currUser = 0;
5  $localGroupNumbers = $this->groupNumbers;
6  $groupsToFill = $localGroupNumbers;
7  $wishesEnum = array(1 => "1wunsch", 2 => "2wuensche", 3 => "3wuensche");
```

Listing 4.1: Die neue Heuristik: Initialisierung

```
8  // get the potential max group sizes
9  $potSizes = array();
10 foreach($registrations as $registrationsSamePriority)
11   foreach($registrationsSamePriority as $registration)
12     $potSizes[ $registration['gruppe'] ]++;
13 $averageGroupSize = $this->usersNumber / $this->groupsNumber;
14
15 $actualSizes = array();
16 foreach($localGroupNumbers as $groupNumber)
17   $actualSizes[$groupNumber] = 0;
```

Listing 4.2: Die neue Heuristik: Potentielle Gruppengrößen

Alle Bewerbungen mit der gleichen Priorität werden im Array `$this->applications[PRIORITÄT]` gespeichert, wobei `PRIORITÄT`$\in \{1,2,3\}$.
Zu jeder Bewerbung gehören die folgenden Angaben:

`benutzer_id`	die Kennung des Benutzers
`studiengang`	der Studiengang des Benutzers
`fachsemester`	das Fachsemester des Benutzers
`gruppe`	die Kennung der Gruppe innerhalb des Kurses
`prioritaet`	die Priorität, mit der sich beworben wird

Diese sind assoziativ für jedes Element von `$this->applications[PRIORITÄT]` ansprechbar.

4. Die Kennungen aller existierenden Gruppen für den gegebenen Kurs werden im Array `$this->groupNumbers` gespeichert. Ihre Anzahl ist mittels `$this->groupsNumber` jederzeit auslesbar.

5. Die Benutzeranzahl wird in der Variable `$this->usersNumber` gespeichert.

6. Da bei jeder Optimierung innerhalb einer Güterklasse der *Erfüllung* nur die ersten n Prioritäten mit $n \in \{1,2,3\}$ für die Einteilung verwendet werden, wird in der Variable `$maxPriority` die Zahl n gespeichert.

```
18 do {
19   // are there unpopular groups?
20   do {
21     $changes = 0;
22     foreach($localGroupNumbers as $groupNumber)
23       if( ($potSizes[$groupNumber] == 0) ||
24       ($potSizes[$groupNumber] + $actualSizes[$groupNumber] <=
             $averageGroupSize) ){
25         if($potSizes[$groupNumber] != 0)// fill the group
26           for($priority = 1; $priority <= $maxPriority; $priority++)
27             foreach($registrations[$priority] as $registration)
28               if($registration['gruppe'] == $groupNumber){
29                 $divisions[] = $this->getDivision($registration,
                       $wishesEnum[$maxPriority]);
30                 $this->delApplicationsOf($registrations, $registration['
                       benutzer_id'], $potSizes);
31                 $actualSizes[$registration['gruppe']]++; // one is inside
32                 $currUser++;
33                 }
34         // exclude the group;
35         array_splice($localGroupNumbers, array_search($groupNumber,
               $localGroupNumbers), 1);
36         array_splice($groupsToFill, array_search($groupNumber,
               $groupsToFill), 1);
37         if(empty($groupsToFill))
38           $groupsToFill = $localGroupNumbers;
39         unset($potSizes[$groupNumber]);
40         $changes++;
41         }
42   } while($changes > 0);
```

Listing 4.3: Die neue Heuristik: Gruppen sofort füllen

7. Dann wird der Einteilungsalgorithmus initialisiert, wie in dem Listing 4.1 gezeigt wird:

   Dabei werden zuerst alle Bewerbungen mit den ersten `$maxPriority` Prioritäten in einer lokalen Variable gespeichert. Variablen für die Speicherung der Zuteilungen, sowie der Anzahl der bisher eingeteilten Benutzer werden initialisiert. Für die Gruppenidentifikatoren wird eine lokale Kopie erstellt, damit Löschen dabei zulässig wird: Wir möchten später die Gruppen, für die es keine Bewerbungen mehr gibt, nicht mehr betrachten. `$groupsToFill` zeigt, welche Gruppen aktuell die kleinsten sind und daher zuerst bearbeitet werden müssen.

```
43  if( $currUser < $this->usersNumber ){
44    $fittestRegistration = array(); // NA
45    // best fulfilled priority found => search complete
46    foreach($registrations as $registrationsSamePriority)
47      foreach($registrationsSamePriority as $registration)
48        if( in_array($registration['gruppe'], $groupsToFill) ){
49          $fittestRegistration = $registration;
50          break 2; // no need to look at the worse solutions
51        }
52
53    // found new member
54    $divisions[] = $this->getDivision($fittestRegistration, $wishesEnum[
          $maxPriority]);
55    $this->delApplicationsOf($registrations, $fittestRegistration['
          benutzer_id'], $potSizes);
56    $actualSizes[$fittestRegistration['gruppe']]++; // one is inside
57    $currUser++;
58
59    // make other groups even
60    array_splice( $groupsToFill, array_search($fittestRegistration['
          gruppe'], $groupsToFill), 1 );
61
62    if(empty($groupsToFill))
63      $groupsToFill = $localGroupNumbers;
64  }
```

Listing 4.4: Die neue Heuristik: Die Einteilung

8. Anschließend werden, wie man im Listing 4.2 sieht, die maximalen potentiellen Gruppengrößen berechnet, indem die Summe aller Bewerbungen für die jeweilige Gruppe (unter Beachtung aller Wünsche) berechnet wird.
   Die durchschnittliche Gruppengröße kann mittels Teilung der Benutzerzahl durch die Gruppenanzahl berechnet werden.
   Die aktuellen Gruppengrößen der Einteilung werden in den Zeilen 16-17 initialisiert.

9. Danach beginnt die Hauptschleife, die in jedem Durchgang mindestens einen Benutzer in eine Gruppe einteilt (Zeilen 18-65, Listings 4.3, 4.4 und 4.5).

   Diese enthält eine andere Schleife (Zeilen 20-42, Listing 4.3), die für alle zu bearbeitenden Gruppen überprüft, ob die Anzahl der vorliegenden Bewerbungen für sie Null beträgt: In diesem Fall muss die Gruppe nicht mehr betrachtet werden.

   Wenn die Summe der Anzahl der bisher in die Gruppe eingeteilten Teilnehmer und der An-

zahl der verbleibenden Bewerbungen für diese Gruppe (d.h. die maximale zu diesem Zeitpunkt erreichbare Gruppengröße) die durchschnittliche Gruppengröße nicht übersteigt, muss diese Gruppe sofort mit allen vorliegenden Bewerbungen gefüllt werden, da sonst deren Anzahl noch kleiner werden kann, was eine stärkere Abweichung von der durchschnittlichen Größe bewirken und damit die *Gleichmäßigkeit* der Verteilung verletzen würde.

Da es möglich ist, dass nach dem Füllen einer Gruppe die Anzahl der Bewerbungen für eine andere Gruppe so gering wird, dass diese ebenfalls so schnell wie möglich gefüllt werden muss, werden in dem Fall, wenn irgendwelche Gruppen gelöscht wurden, alle anderen Gruppen nochmals auf die Notwendigkeit der Füllung bzw. des einfachen Löschens auf der Gruppenliste überprüft.

10. Im Anschluss wird unter den Bewerbungen, die sich auf die Gruppen beziehen, welche momentan am wenigsten Teilnehmer haben (`$groupsToFill`) nach einer der besten Bewerbungen gesucht, wie man im Listing 4.4 erkennen kann.

    Falls alle Benutzer zu diesem Zeitpunkt durch das Füllen von Gruppen bereits eingeteilt wurden, kann man den Rest überspringen und die Ausführung beenden (Zeile 43). Sonst geht es weiter:

    Eine der besten Bewerbungen ist jene, mit der die höchste mögliche Priorität erfüllt werden kann. Da mehrere Bewerbungen mit der gleichen Priorität für die gleiche Gruppe oder auch für unterschiedliche Gruppen mit unterschiedlichen Losnummern existieren können, wird die erste gefundene Bewerbung genommen.

    Da die Liste der Bewerbungen von Anfang an nach Losnummer sortiert war, sind dabei die ersten gefundenen Bewerbungen zufällig auf diesem Platz gelandet. Damit werden *gleiche Chancen* allen Teilnehmern versichert.

    Die beste Bewerbung wird erfüllt, indem der entsprechende Teilnehmer in seine Wunschgruppe eingeteilt wird. Anschließend werden die aktuellen und potentiellen Gruppengrößen angepasst. Weiterhin müssen die Bewerbungen des eingeteilten Benutzers nicht mehr betrachtet werden. Daher werden sie sofort aus der Liste der immer noch vorliegenden Bewerbungen mit der Hilfe der Funktion `$this->delApplicationsOf()` gelöscht.

    Als Ergebnis der Einteilung des aktuellen Benutzers, wächst die Größe der aktuellen Gruppe. Damit wird sie nicht mehr eine der kleinsten und muss bei der Einteilung des nächsten Mitgliedes nicht am Wettbewerb teilnehmen (Zeilen 60-61).

    Falls als Ergebnis die Liste der kleinsten Gruppen leer wird, wird diese mit allen immer noch an der Einteilung teilnehmenden Gruppen gefüllt.

11. Nachdem alle Teilnehmer eingeteilt wurden, hält die Hauptschleife, und das Ergebnis der Einteilung, das in der Variable `$divisions` gespeichert wurde, wird nach Notwendigkeit der aktuellen Funktion zurückgegeben, wie im Listing 4.5 gesehen werden kann.

```
65 } while($currUser < $this->usersNumber);
66
67 return $divisions;
```

Listing 4.5: Die neue Heuristik: Ende der Hauptschleife

Der vorgestellte Algorithmus teilt alle Teilnehmer ein, verwendet einen priorisierten Einteilungsverfahren und ist sehr einfach verständlich, da auf den Ideen aus dem Abschnitt 4.1.3 basiert, die auch für nicht-Fachspezialisten verständlich sind.

4.2 Laufzeitanalyse

Eine der im Abschnitt 4.1.2 formulierten Anforderungen zum Algorithmus ist die hohe Geschwindigkeit der Einteilung, die dafür notwendig ist, dass die *vorläufigen Einteilungen* mehrmals auf Wunsch berechnet werden können, ohne dass dafür wochenlang gewartet werden muss.

Bei kleinen Problemgrößen ist die Ausführungsdauer gering auch bei der Einsetzung der Branch-and-Cut-Methode, wie im Abschnitt 3.1.2 beschrieben wurde. Daher wird eine schnellere Heuristik in solchen Fällen gar nicht benötigt. Bei größeren Problemgrößen, wo Hunderte Interessenten zwischen vielen Gruppen geteilt werden müssen, ist die Ausführungsdauer stark davon abhängig, welche Komplexitätsklasse der Algorithmus hat.

Betrachen wir die Worst-Case Laufzeit des vorgestellten Algorithmus auf theoretischer Sicht:

- Die Problemgröße wird mit zwei Variablen definiert:
  n die Anzahl der Teilnehmer
  m die Anzahl von Gruppen, in die sie eingeteilt werden müssen
- Für die Sortierung der Bewerbungen nach Priorität und dann nach der Losnummer z.B. mittels Heap-Sort werden $\mathcal{O}(n \cdot \log n)$ Rechenschritte benötigt.
- Um die Kennungen der existierenden Gruppen zu laden, braucht man $\mathcal{O}(m)$ Rechenoperationen.
- Die Initialisierung (Listing 4.1) benötigt $\mathcal{O}(n+m)$ Schritte, um die Referenzen auf die Variablen zu kopieren.
- Für die Berechnung der potentiellen Gruppengrößen, die im Listing 4.2 angegeben wird, müssen alle Bewerbungen durchgesucht werden. Die entsprechende Laufzeit beträgt $\mathcal{O}(n)$.
- Die Initialisierung der Gruppengrößen wird für jede Gruppe durchgeführt. Somit braucht man dafür $\mathcal{O}(m)$ Rechenschritte.
- Die Hauptschleife (Zeilen 18-65) wird höchstens n Mal durchgelaufen, da bei jedem Durchlauf mindestens ein Teilnehmer eingeteilt wird.
- Falls keine Gruppen sofort gefüllt werden können, wird die Füllschleife (Listing 4.3, Zeilen 20-42) höchstens $\mathcal{O}(m)$ Rechenoperationen benötigen, um das festzustellen.
  Im sonstigen Fall würde sie Gruppen füllen, wobei für die Einteilung jedes Teilnehmers $\mathcal{O}(n)$ Schritte benötigt werden, damit alle seine Bewerbungen aus der lokalen Bewerbungsliste gelöscht werden können. Die das gewährleistende Funktion `$this->delApplicationsOf()` wird im Listing 4.6 zur Anzeige vorgelegt. In solchem Fall wären für die Einteilung aller n Teilnehmer $\mathcal{O}(n \cdot n)$ Rechenschritte notwendig.
- Fürs Feststellen einer am besten passenden Bewerbung für die Einteilung einer besten Bewerbung (4.4, Zeilen 46-51) werden im Worst-Case alle vorhandenen Bewerbungen durchgesucht,

was die Laufzeit von $\mathcal{O}(n)$ verursachen würde.

```
1  function delApplicationsOf(&$registrations, $benutzer_id, &$potSizes){
2    $prioritiesNumber = count($registrations);
3    for($priority = 1; $priority <= $prioritiesNumber; $priority++){
4      $newSamePriorRegistrations = array();
5      foreach($registrations[$priority] as $registration)
6        if($registration['benutzer_id'] == $benutzer_id)
7          $potSizes[$registration['gruppe']]--; // no application anymore
8        else // stays in the list
9          $newSamePriorRegistrations[] = $registration;
10     $registrations[$priority] = $newSamePriorRegistrations;
11   }
12 }
```

Listing 4.6: Die neue Heuristik: Lösche alle Bewerbungen eines Benutzers

- Der diese Bewerbung gesendete Interessent wird in konstanter Zeit eingeteilt, verursacht allerdings dabei einen Aufruf von `$this->delApplicationsOf()`, was $\mathcal{O}(n)$ Rechenoperationen dauert.
- Für die Wiederherstellung der Liste der Gruppen mit der kleinsten Anzahl der Teilnehmer werden $\mathcal{O}(m)$ Schritte benötigt. Da diese allerdings im Durchschnitt nur in jeder m-ten Schleife gemacht wird, ist die Laufzeit pro Schleifendurchlauf konstant $\mathcal{O}(1)$.

Insgesamt beträgt somit die theoretisch für die Ausführung vom gesamten Algorithmus notwendige Anzahl der Rechenoperationen:

$$\mathcal{O}\left(n \cdot \log n + m + n + m + n + m + n \cdot (n+n)\right) = \mathcal{O}\left(n^2 + m\right)$$

Da davon ausgegangen werden kann, dass die Gruppenanzahl kleiner als die gesamte Teilnehmeranzahl sein wird, kann das auf $\mathcal{O}(n^2)$ reduziert werden.

Diese theoretische Einschätzung zeigt, dass selbst für die größten in der Praxis vorstellbaren Problemgrößen so wenig Rechenzeit benötigt wird, dass diese mit der vorgestellten Heuristik innerhalb von Sekunden gelöst werden können. Für ein Kurs mit 10000 Teilnehmern, die in 500 Gruppen eingeteilt werden müssen werden dabei ungefähr $10000^2 = 10^8 = 100$ Mio. Operationen benötigt. Wenn ein Rechner für die Berechnung genommen wird, der Zehn Milliarden Berechnungen pro Sekunde durchführen kann, wird die Berechnung $\frac{10^8}{10^10} = 0.01\,\text{s} = 10\,\text{ms}$ benötigt.

4.3 Anforderungen an das System und ihre Erfüllbarkeit

Zu den allgemeinen Anforderungen an den Algorithmus kommen noch einige weitere Gruppeneinteilungssystemanforderungen hinzu, die wir beim Vergleich der existierenden Lösungen in der Tabelle 3.2 betrachtet haben. Ein gutes Gruppeneinteilungssystem muss sie alle erfüllen.

Verfahrensbedingt werden allen Bewerbern *gleiche Chancen* bei der Einteilung gegeben. Der Algorithmus sorgt auch dafür, dass alle Interessenten in Gruppen eingeteilt werden, selbst wenn das für die *Gleichmäßigkeit* der Gruppengrößen eher ungünstig wäre. Die *vorläufige Einteilung* zur Probe ist aufgrund von geringer Einteilungsdauer problemlos möglich.

Die anderen Vergleichskriterien werden vom Kernalgorithmus nicht direkt erfüllt, sind allerdings bei dessen Verwendung sehr einfach einbaubar. Hier kommt eine Auflistung davon, wie die entsprechenden Algorithmen aussehen müssen. Die Implementierungsdetails sprengen den Rahmen dieser Ausarbeitung.

- Die *Wunschänderung* ist problemlos möglich: Vor der endgültigen Einteilung können die alten Wünsche aus der Liste der Bewerbungen gelöscht werden. Diese werden vollständig durch die Abänderungen ersetzt.
- Der Kursleiter kann die Gruppenkennungen bei den Zuteilungsergebnissen ausgewählter Benutzer nach der erfolgten Einteilung manuell in der entsprechenden Liste verändern.
- Die aktuelle Situation kann sehr einfach und übersichtlich auf ähnliche Art und Weise dargestellt werden, wie das im Gruppeneinteilungssystem des Instituts für Mathematik der Goethe-Universität Frankfurt praktiziert wird: Wie auf der Abbildung 4.1 gezeigt wird, kann ein Ampelartiges System für die Darstellung vom aktuellen Füllungsstatus von Gruppen verwendet werden.

  Dabei werden *Schwellengrößen* in Prozentsätzen zur durchschnittlichen Gruppengröße ausgewählt. Liegt die Anzahl der Bewerbungen für diese Gruppe unter dem kleinsten Schwellenwert, wird der Gruppenstatus als *unpopulär* bezeichnet und mit einem grünen Kästchen markiert. Bei den Gruppen, deren Größen über dem Schwellenwert liegen, ist der Status *überpopulär*. Sie werden mit einem roten Kästchen markiert. Die restlichen Gruppen bekommen den Status *normal* und ein gelbes Kästchen.

  Die Bewerbungen mit unterschiedlichen Prioritäten werden separat voneinander betrachtet, damit besser eingeschätzt werden kann, welche Termine tatsächlich bevorzugt und welche nur als Ersatz genommen werden.

  Die Popularität jeder Gruppe kann somit durch drei Kästchen beschrieben werden: Das erste (z.B. von links) entspricht der ersten Priorität, dann kommt die zweite und anschließend die dritte Priorität.

Abbildung 4.1: Gruppengrößenampeln

Betrachten wir den Fall, wo die Durchschnittsgröße der Gruppen 20 beträgt und für eine Gruppe bekannt ist, dass sie von 12 Teilnehmern als erster Wunsch ausgewählt wurde, als zweiter Wunsch von 21 und als dritter Wunsch von 30 Bewerbern. Die Schwellengrößen liegen bei 70% und 130%, also bei $70\% \cdot 20 = 14$ und $130\% \cdot 20 = 26$ Teilnehmern:

- $12 < 14 \Rightarrow$ das erste Kästchen ist grün, da die Gruppe gemäß den ersten Wünschen *unpopulär* ist.
- $14 \leq 21 \leq 26 \Rightarrow$ das zweite Kästchen ist gelb, da die Gruppe entsprechend den zweiten Wünschen *normal* ist.
- $30 < 26 \Rightarrow$ das dritte Kästchen ist rot, da laut den dritten Wünschen die Gruppe *überpopulär* ist.

Die aus solcher Belegung entstehende Ampel kann man auf der Abbildung 4.1 sehen.

- Die Beachtung paralleler Veranstaltungen, damit die unbeabsichtigte Einteilung in mehrere zur gleichen Zeit angebotene Gruppen verhindert wird ist einfach einbaubar, indem nach der endgültigen Einteilung eines Kurses das System alle Bewerbungen des gleichen Benutzers (ersichtlich anhand seiner Benutzerkennung) für die in der gleichen Zeit angebotenen Gruppen aller anderen vom System verwalteten Kurse gelöscht werden. Falls Wünsche der ersten oder zweiten Priorität auf solche Art und Weise gelöscht werden, werden die verbleibenden Wünsche prioritätsmäßig nach oben verschoben. Das bedeutet, falls ein Wunsch der Priorität 1 gelöscht wird, wird der restliche zweite Wunsch zum Ersten und der dritte zum Zweiten. Aufgrund der Tatsache, dass diese Änderungen vom Benutzer so schnell wie möglich erkannt werden müssen, damit er darauf reagieren kann, wird anschließend eine E-Mail Benachrichtigung sowie über die erfolgte Einteilung als auch über die aufgrund dessen durchgeführte Änderungen seiner Wünsche an ihn gesendet.
- Die Beachtung paralleler Gruppen, damit Termine mit gleichen Bedingungen auch für das System als gleich angesehen werden, kann bei der Verwendung der neuen Heuristik komplett problemlos in den Algorithmus integriert werden, indem vor der eigentlichen Einteilung die lokale Kopie aller Bewerbungen durchsucht wird und zu jeder Bewerbung, die sich auf eine der Gruppen bezieht, zu denen parallel andere Gruppen angeboten werden, werden Duplikate der Bewerbung hinzugefügt, wo anstatt der originalen Gruppe die alternative Gruppe steht. Das ist in linearer Zeit abhängig von der Bewerbungsanzahl machbar, wie im nächsten Kapitel gezeigt

wird.

- Die Überschneidungsdarstellung ist dadurch möglich, dass der gleiche Algorithmus, der zur Erkennung zur gleichen Zeit belegter Gruppen paralleler Veranstaltungen dient, eingesetzt wird, um beim Einloggen ins Gruppeneinteilungssystem die möglichen Überschneidungen bereits vor der Einteilung zu zeigen, damit diese für den Benutzer offensichtlich werden.
- Für die Referenzfunktion muss der Einteilungsalgorithmus zum Teil modifiziert werden:

  Die Bewerbungen, bei denen anstatt der Wunschgruppen Referenzen auf die anderen Benutzer gegeben werden, müssen in einer Hashtabelle mit den Referenzen als Schlüssel erstmal erfasst werden. Nachdem eine Person in eine Gruppe eingeteilt wird, mit der gemäß dieser Hashtabelle weitere Teilnehmer in der gleichen Gruppe sein möchten, werden sie sofort in die gleiche Gruppe eingeteilt.

  Aus diesem Grund wird die Größe dieser Gruppe nicht um ein, sondern gleich um einige Teilnehmer steigen.

  Die eingeführte Heuristik arbeitet mit zwei Listen von Gruppen: Mit der Liste aller Gruppen, in die eingeteilt werden muss und einer Liste, wo die momentan kleinsten Gruppen stehen. Dabei beträgt der Größenunterschied zwischen den kleinsten Gruppen höchstens eins und den restlichen verfahrenstechnisch genau eins.

  Daher ist es möglich, falls die Liste der Gruppen mit der geringsten Teilnehmerzahl leer ist, diese mit allen anderen Gruppen zu füllen, um wieder die Liste der Gruppen mit der kleinsten Teilnehmerzahl zu bekommen. Das sieht man im Listing 4.4 in den Zeilen 61-62.

  Da bei der Verwendung der Referenzfunktion die Größenunterschiede unterschiedlich sind, macht es Sinn:

  1. Eine zusätzliche Variable einzuführen, wo die geringste Gruppengröße gespeichert wird.
  2. Falls die Liste der kleinsten Gruppen leer ist, die Variable mit der geringsten Gruppengröße zu inkrementieren und dann nur diejenigen Gruppen aus der Menge aller zu bearbeitenden Gruppen in die Menge der kleinsten Gruppen einzutragen, deren Größe genauso viel beträgt.
  3. Den Schritt 2 wiederholen, solange die Liste der kleinsten Gruppen leer ist (gut für die Situation, wenn alle Gruppen früher mehrere Teilnehmer erhalten haben).

4.4 Schnittstellen

Für die Verwaltung von Kursen werden diverse Lernplattformen eingesetzt, die zahlreiche mit dem Lernprozess verbundene Funktionalitäten in digitaler Form anbieten. Einige solcher Lernplattformen bieten auch die Möglichkeit der Gruppeneinteilung an, wie das im Abschnitt 3.3 beschrieben wurde.

Aufgrund der Tatsache, dass diese Lernplattformen oft mangelhafte Funktionalität im Gruppeneinteilungsbereich besitzen, da bei ihrer Entwicklung der Fokus nicht auf diesen Aspekt gelegt wird, müssen dedizierte Gruppeneinteilungssysteme parallel damit eingesetzt werden.

Falls mehrere Systeme parallel mit den gleichen Daten operieren, entstehen zwei grundsätzliche Probleme:

- Der mehrfach vergrößerte *Verwaltungsaufwand* für das simultane Eintragen bzw. Verändern von gleichen Datensätzen in alle Systeme.
- *Inkonsistenz* einzelner Datensätze, die aufgrund der menschlichen Fehler bei der mehrmaligen Eingabe von gleichen Daten bzw. aufgrund fehlender für die Verifizierung aller Angaben in allen Systemen Zeit entsteht.

Diese Probleme können dadurch vermieden werden, dass alle Systeme für den Umtausch der Informationen bestimmte *Schnittstellen* verwenden, die einen der zwei Arten vom Informationsaustausch zwischen den Systemen ermöglichen:

1. Beim *automatisierten Informationsaustausch*, können die Systeme ohne manuellen Einsatz untereinander kommunizieren, damit Daten synchronisiert werden.

   Bei dieser Art vom Informationsaustausch ist der *Verwaltungsaufwand* durch die parallele Verwendung von mehreren Systemen bis auf das Geringste minimiert und *Inkonsistenz* kaum möglich, da die durch Menschen verursachten Fehler nicht vorkommen können.

2. Beim *halbautomatischen Informationsaustausch*, werden Daten vom Kursverwalter von einem System auf ein Speichermedium exportiert, eventuell angepasst, anschließend für das zweite System zugänglich gemacht und dann über seine Schnittstellen in das System importiert.

   Hierbei ist der Verwaltungsaufwand nur in dem Fall geringer als das manuelle Eintragen, wenn viele Werte eingetragen werden müssen. Außerdem ist die Wahrscheinlichkeit entstehender Inkonsistenz in dem Fall geringer als beim manuellen Eintragen der Werte, wenn die exportierten Daten vor dem Import in das andere System nicht manuell verarbeitet werden müssen.

Für den im Rahmen meiner Bachelorarbeit erstellten Prototyp des Gruppeneinteilungssystems, der in der Goethe-Universität Frankfurt verwendet werden soll, war es aus diesen Gründen sinnvoll, die existierenden Schnittstellen zu analysieren, um verstehen zu können, ob man den automatisierten oder zumindest den halbautomatischen Informationsaustausch mit den an der Universität verwendeten

Lernsystemen organisieren kann.

4.4.1 QIS/LSF

Das Hochschulverwaltungssystem QIS/LSF wird als ein zentrales Kursverwaltungssystem an der Goethe-Universität Frankfurt am Main verwendet. Daher sollten vom Prototyp des Gruppeneinteilungssystems die Möglichkeiten der Synchronisierung angeboten werden. QIS/LSF bietet zwei unterschiedliche Möglichkeiten zum Export von Daten: Eine SOAP[1]-Schnittstelle sowie auch den Export aus dem Webinterface in eine XML-Datei, die dann von einem anderen System geparsed werden kann.

Auf meine E-Mail-Anfragen an die QIS/LSF-Administration bezüglich der Informationen zu der SOAP-Schnittstelle habe ich bisher leider immer noch keine Informationen erhalten. Mir wurde mitgeteilt, dass diese existiert aber nicht wie sie verwendet und was genau dadurch exportiert werden kann. Am 28.08.12 wurde mir gesagt, dass ich die Informationen dazu bald bekomme. Seit diesem Zeitpunkt bekomme ich allerdings keine Antworten von der Administration. Damit war für mich die Verwendung dieser theoretisch existierenden Schnittstelle in der Praxis leider nicht möglich. Ich hoffe, dass andere Entwickler dazu mehr Informationen bekommen.

Über die aus dem Kursleiterinterface herunterladbare XML-Export-Dateien können die Details der über QIS/LSF durchgeführten Gruppeneinteilung ausgelesen werden. Ein Beispiel einer solchen Datei ist im Listing 4.7 zu sehen. Die daraus auslesbaren Informationen sind allerdings für ein Gruppeneinteilungssystem leider nutzlos. Die QIS/LSF-Administration hat mir mitgeteilt, dass die allgemeinen Kurseinstellungen ebenfalls exportiert werden können aber keine weiteren Informationen dazu gegeben. Da gemäß den Behauptungen der Kursleiter, um eine solche Datei zu bekommen, „langes Klicken“ im Webinterface nach dem Einloggen mit einem Kursleiteraccount benötigt wird, ist ein solcher Export im Vergleich zum manuellen Eintragen von Kursangaben deutlich schwieriger und kann somit nicht empfohlen werden.

```
1 <Cell> <Data ss:Type="String">1234567</Data> </Cell>
2 <Cell> <Data ss:Type="String">Max</Data> </Cell>
3 <Cell> <Data ss:Type="String">Mustermann</Data> </Cell>
4 <Cell> <Data ss:Type="String">M</Data> </Cell>
5 <Cell> <Data ss:Type="String">max@mustermann.de</Data> </Cell>
```

Listing 4.7: Beispiel einer Export-XML-Datei von QIS/LSF

Eine alternative Möglichkeit, die Daten automatisch zu bekommen, bestünde in Verbindung mit dem Parsen der öffentlich erreichbaren Kursseite in QIS/LSF. Das wäre allerdings auch keine gute Lösung, da bei jeder Webinterface-Änderung von QIS/LSF die Gefahr der fehlerhaften Erkennung von

[1] **S**imple **O**bject **A**ccess **P**rotocol - Ein Netzwerkprotokoll zur Übertragung von Daten zwischen Systemen

Benutzer in Gruppe hinzufügen

```
PUT http://demo.olat.com/olat/restapi/groups/{groupId}/users/{userId}
Content-Type: application/json
Response: 200
```

Abbildung 4.2: Zuordnung Benutzer zu Gruppen mit OLAT REST API[7]

Kursdaten bestehen würde und das Parser-Modul daher jedes mal auf das aktuelle Interface angepasst werden müsste.

Der Import von Daten wird von QIS/LSF grundsätzlich nicht unterstützt.

Zusammenfassend lässt sich sagen, dass QIS/LSF keine verwendbaren Schnittstellen für ein parallel verwendetes Gruppeneinteilungssystem anbietet.

4.4.2 OLAT

Die Lernplattform OLAT bietet seit der 7. Version die sogenannte REST API[2]-Schnittstelle dafür an, um über HTTP-Anfragen Gruppen zu erstellen und Benutzer dorthin zu platzieren.

Ein Beispiel dazu, wie das erfolgen könnte, findet man auf der Abbildung 4.2.

Für die Zuordnung der OLAT-Teilnehmer zu den OLAT-Gruppen muss das Gruppeneinteilungssystem allerdings die Kennungen aller einzuteilenden Benutzer in OLAT kennen, was jeder Benutzer dann mittels eines REST API-Skriptes erfahren könnte.

4.4.3 Moodle

Die Lernplattform Moodle bietet seit der Version 2.0 analog zu OLAT REST API-Schnittstellen an. Diese heißen allerdings *Web service protocols*. Dadurch sind ebenfalls sowohl die Gruppenerstellung (Funktion `moodle_course_create_courses()`[8]), als auch die Zuordnung der Benutzer in Gruppen (Funktion `moodle_group_add_groupmembers()`[8]) möglich.

Dafür müssen allerdings die Moodle-Benutzerkennungen dem Einteilungsskript genauso wie bei OLAT bekannt sein. Diese können Benutzer von Moodle allerdings nach dem Einloggen in der Adresszeile ablesen und dann dem Gruppeneinteilungssystem mitteilen.

4.4.4 Central Authentication Service (CAS)

Falls bestimmte Studierende mehr Chancen als die anderen bei der Einteilung in die gewünschten Gruppen haben möchten, können sie auf die Idee kommen, mehrere Accounts im Gruppeneintei-

[2] **Re**presentational **S**tate **T**ransfer **A**pplication **P**rogramming **I**nterface - Ein Netzwerkprotokoll für die Übertragung von Daten zwischen Systemen

lungssystem zu erstellen und sich damit mehrmals zu bewerben. Das verletzt die Chancen der anderen Benutzer und führt zur Füllung von echten Gruppen mit nicht-existierenden Teilnehmern, was die Gleichmäßigkeit der Verteilung stark negativ beeinflusst.

Damit jeder Studierende sich nur einmal mit einem Account bewerben kann, muss seine Identität mit diesem Account verbunden werden. Da jeder Studierende der Goethe-Universität Frankfurt genau ein HRZ[3]-Login hat, kann man für die Bewerbung für die Kurse für Studierende nur diejenigen Benutzer des Gruppeneinteilungssystems zulassen, die ihre dortigen Accounts mit den HRZ-Accounts verbinden.

Eine solche Verbindung ist möglich und kann von einem anderen System beispielsweise mit Hilfe des *phpCAS*-Skripts[4] durchgeführt werden, das sich mit dem CAS-Server in Verbindung setzt und die Accountdaten überprüft. Der CAS-Server ist der zentrale Authentifizierungsservice für die Angehörigen der Goethe-Universität Frankfurt.

Die Verwendung von *phpCAS* ist einfach, wie im Listing 4.8 gezeigt wird: Unter Verwendung von dem dem CAS-Server bekannten Zertifikat kann ein Computer, der sich innerhalb des universitären Netzwerks befindet, den CAS-Server ansprechen und ihn darum bitten, den aktuellen Benutzer zu authentifizieren. Im Falle eines Erfolges, liefert phpCAS das Login des Benutzers im CAS-System. Sonst wird die Ausführung des Skriptes angehalten, damit keine Aktivitäten seitens der nicht-authentifizierten Benutzer möglich ist.

```
1  /* phpCAS-Client */
2  require_once('CAS.php');
3  /* Initialisierung */
4  phpCAS::client(CAS_VERSION_2_0,'ssl.sd.uni-frankfurt.de',443,'/cas',false);
5  /* Server ueberpruefen */
6  phpCAS::setCasServerCACert('/etc/ssl/certs/gdv-ca-certificates.crt');
7  /* CAS-Authentifizierung; ruft CAS auf und kehrt bei erfolgreicher
       Authentifizierung hierher zurueck */
8  phpCAS::forceAuthentication();
9  /* Ab dieser Stelle war die CAS-Authentifizierung erfolgreich, und der
10 * Benutzername kann mit phpCAS::getUser() ausgelesen werden */
11 $hrzlogin = strtolower(phpCAS::getUser());
```

Listing 4.8: Verwendung von phpCAS

[3] **H**ochschul**r**echen**z**entrum
[4] CAS-Client auf PHP-Basis

4.4.5 Die Schnittstellen des Gruppeneinteilungssystems

Unter Verwendung von solchen Open Source Softwarebibliotheken, wie beispielsweise PHPExcel[5] besteht die Möglichkeit für das Gruppeneinteilungssystem:

- Die Anmeldedaten der Benutzer in XLS- und OpenXML-Formate zu exportieren, damit diese an andere Systeme weitergegeben werden können.
- Die Anmeldedaten im gleichen Format aus anderen Systemen zu importieren.
- Die Anmeldedaten aus einer lokalen Kopie zu importieren, um im Falle eines technischen Problems die Daten vom Server wiederherstellen zu können.
- Die allgemeinen Kursangaben zu importieren, um die halbautomatische Synchronisation mit den Kursverwaltungssystemen zu ermöglichen.

[5] http://phpexcel.codeplex.com/

Kapitel 5

Implementierung

Dieses Kapitel erläutert, wie der im Rahmen dieser Bachelorarbeit erstellte Prototyp eines guten Gruppeneinteilungssystems implementiert wurde. Hier wird beschrieben, welche Ergebnisse der implementierte Algorithmus der vorgeschlagenen speziellen Heuristik beim Testen mit den anonymisierten Anmeldedaten aus den vergangenen Jahren erzielt. Anhand dessen wird beurteilt, ob diese Heuristik praxistauglich ist. Außerdem wird dabei die Laufzeit gemessen, damit die zeitlichen Kosten der vorläufigen Verteilungen ersichtlich sind.

5.1 Sprach- und Plattformauswahl

Damit die Auswahl der Wunschtermine von allen Kursteilnehmern durchgeführt werden kann, unabhängig von ihren persönlichen zeitlichen und räumlichen Präferenzen, muss diese per Internet erfolgen. Daher ist es sinnvoll, ein Web-basiertes Gruppeneinteilungssystem zu implementieren, dessen Funktionalitäten über Internet erreichbar sind.

Von allen Web-Programmiersprachen sollte die populärste genommen werden, damit möglichst viele Entwickler den Programmcode des Systems verstehen können und die Weiterentwicklung des Gruppeneinteilungssystems auch in Abwesenheit der Erstentwickler möglich ist. Außerdem werden damit die Codebeispiele in dieser Sprache damit für die meisten Leser dieser Arbeit anschaulicher. Zum heutigen Zeitpunkt ist somit PHP die Sprache der Wahl, da sie von ca. 78% aller Webseiten verwendet wird[1].

Der Nachteil der Verwendung von PHP ist seine langsamere Ausführungszeit als beispielsweise beim C-Code. Falls aus irgendwelchen Gründen eine schnellere Ausführung benötigt wird, können für die Beschleunigung vom Programmcode die sogenannten PHP-Beschleuniger verwendet werden, die die Ausführung durch die Vorkompilierung um den Faktor 2-7 beschleunigen (beispielsweise *Zend Op-*

[1] Gemäß den Angaben von `http://w3techs.com`. Die Übersicht ist auf der Abbildung 5.1 findbar.

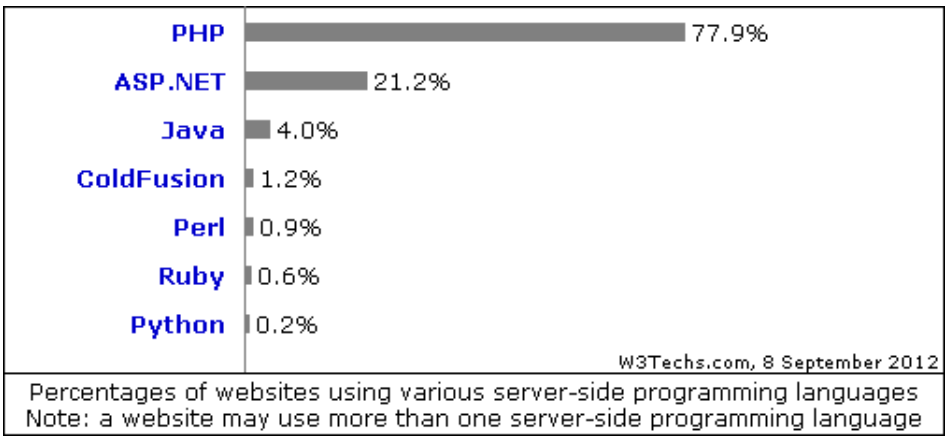


Abbildung 5.1: Verwendung der Server-Side Programmiersprachen für die Webservices

timizer+[2]). Für eine noch stärkere Beschleunigung wäre die Umschreibung der Kernmodule in C ratsam.

Für die Speicherung von Teilnehmerdaten wird aufgrund der zu erwartenden großen Mengen beim aktiven Einsatz und zwecks Verbesserung der Zugriffszeiten bei mehreren parallelen Zugriffen eine Datenbank verwendet. Als Datenbankverwaltungssystem wird MySQL genommen, da dieses kostenlos bei der Verwendung, sowie auch weit verbreitet ist und den Geschwindigkeit- und Sicherheitsanforderungen völlig entspricht.

Die Verwendung populärer und kostenloser Plattformen ermöglicht außerdem gute Portabilität des damit implementierten Systems. Das kann beim Serverwechsel aufgrund Software bzw. Hardwareupgrades oder aufgrund eines Hardwareausfalls sehr hilfreich sein. Weiterhin ist damit das Klonen des implementierten Gruppeneinteilungssystems für die Verwendung von anderen Institutionen ohne großem Aufwand möglich.

Für die statistischen Graphen im Gruppeneinteilungssystem wird das Open Source Framework *pChart*[3] eingesetzt. Dieses verwendet PHP-Funktionen für die Generierung von Graphen direkt auf dem Web-Server. Die in dieser Arbeit bei der Analyse von Gruppeneinteilung verwendeten Graphen wurden mit Hilfe von diesem Framework generiert.

[2] Weitere Informationen dazu können hier gefunden werden: http://www.zend.com/products/server/
[3] Herunterladbar unter http://www.pchart.net/

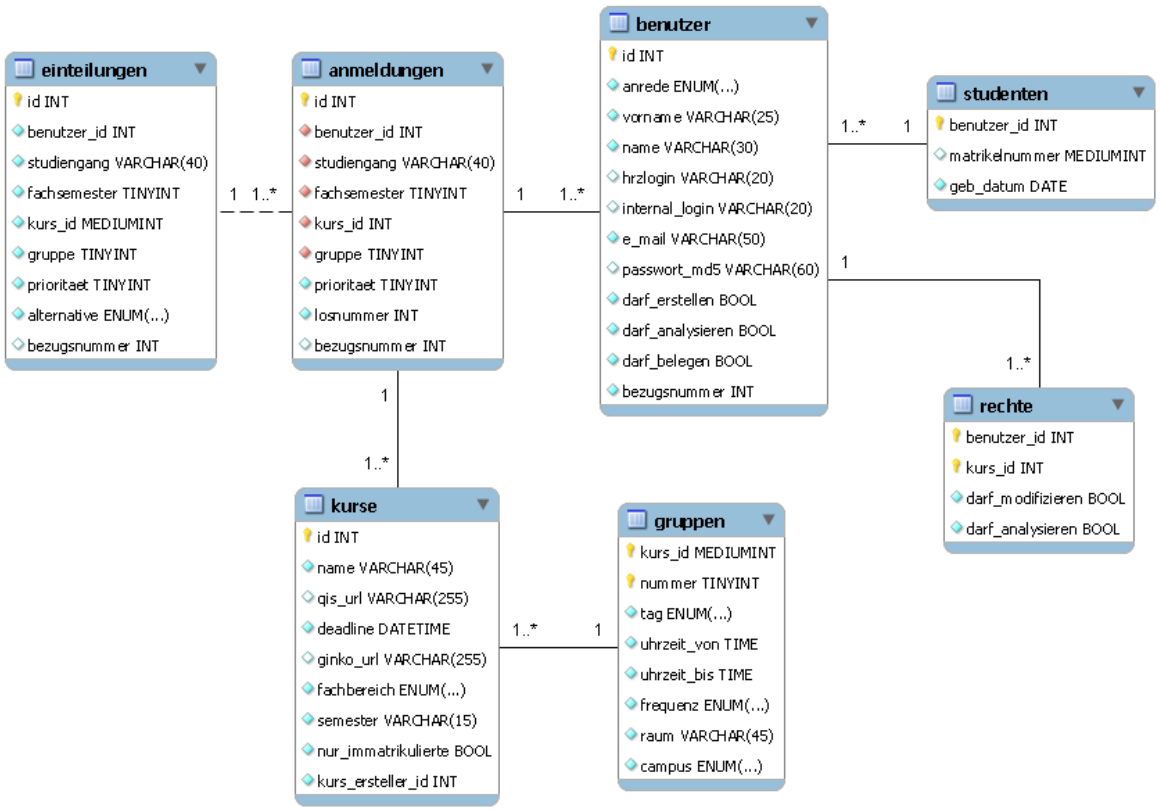


Abbildung 5.2: Strukturelle Darstellung der vom Gruppeneinteilungssystem verwendeten Datenbank

5.2 Datenbankstruktur

Die Datenbank besteht aus sieben Tabellen, die in der Abbildung 5.2 dargestellt sind: `anmeldungen`, `benutzer`, `studenten`, `einteilungen`, `kurse`, `gruppen` und `rechte`.

Die Tabelle `anmeldungen` speichert die **Bewerbungen** einzelner Teilnehmer.

- Jede Bewerbung kann anhand einer eindeutigen Kennungsnummer `id` identifiziert werden. Diese Nummer vergrößert sich automatisch mit jeder neuen Bewerbung.
- `benutzer_id` speichert die Kennung des Benutzers, der die aktuelle Bewerbung erstellt hat. Der Benutzer kann dann unter dieser Kennung in der Tabelle `benutzer` gefunden werden (das entsprechende Feld heißt dort allerdings `id`) .
- Im Feld `studiengang` wird der Studiengang gespeichert, in dem sich beworben wird. Da Studierende mehrere Studiengänge haben können, müssen sie bei der Anmeldung zu den Kursübungen einen bestimmten auswählen.

  `fachsemester` entspricht der Angabe vom Fachsemester im ausgewählten Studiengang.
- `kurs_id` ist die eindeutige Kennung des Kurses, für dessen Gruppen sich beworben wird.

- `gruppe` bezeichnet die Nummer der ausgewählten Gruppe, für die sich mit der Priorität `prioritaet` beworben wird.

  Die Gruppennummern sind im Rahmen eines Kurses einmalig. Mehrere Kurse dürfen gleiche Gruppennummern haben, damit diese im Laufe der Zeit bei der Verwendung des Systems nicht zu groß werden.

  Für einen Kurs kann jeder Benutzer maximal drei Einträge in der Tabelle `anmeldungen` haben. Alle diese Einträge beziehen sich auf unterschiedliche Gruppen und unterschiedliche Prioritäten aus der Liste $\{1,2,3\}$. Dabei wird die größte Priorität mit einer Eins bezeichnet und die geringste mit einer Drei.

- Bei der ersten Bewerbung für einen Kurs bekommt der Benutzer eine zufällig generierte Losnummer zugewiesen. Im Rahmen des Kurses ist sie eindeutig. Anhang dieser Nummer wird entschieden, welcher Benutzer bei gleichen Bedingungen bevorzugt wird.

- Falls bei der Anmeldung eine Referenz auf einen anderen Benutzer angegeben wurde, wird sie im Feld `bezugsnummer` gespeichert.

Die Tabelle `einteilungen` enthält die **ausgewählten Bewerbungen** aus der Tabelle `anmeldungen` und hat zum großen Teil die gleichen Felder mit der gleichen Bedeutung (die `id` muss hier allerdings der `id` in der Tabelle `anmeldungen` nicht entsprechen). Zwei Unterschiede sind:

- Die Losnummer wird nicht mehr gespeichert, da diese nach der Einteilung nicht mehr notwendig ist.

- Das Feld `alternative` bekommt als Wert `1wunsch`, falls es sich um das Ergebnis der Einteilung in der ersten Güteklasse mit nur einem Wunsch handelt und entsprechend `2wuensche` bzw. `3wuensche` in den beiden restlichen Fällen.

  Nachdem eine der Einteilungsmöglichkeiten vom Kursleiter ausgewählt wird, tauchen in dieser Tabelle die Einträge mit `alternative=endlich` auf.

Für die Einteilungen wird eine zusätzliche Tabelle erstellt, damit diese unabhängig von den Anmeldungen gehalten werden und von den Kursleitern im Falle der Notwendigkeit manuell verändert werden können.

In der Tabelle `benutzer` sind die Daten der **Benutzer** des Gruppeneinteilungssystems zu finden:

- `id` speichert eine eindeutige Kennung eines Benutzers.

- Die Felder `anrede`, `vorname`, `name` und `e_mail` enthalten die entsprechenden Angaben über die Benutzer.

- Falls ein Benutzer über CAS-Server authentifiziert wird, wird im Feld `hrzlogin` seine dortige Kennung findbar sein. Sonst wird ein lokaler Account erstellt, zu dem eine Benutzerkennung `internal_login` sowie auch ein Passwort, dessen MD5-Hash in `passwort_md5` gespeichert wird, gehören.
- `darf_erstellen`, `darf_analysieren` und `darf_belegen` sind boolesche Werte, die die globalen Rechte des aktuellen Benutzers beschreiben. Falls der Benutzer ein Kursleiter ist und daher Kurse erstellen darf, wird beispielsweise im Feld `darf_erstellen` eine 1 bzw. logisches *wahr* gespeichert.
- `bezugsnummer` speichert die im System eindeutige Zahl, die zufällig generiert wird und als Referenz auf den entsprechenden Benutzer gilt.

Falls ein Benutzer ein **Student** ist, wird in der Tabelle **`studenten`** ein Eintrag erstellt, bei dem `benutzer_id` gleich der `id` des Benutzers in der Tabelle `benutzer` ist. Seine Matrikelnummer und Geburtsdatum werden in entsprechenden Feldern der Tabelle `studenten` gespeichert.

Nachdem ein Kurs erstellt wurde, darf grundsätzlich nur sein Ersteller die Einstellungen des Kurses verändern. Falls er seiner Sekretärin die Rechte für die Analyse der Anmeldedaten oder seinem Kollegen die **Rechte** für die Verwaltung geben möchte, wird das in der Tabelle **`rechte`** vermerkt. Jeder Eintrag dort beschreibt, welcher Benutzer (`benutzer_id`) zu welchem Kurs (`kurs_id`) welche besondere Rechte hat:

1. `darf_modifizieren` wird auf eine logische eins gesetzt, falls der entsprechende Benutzer die Einstellungen des beschriebenen Kurses bearbeiten darf.
2. `darf_analysieren` ist *wahr*, wenn der entsprechende Benutzer nur die Leserechte zum Kurs-Backend hat.

Die im System vorhandenen **Kurse** werden in der gleichnamigen Tabelle gespeichert. Jeder Kurs wird dabei durch die folgenden Eigenschaften beschrieben:

1. Eine eindeutige Kennung `id`.
2. Zwei Platzhalter für die Internet-Adressen, wo dieser Kurs in den Hochschulverwaltungssystemen gefunden werden kann (`qis_url` und `ginko_url`).
3. Der Stichtag, ab den weitere Anmeldungen für diesen Kurs nicht mehr möglich sind.
4. Der Kursname `name`.
5. Die Zugehörigkeit zu einem bestimmten Fachbereich `fachbereich`.
6. Das Semester, in dem diese Veranstaltung stattfindet.

7. Eine boolesche Variable, die angibt, ob die Teilnahme an diesem Kurs nur auf die immatrikulierten Teilnehmer begrenzt ist, d.h. nur auf die Teilnehmer, die mit sich einem HRZ-Account eingeloggt haben.

8. Die Kennung des Kurserstellers kurs_ersteller_id für die Bestimmung davon, wer für diesen Kurs verantwortlich ist und daher auch die meisten Rechte hat.

Die **Kursgruppendaten** werden in der separaten Tabelle `gruppen` gespeichert. Dazu gehören

1. Die Kurskennung `kurs_id`.

2. Die Nummer der Gruppe `nummer`, die innerhalb eines Kurses einzigartig ist.

3. Die Termindaten: `tag`, `uhrzeit_von` und `uhrzeit_bis`, sowie auch der Platz: `raum` und `campus`. Die Angabe vom Campus ist für das Gruppeneinteilungsskript notwendig, um die Entscheidung treffen zu können, ob zwei zur gleichen Zeit angebotenen Gruppen auch in der gleichen Zeit erreicht werden können und somit äquivalent sind.

4. Der Parameter `frequenz` gibt an, ob die Termine dieser Gruppe wöchentlich, nur in geraden Wochen, nur in ungeraden Wochen oder einzeln angeboten werden. Das ist auch für die Erkennung äquivalenter Termine notwendig und ist außerdem eine wichtige Eigenschaft eines Termins, die bei der Gruppenauswahl auf jeden Fall angegeben werden muss.

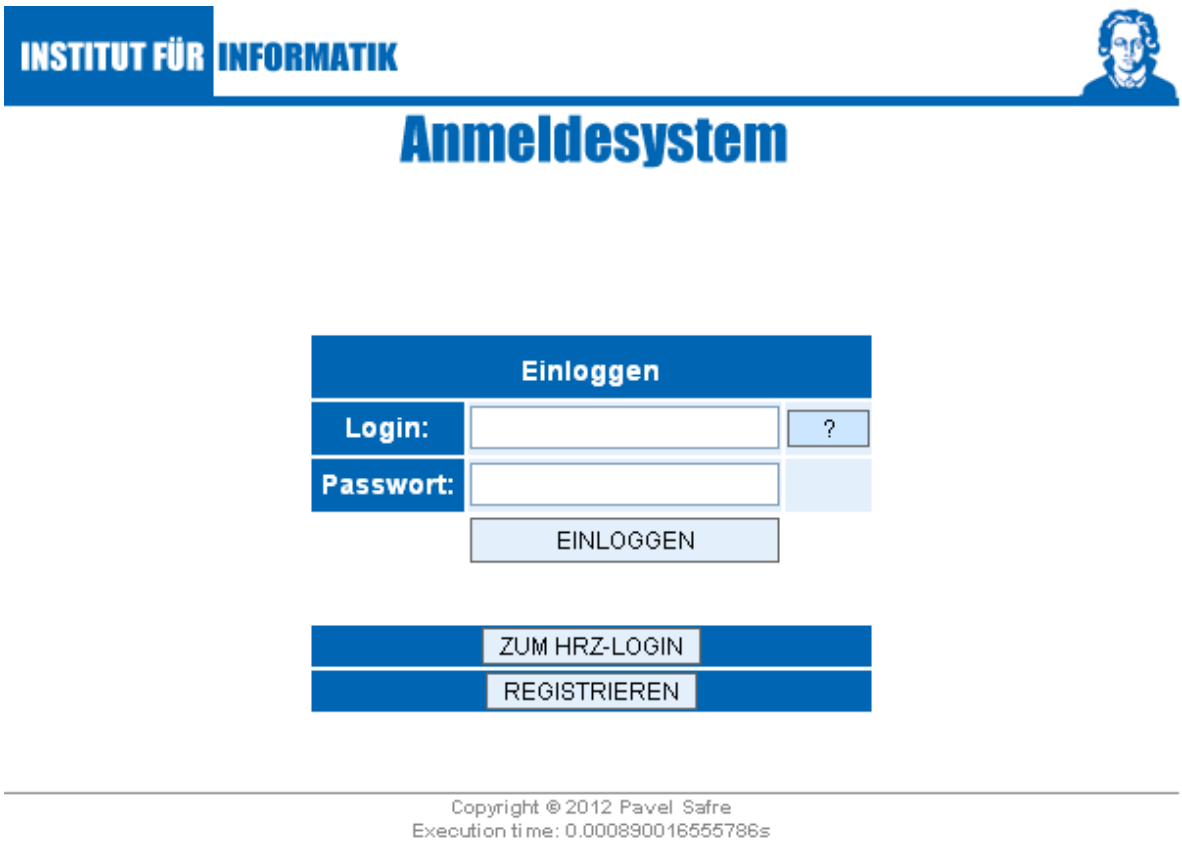


Abbildung 5.3: Anmeldebildschirm des Prototyps

5.3 Benutzerinterface

Damit das neue Gruppeneinteilungssystem effizient verwendet werden kann, muss es ein übersichtliches Benutzerinterface haben, wo alle Funktionalitäten gut findbar sind. Es soll außerdem beachtet werden, das einige Benutzer riesige 30-Zoll Große Bildschirme beim Zugriff darauf verwenden und einige andere das von einem Smartphone erreichen wollen, das eine deutlich geringere Auflösung anbietet.

Unter Beachtung dieser Aspekte wurde ein Benutzerinterface entwickelt, dass unter Verwendung aller populären Browser ähnlich aussieht. Es wurden keine Mängel festgestellt, die zur Einschränkung der Verwendbarkeit führen könnten bzw. das ästhetische Empfinden verschlechtern.

Auf der Abbildung 5.3 wird gezeigt, wie dabei der Anmeldebildschirm aussieht, wo man bestätigt, dass man Rechte zur Verwendung des Systems hat. Sollte der Benutzer dabei einen lokalen Account haben, kann er seine Daten direkt eingeben. Sollte es ein Student sein, kann er mit einem Klick auf die Taste ZUM HRZ LOGIN zur entsprechenden Authentifizierungsart wechseln.

Sollte der Benutzer zum ersten Mal hier erscheinen, kann dieser mittels eines Klicks auf die Taste REGISTRIEREN zum Registrierungsformular wechseln.

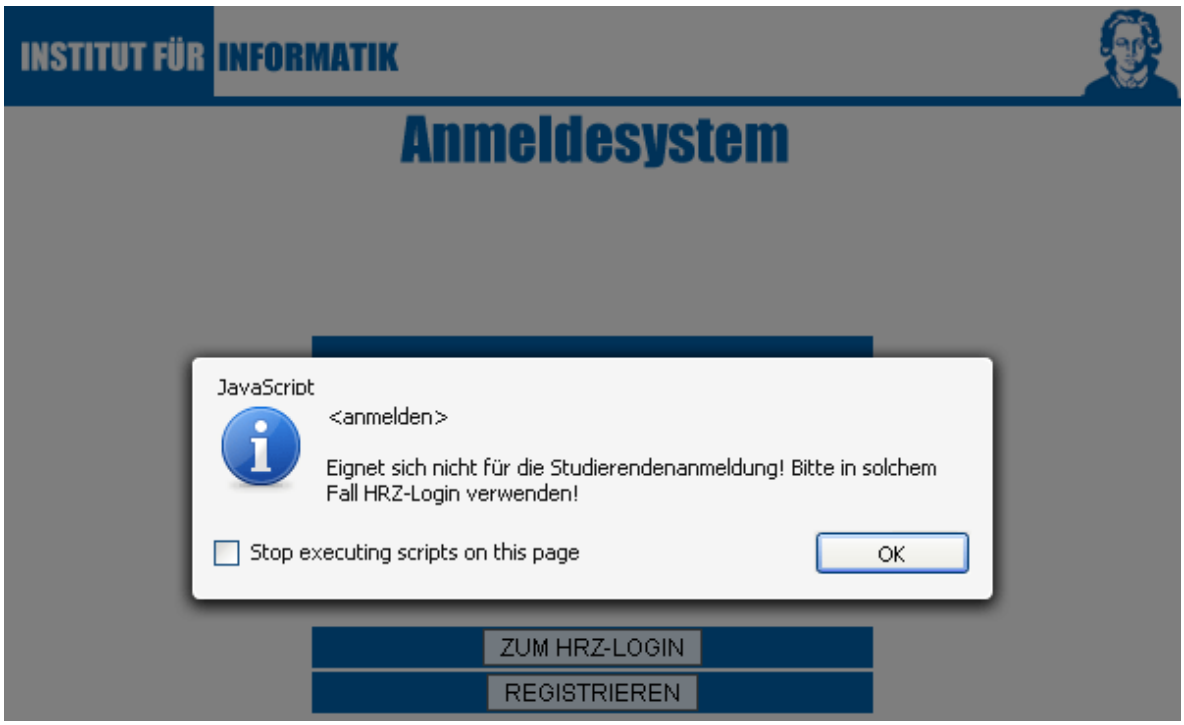


Abbildung 5.4: Informationsfenster des Prototyps

Falls im Prozess der Verwendung des Anmeldesystems beim Benutzer Fragen auftauchen, kann er in den meisten Fällen die Taste mit dem Fragezeichen betätigen, um zu erfahren, wofür welches Interfaceelement verwendet werden kann. In diesem Fall wird die begleitende Hilfsinformation in einem Informationsfenster ausgegeben, das mittels einer Browser-Funktion dargestellt wird. Das sieht man auf der Abbildung 5.4. Damit wird gesichert, dass selbst diejenigen Browser, die nur ein laufendes Fenster unterstützen, diese Informationen darstellen können.

Selbst in den Fällen, wo viele Angaben gemacht werden müssen, wird der Bildschirm so sauber wie möglich gehalten, damit der Benutzer mit den Informationen nicht überfordert wird. Falls beispielsweise ein Benutzer ein Kursleiter ist, wird er nur drei Punkte im Menü sehen: *Persönliche Angaben*, *Verwaltung* und *Logout*. Auf der Seite, wo er seine persönlichen Angaben hinterlassen kann, werden außer den Menüpunkten nur das aktuell relevante Eingabeformular ausgegeben, wie man auf der Abbildung 5.5 sieht. Bei der Verwaltung der Kurse werden nur diejenigen ausgegeben, die dieser Leiter verwalten bzw. analysieren darf.

Sollte dieser Kursadministrator zu der Idee kommen, die Details eines seiner Kurse zu bearbeiten, findet er die kompakte Auflistung aller Details dazu auf einem Bildschirm, wie auf der Abbildung 5.6 gezeigt wird. Mit einem Klick kann dort eine Gruppe hinzugefügt und beliebige Veranstaltungsdaten geändert werden.

Ein Student sieht das System ganz anders: Anstatt des Menüpunkts *Verwaltung* hat er einen anderen Menüpunkt *Anmeldung*. Beim Klicken darauf bekommt er die Liste aller Veranstaltungen, für die er

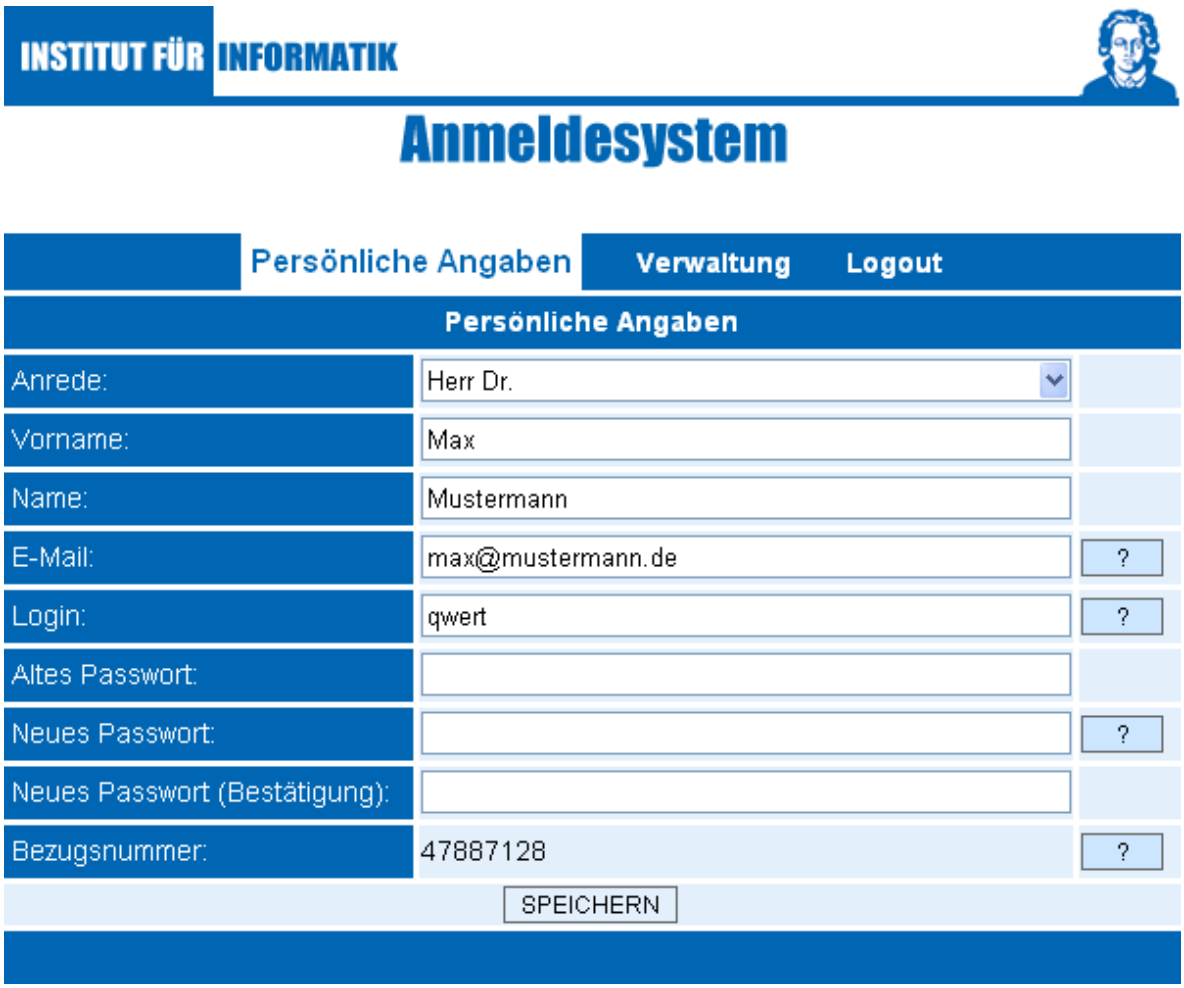


Abbildung 5.5: Persönliche Angaben eines Kursleiters

sich bereits angemeldet hat. Mit einem Klick in dieser Liste kann er somit zu dem Bildschirm wechseln, wo die Bewerbungsdaten geändert bzw. die Endergebnisse seiner Einteilung gefunden werden können. Falls der Benutzer sich für eine weitere Veranstaltung anmelden will, kann er aus der Liste aller über das System verwalteten Veranstaltungen die passende für sich auswählen.

Nachdem eine Veranstaltung ausgewählt wurde, deren Einteilungsfrist noch nicht abgelaufen ist, hat der Benutzer die Möglichkeit, seine drei Wunschgruppen auszuwählen und mit Prioritäten zu versehen. Neben jeder Gruppe, wie man auf der Abbildung 5.7 erkennen kann, stehen drei Ampeln. Diese Ampeln, wie im Kapitel 4.3 beschrieben wurde, bezeichnen die Popularität der dazugehörigen Gruppe für jede Priorität. Eine solche Darstellung hilft, gut einzuschätzen, wie viele Chancen man hat, in eine bestimmte Gruppe zu kommen. Damit ist die Auswahl recht einfach. Jeden Wunsch kann der Benutzer mittels nur eines Klicks angeben, was die Bedienung sehr bequem macht.

Damit wurden die Hauptideen des entwickelten Interfaces beschrieben. Alle weiteren Interfaceelemente wurden ähnlich gehalten. Eine gesamte Übersicht davon sprengt leider der Rahmen dieser Arbeit.

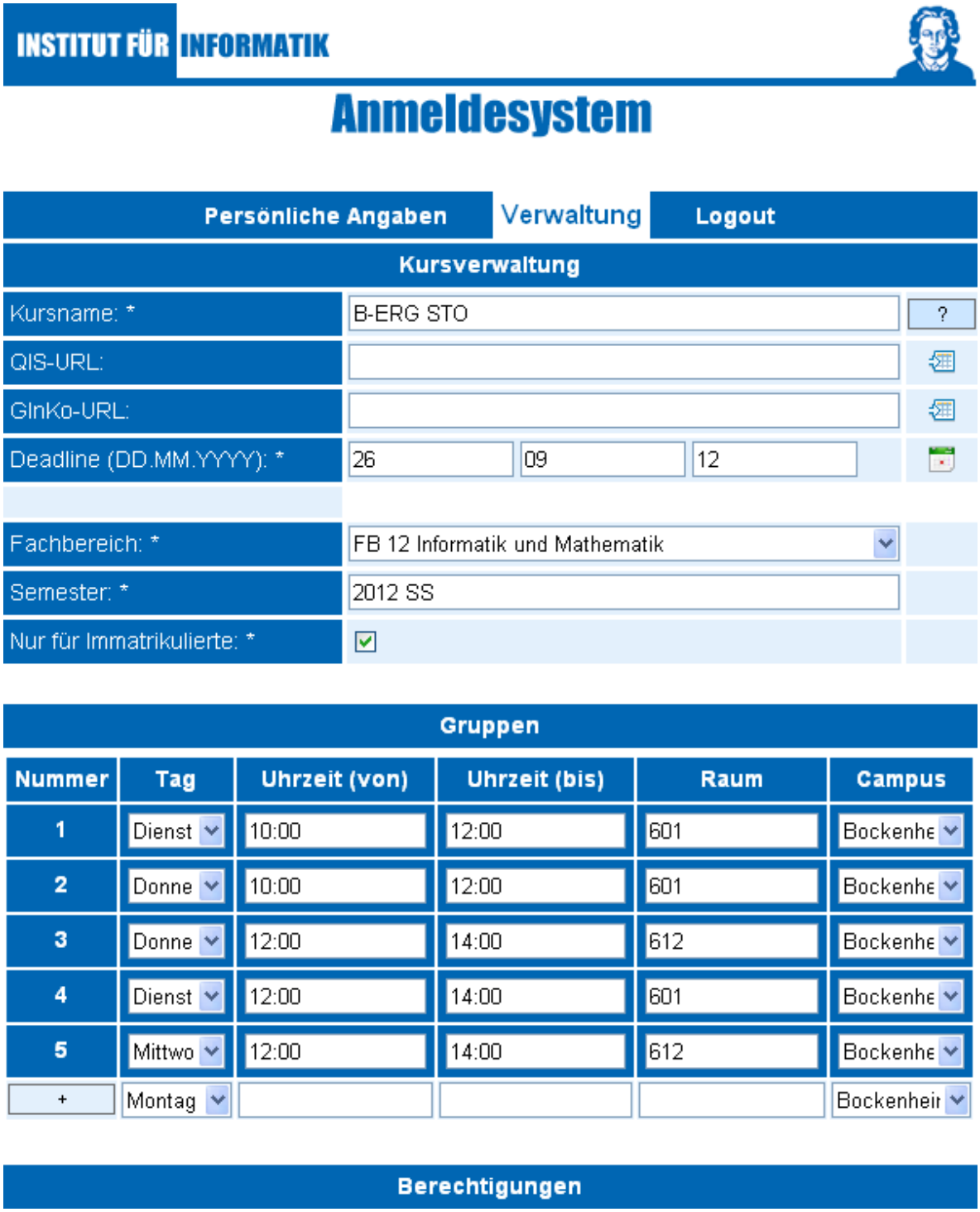


Abbildung 5.6: Verwaltung eines Kurses

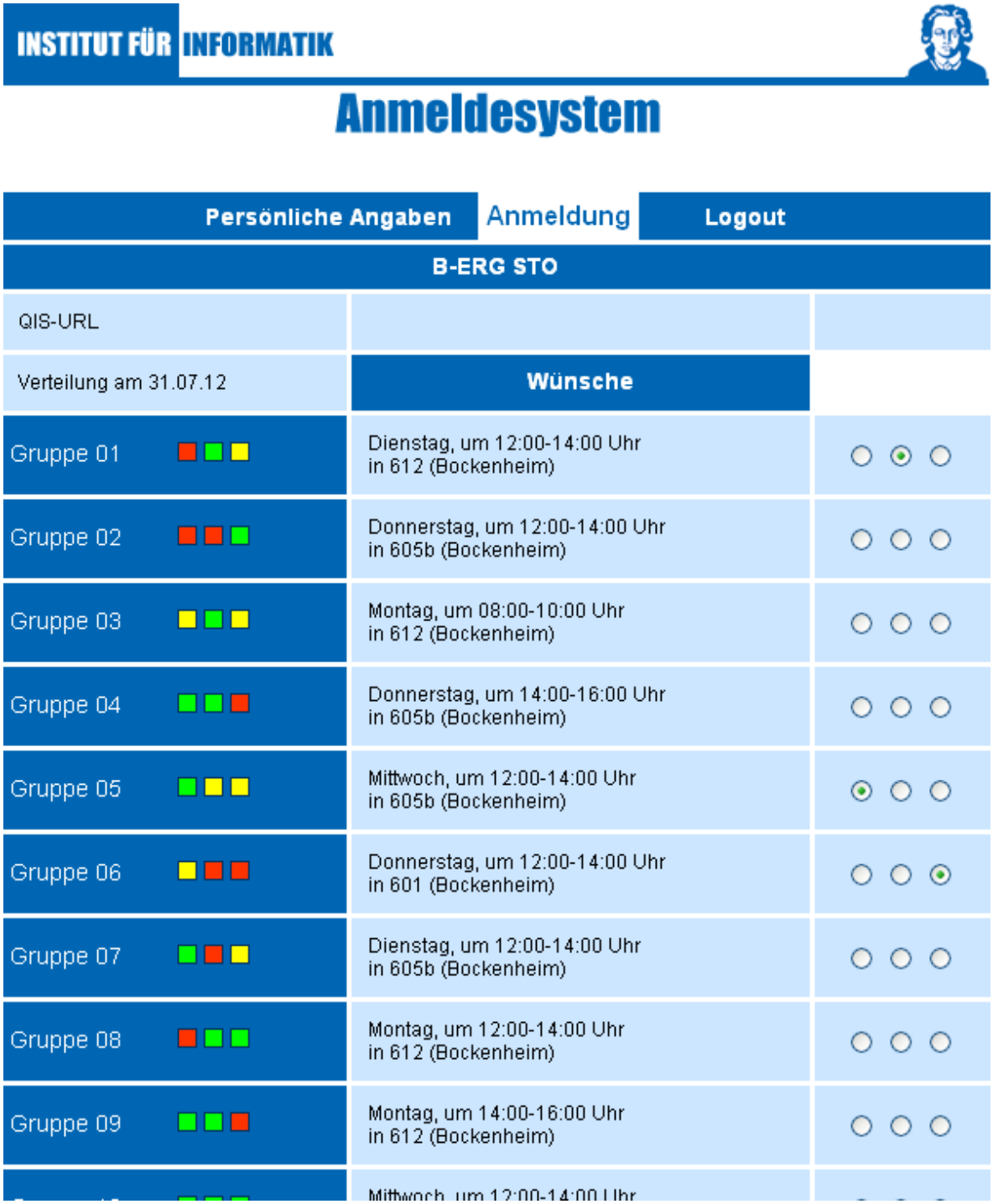


Abbildung 5.7: Bewerbung für einen Kurs

5.4 Bedingungen des Tests

5.4.1 Die Anmeldedaten

Für die Evaluation der entwickelten Heuristik und des damit implementierten Gruppeneinteilungssystems wurden anonymisierte Anmeldedaten folgender Veranstaltungen genommen:

1. **STO WS 2011/12** Studiumsorientierungsveranstaltung (STO) *Einführung in das Studium* vom Wintersemester 2011/2012 mit 147 Teilnehmer und 3×2 parallel angebotenen Gruppen[4].
2. **STO SS 2012** STO *Einführung in das Studium* vom Sommersemester 2012 mit 55 Teilnehmer und keinen parallel angebotenen Gruppen.
3. **PRG1 WS 2008/09** *Grundlagen der Programmierung 1* vom Wintersemester 2008/2009 mit 168 Teilnehmer und 3×2 parallelen Gruppen.
4. **PRG1 WS 2009/10** *Grundlagen der Programmierung 1* vom Wintersemester 2009/2010 mit 252 Teilnehmer und 3×2 parallelen Gruppen.
5. **PRG1 WS 2010/11** *Grundlagen der Programmierung 1* vom Wintersemester 2010/2011 mit 249 Teilnehmer und keinen parallel angebotenen Gruppen.
6. **PRG1 WS 2011/12** *Grundlagen der Programmierung 1* vom Wintersemester 2011/2012 mit 293 Teilnehmer und 2×2, 1×4 und 1×3 parallel angebotenen Gruppen.

Bei allen davon handelt es sich um die Pflichtveranstaltungen des Studiengangs Bachelor Informatik der Goethe-Universität Frankfurt. Die Plätze für den Übungsbetrieb aller Pflichtveranstaltungen müssen von der Universität allen Interessenten angeboten werden, damit die Fristen der Regelstudienzeit einhaltbar sein können.

Für jede der Veranstaltungen wurde mit dem implementierten Gruppeneinteilungssystem eine Einteilung in allen Güteklassen, d.h. nur nach 1. Wünschen, nach 1. und 2. Wünschen und nach allen drei Wünschen durchgeführt.

Die Einteilung wurde einmal mit der Erkennung paralleler Gruppen und einmal ohne durchgeführt: Bei der Einteilung in parallele Gruppen können gleichmäßigere Ergebnisse erwartet werden. Dabei müssen allerdings vom Programm neue (virtuelle) Bewerbungen beachtet werden, was die Ausführungsdauer vergrößern wird.

Bei der Laufzeitberechnung wurden alle Tests jeweils fünf mal wiederholt und die mittlere Ausführungsdauer wurde notiert. Damit wurde der Einfluss zufälliger Ereignisse auf die Ergebnisse reduziert.

[4] D.h. dreimal wurden jeweils zwei Gruppen in der gleichen Zeit am gleichen Campus angeboten

5.4.2 Das Testsystem

Die Tests wurden auf einem ordinären Computer durchgeführt, dessen relevante Eigenschaften in der Tabelle 5.1 aufgelistet sind.

Es ist zu beachten, dass mit einem dedizierten Server deutlich bessere Ergebnisse erreicht werden könnten.

Prozessor:	Core i5 750 (ohne Overclocking)
Arbeitspeicher:	3.2 GB
Betriebsystem:	Microsoft Windows XP SP3
Web-Server:	Apache 2.2.4
Datenbankserver:	MySQL 5.1.40
PHP-Version:	5.3.3

Tabelle 5.1: Parameter des Testsystems

5.5 Laufzeit

Wie man in der Tabelle 5.2 sieht, ist die Laufzeit des Algorithmus in allen Fällen praxistauglich.

Wie erwartet, dauert die Einteilung unter Beachtung paralleler Gruppen bei allen Kursen länger, die parallele Gruppen überhaupt haben. Bei der letzten Veranstaltung wird dafür insgesamt doppelt so viel Zeit benötigt. Das liegt an einer sehr großen Anzahl paralleler Gruppen, die die Anzahl der Bewerbungen für den Algorithmus (lokal) stark vergrößert haben.

Kurs	PV?	BP	1	2	3	4	5	Durchschnitt
STO WS 2011/12	✓		0.097	0.148	0.150	0.154	0.069	0.124
	✓	✓	0.121	0.159	0.101	0.151	0.139	0.134
STO SS 2012			0.071	0.046	0.058	0.038	0.059	0.054
		✓	0.061	0.049	0.048	0.030	0.068	0.051
PRG1 WS 2008/09	✓		0.167	0.121	0.138	0.136	0.130	0.138
	✓	✓	0.214	0.207	0.165	0.208	0.203	0.199
PRG1 WS 2009/10	✓		0.175	0.227	0.240	0.163	0.189	0.199
	✓	✓	0.257	0.260	0.303	0.252	0.311	0.277
PRG1 WS 2010/11			0.135	0.218	0.215	0.237	0.173	0.196
		✓	0.232	0.199	0.230	0.219	0.186	0.213
PRG1 WS 2011/12	✓		0.187	0.202	0.204	0.232	0.264	0.218
	✓	✓	0.528	0.561	0.496	0.511	0.453	0.510

Tabelle 5.2: Laufzeit der Einteilung. Alle Ergebnisse sind in Sekunden.
PV? steht für *Parallelität vorhanden?* BP steht für *Beachtung der Parallelität*

5.6 Ergebnisse der Einteilung

Auf den Abbildungen 5.8-5.23 findet man die Graphen mit der Zusammenfassung der Ergebnisse der durchgeführten Einteilung.

Die mittleren Abweichungen (mittlere Fehler) der Gruppengrößen von der Durchschnittsgröße wurden jeweils ausgerechnet und oben mittig als *m.F.* angegeben. Der Durchschnitt mit der aktuellen Gruppenbelegung erfüllter Wünsche wurde unter den mittleren Abweichungen als *Erfüllung* angegeben. Durchschnittliche Gruppengrößen sind auf jeder Abbildung im Form einer rot gestrichelten horizontalen Linie erkennbar.

Grüne Balken bezeichnen die Anzahl der Teilnehmer, deren erste Wünsche mit der aktuellen Belegung erfüllt wurden. Die gelben und roten Balken stehen entsprechend für die zweiten und dritten Wünsche.

Bei der eingeschalteten Beachtung paralleler Termine werden zur gleichen Zeit am gleichen Campus stattfindende Termine einander gleichgesetzt. Aus diesem Grund haben die Wünsche, die mit parallelen Gruppen erfüllt wurden, die gleichen Farben wie die „normal“ erfüllten Wünsche.

Wie erwartet, steigt die durchschnittliche Erfüllung mit der Anzahl der beachteten Wünsche. Dabei sinkt allerdings die mittlere Abweichung der Gruppengrößen, d.h. Gruppen werden gleichmäßiger verteilt.

Alle Abbildungen, bis auf 5.20 vergleichen zwei unterschiedliche Einteilungsmodi: Mit der Parallelitätserkennung (oben) und ohne diese (unten). Wir man auf allen Abbildungen sehen kann, die sich auf die Veranstaltungen beziehen, wo parallele Gruppen angeboten wurden, werden *mit* der eingeschalteten Funktion die Ergebnisse besser: Die mittlere Abweichung **und** die Erfüllung der Wünsche sinken, da dem Programm mehr Möglichkeiten zur Auswahl stehen.

Die einzige Ausnahme ist auf der Abbildung 5.17 zu sehen: Die mittlere Abweichung ist bei der Verwendung der Parallelitätserkennung um Einiges schlechter geworden, da einige Gruppen in der falschen Reihenfolge gefüllt wurden. Das liegt hauptsächlich daran, dass der vorgestellte Algorithmus kein optimales Ergebnis liefert. Dieses Problem kann durch die Verbesserung der Heuristik vermieden werden, wie das im Kapitel 6 beschrieben wird.

Die Ergebnisse der Versuche haben bestätigt, dass bei der Verwendung der Parallelitätserkennung bei den Veranstaltungen, die keine Gruppen parallel anbieten, die Ergebnisse sich von denen ohne Parallelitätserkennung nicht unterscheiden. Abbildungen 5.11, 5.12 und 5.13 demonstrieren das graphisch. Aufgrund der Analogie von erzielten Ergebnissen zeigt die Abbildung 5.20 gleichzeitig die Ergebnisse der Einteilung mit und ohne der Parallelitätserkennung und erspart damit zwei Seiten dieser Arbeit.

Bei der Verwendung von allen Lösungen sind die Gruppengrößen sehr nah (im Falle der Abbildung 5.11 sogar gleich) an den Durchschnitt. Das ist sehr gut unter Beachtung der begrenzten Raumkapa-

zitäten und der nicht unendlich teilbaren Aufmerksamkeit der Übungsleitern.

Die Frage, welche Güteklasse die besten Lösungen liefert, kann immer noch nicht pauschal objektiv beantwortet werden:

Betrachten wir die Abbildungen 5.22 und 5.23. Insbesondere interessant ist der Fall mit der eingeschalteten Parallelitätserkennung: Um die mittlere Abweichung von 2.09 auf 1.44 zu senken, müsste man die gesamte Erfüllung nur um 0.01 (von 1.16 auf 1.17) verschlechtern. Obwohl das in diesem Fall eine effizientere Füllung der Gruppe 13 bedeuten würde, müssten sich vier Teilnehmer anstatt mit ihren ersten und zweiten Wünschen mit den dritten Wünschen zufrieden geben.

Daher ist die vorgeschlagene Lösung, wo Kursteilnehmer die ihrer Meinung nach beste Verteilung auswählen, weiterhin sinnvoll.

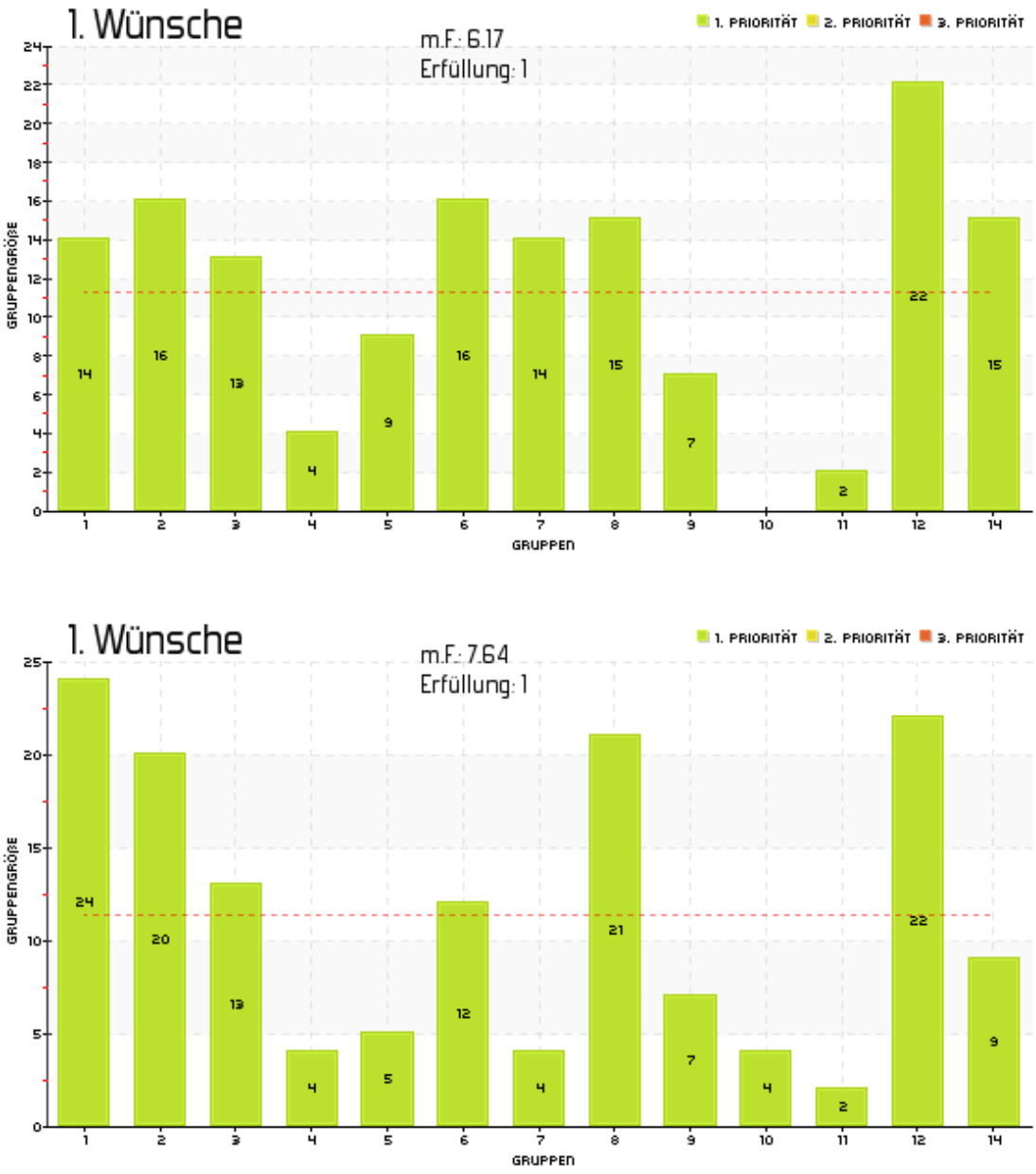


Abbildung 5.8: STO WS 2011/12: Einteilung anhand 1. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

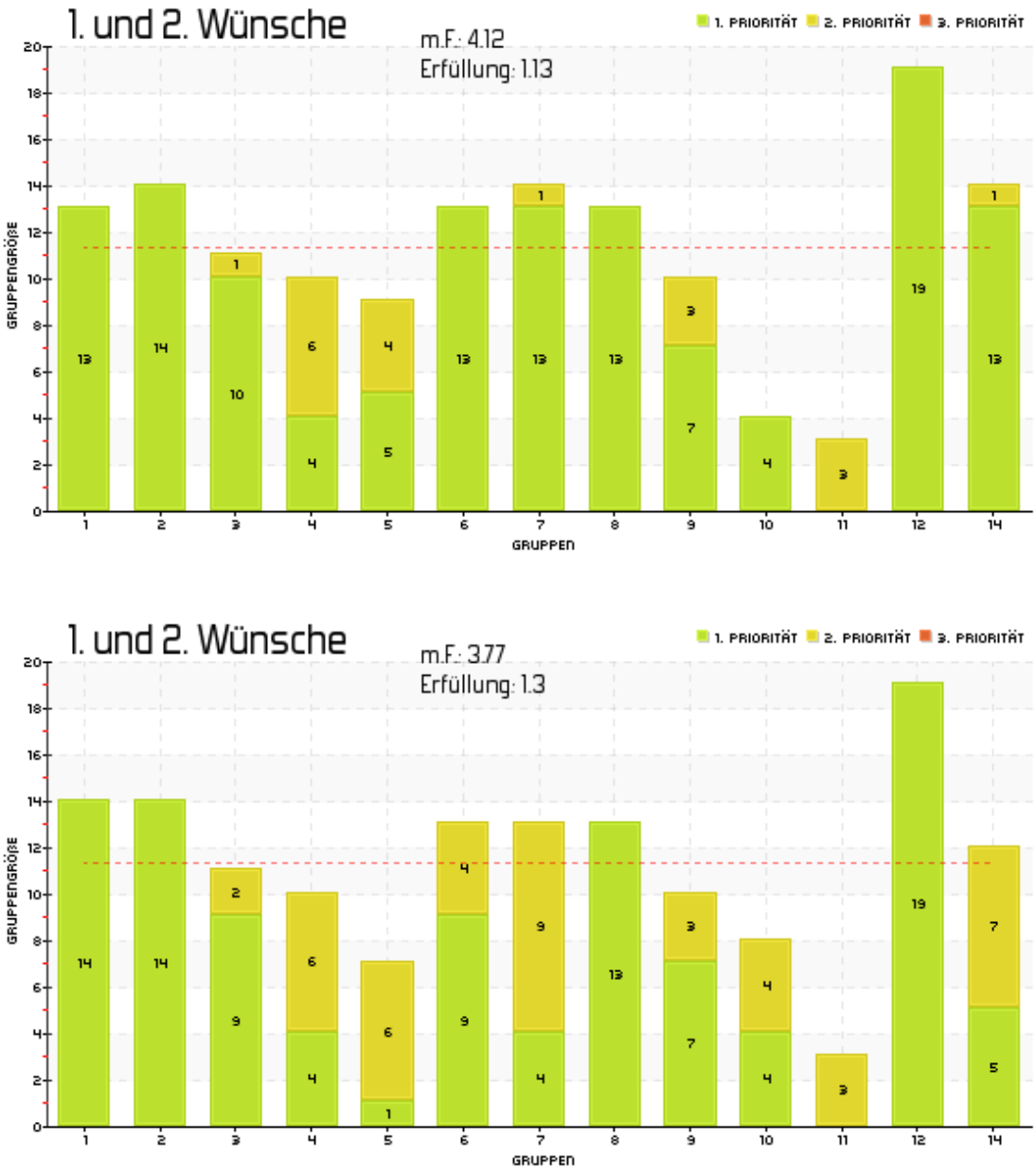


Abbildung 5.9: STO WS 2011/12: Einteilung anhand 1. und 2. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

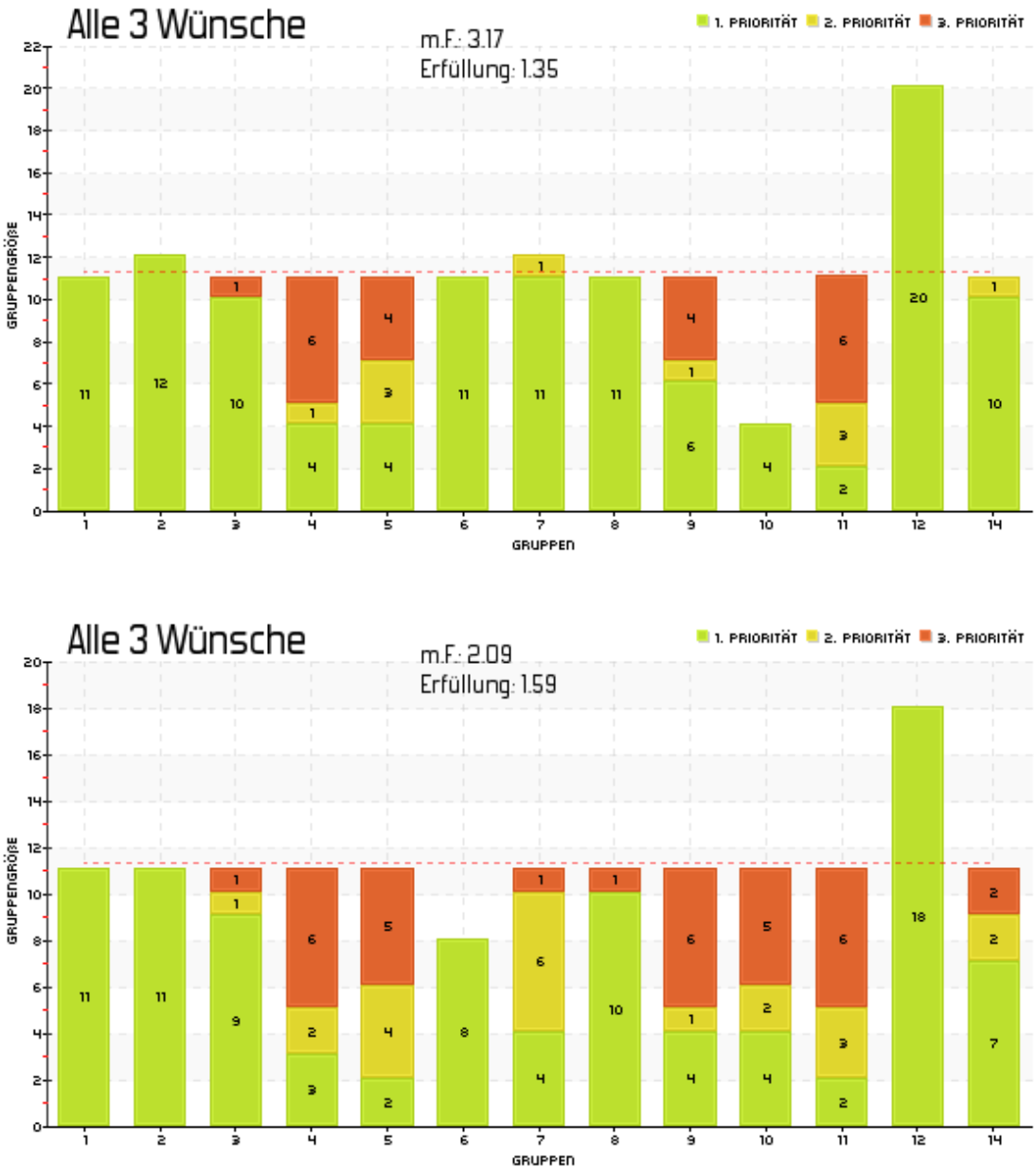


Abbildung 5.10: STO WS 2011/12: Einteilung anhand aller Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

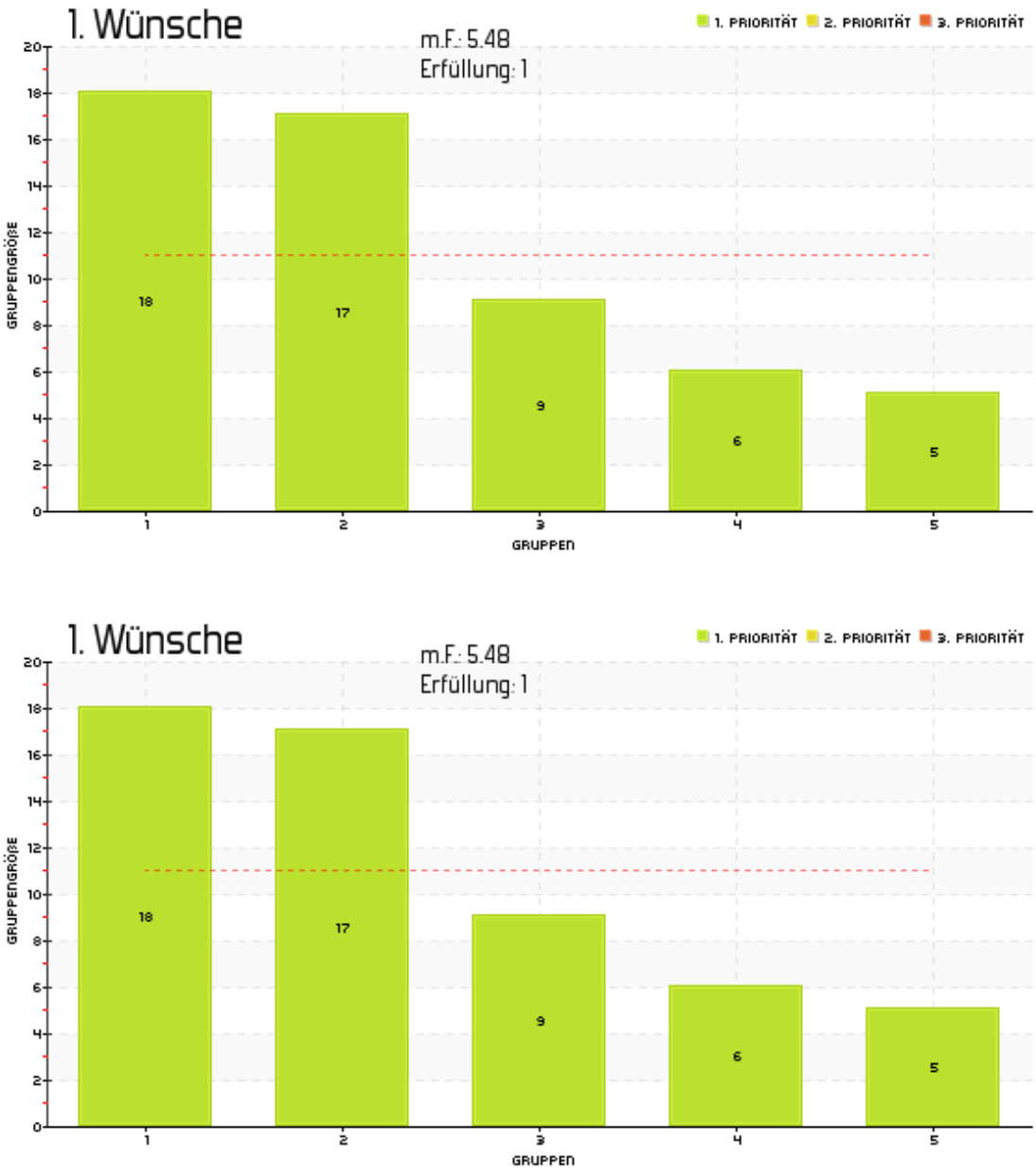


Abbildung 5.11: STO SS 2012: Einteilung anhand 1. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

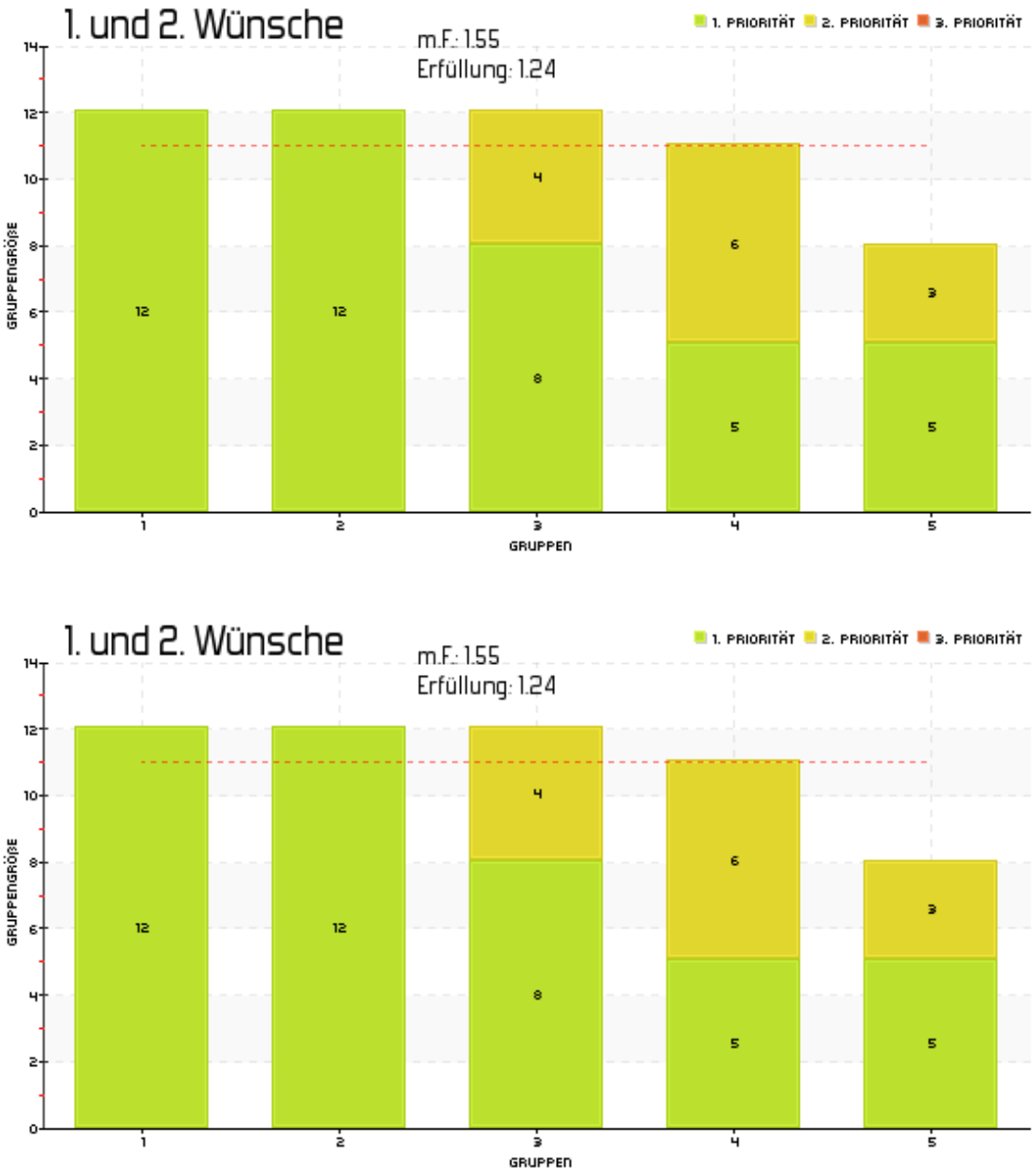


Abbildung 5.12: STO SS 2012: Einteilung anhand 1. und 2. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

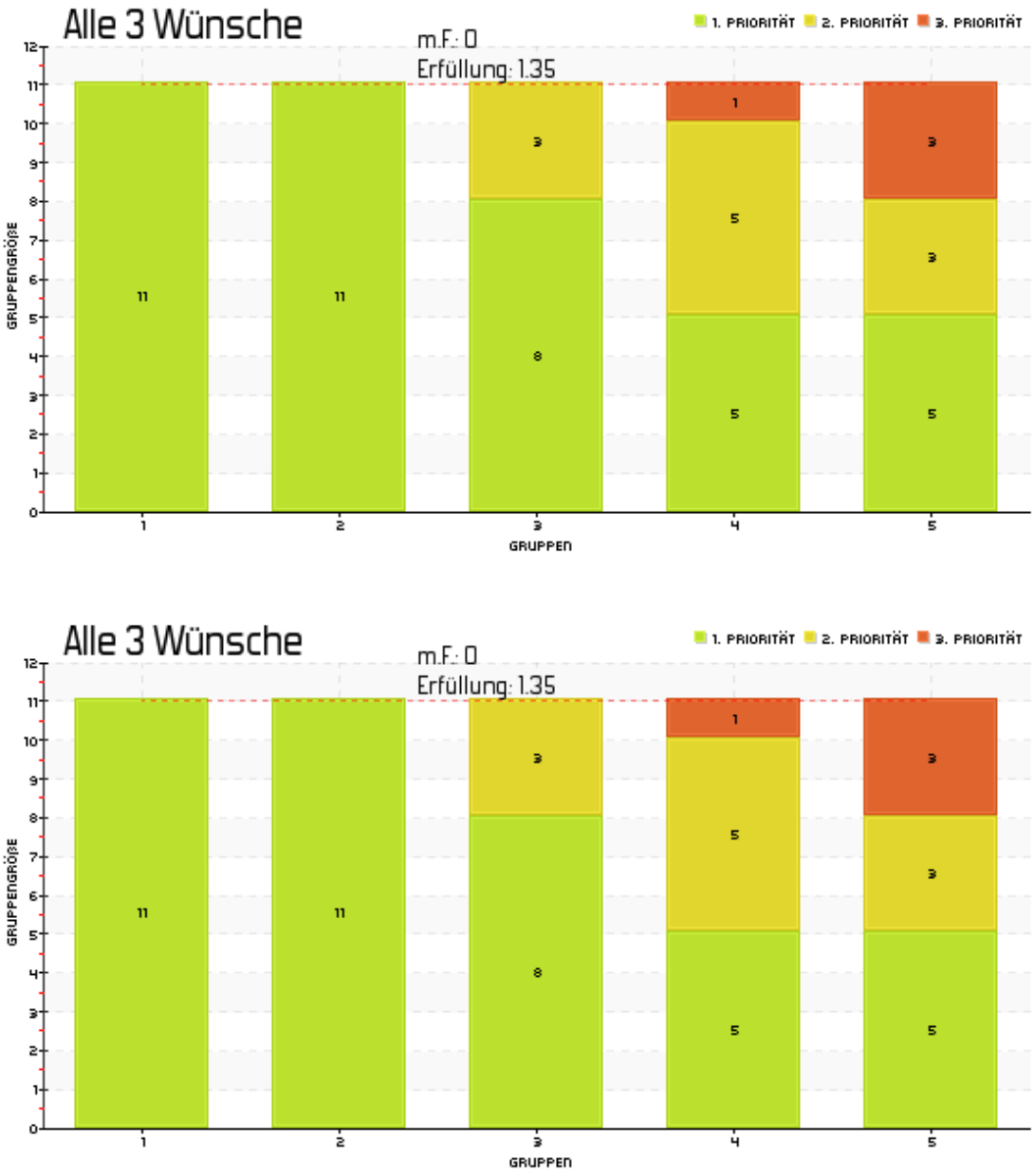


Abbildung 5.13: STO SS 2012: Einteilung anhand aller Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

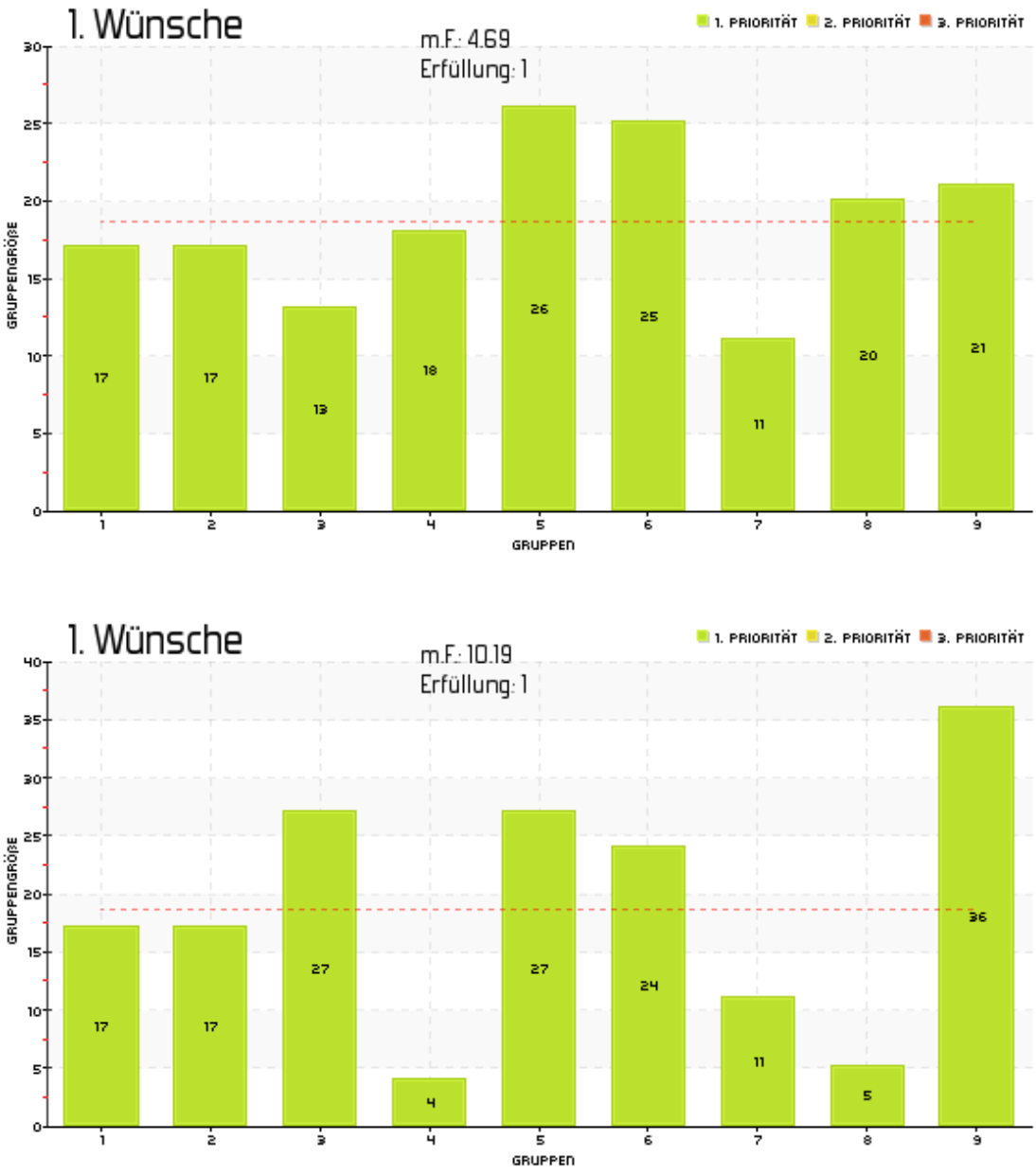


Abbildung 5.14: PRG1 WS 2008/09: Einteilung anhand 1. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

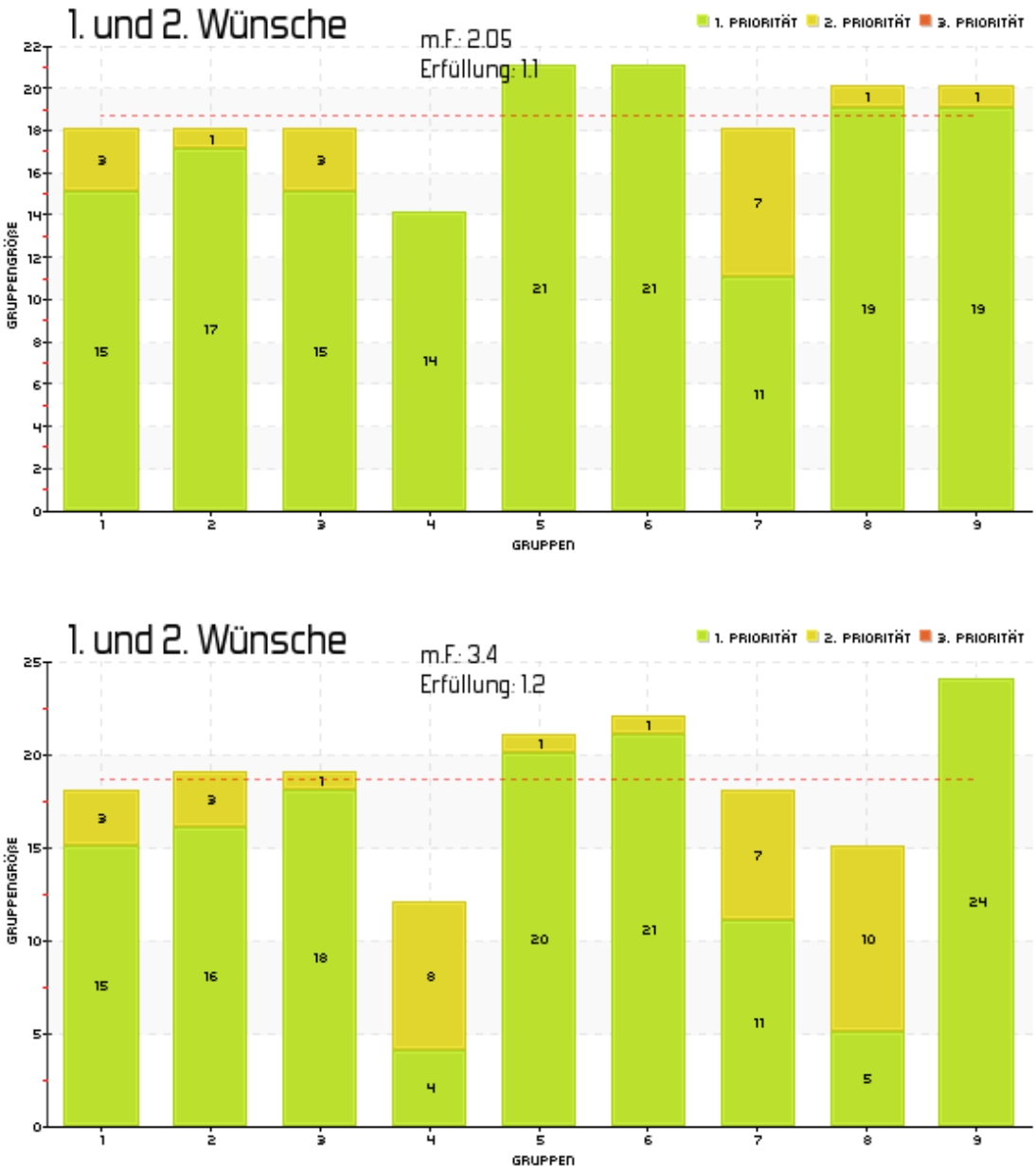


Abbildung 5.15: PRG1 WS 2008/09: Einteilung anhand 1. und 2. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

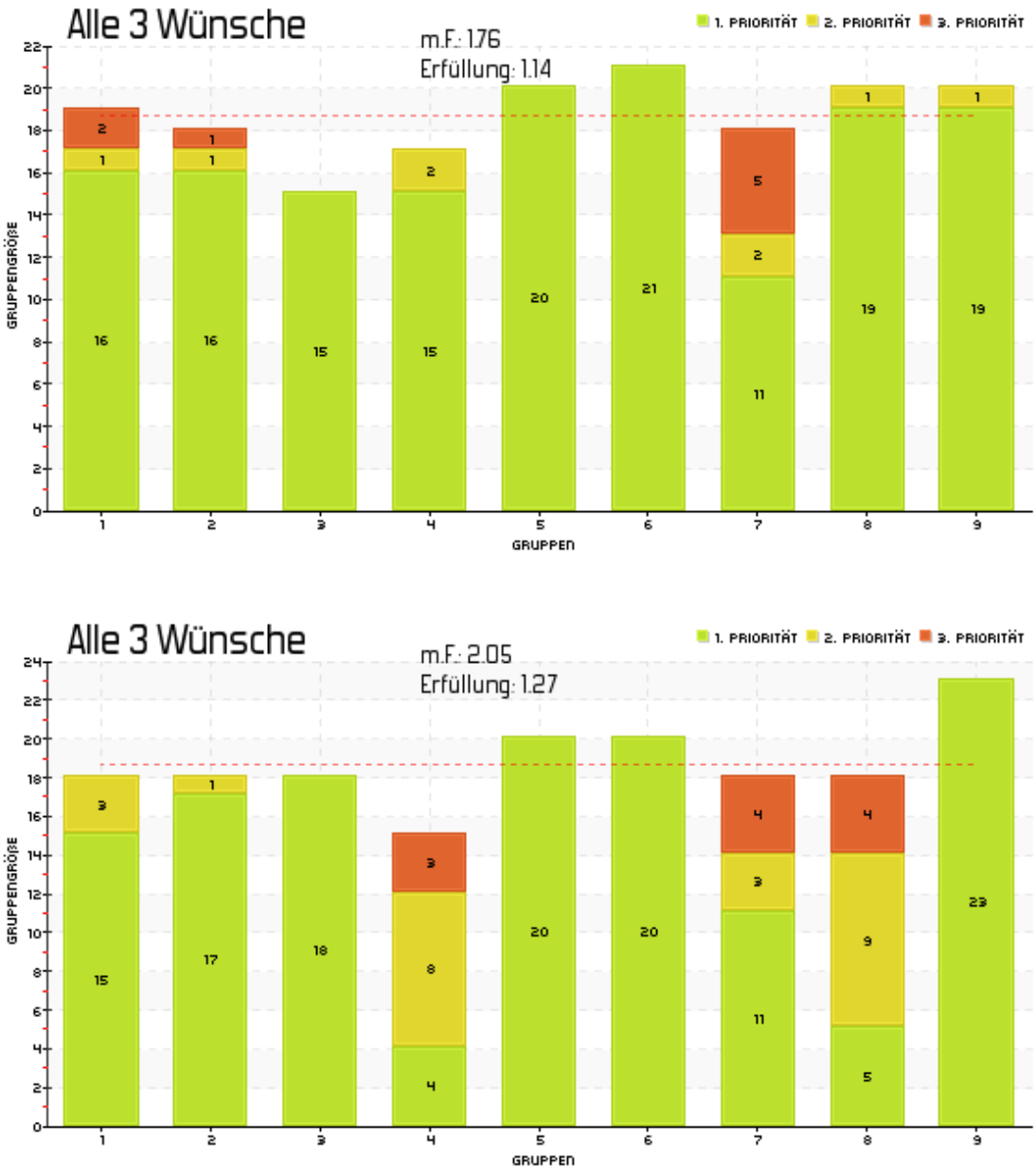


Abbildung 5.16: PRG1 WS 2008/09: Einteilung anhand aller Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

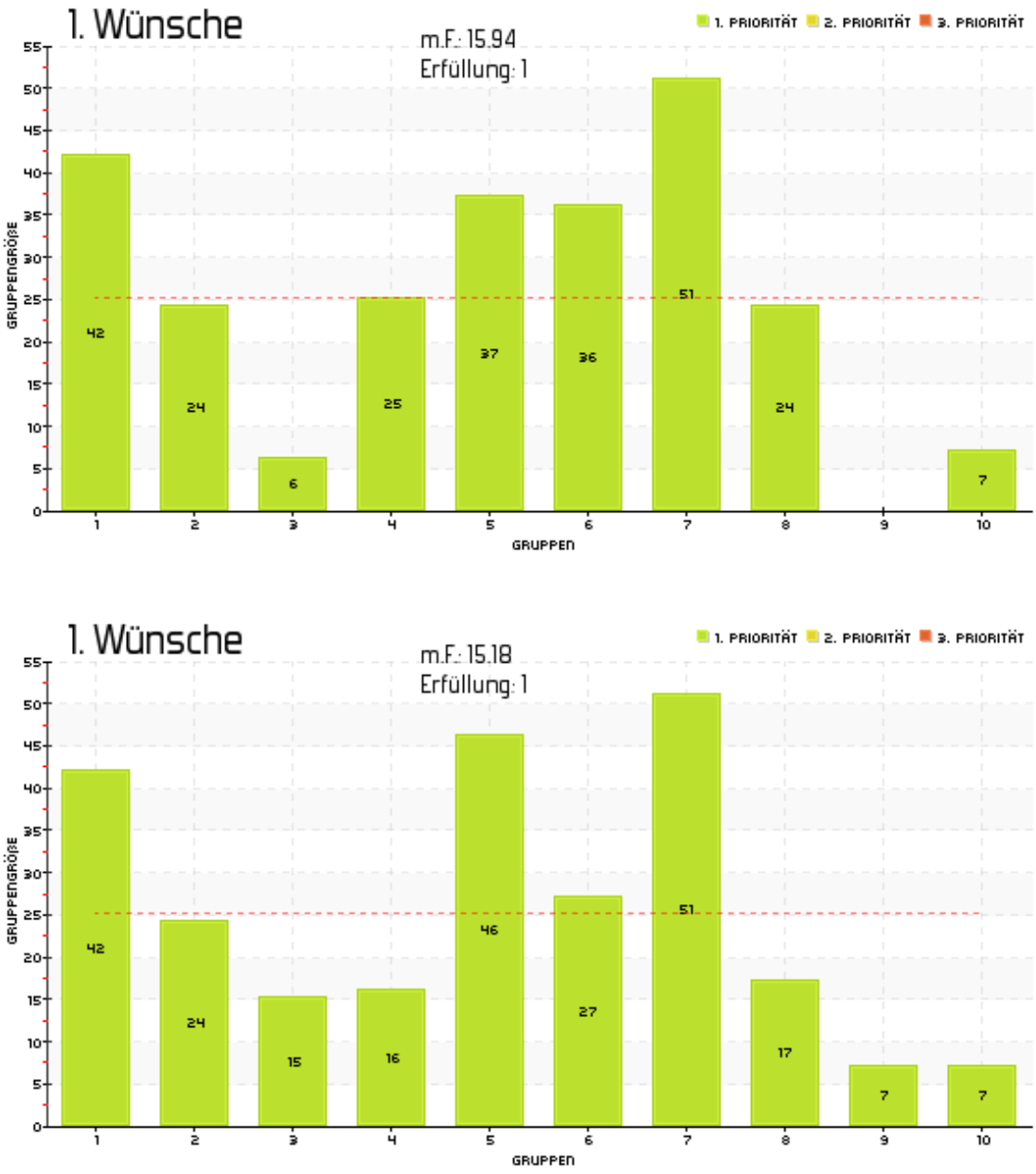


Abbildung 5.17: PRG1 WS 2009/10: Einteilung anhand 1. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

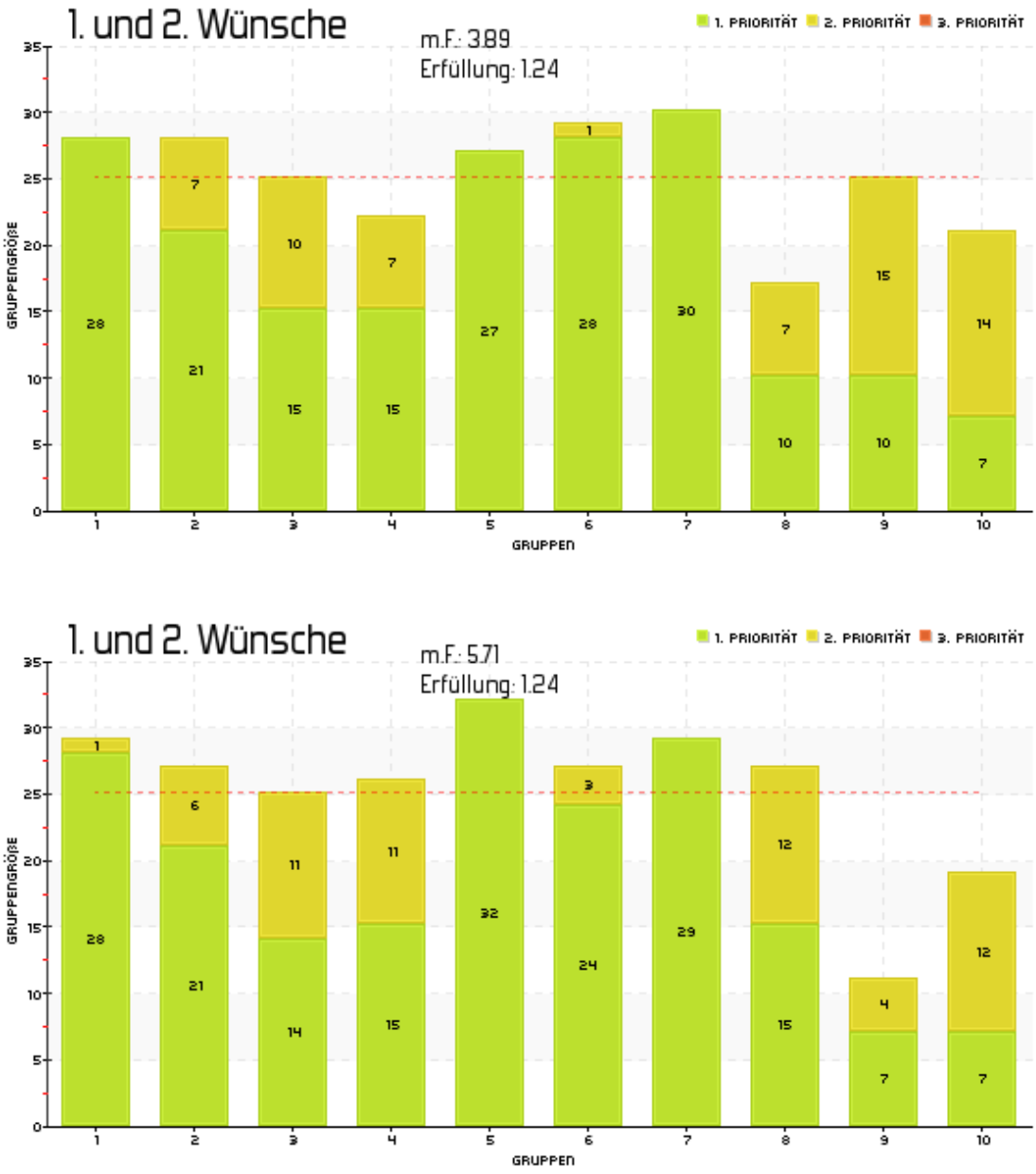


Abbildung 5.18: PRG1 WS 2009/10: Einteilung anhand 1. und 2. Wünsche mit (oben) und ohne (unten) Parallelitätserkennung

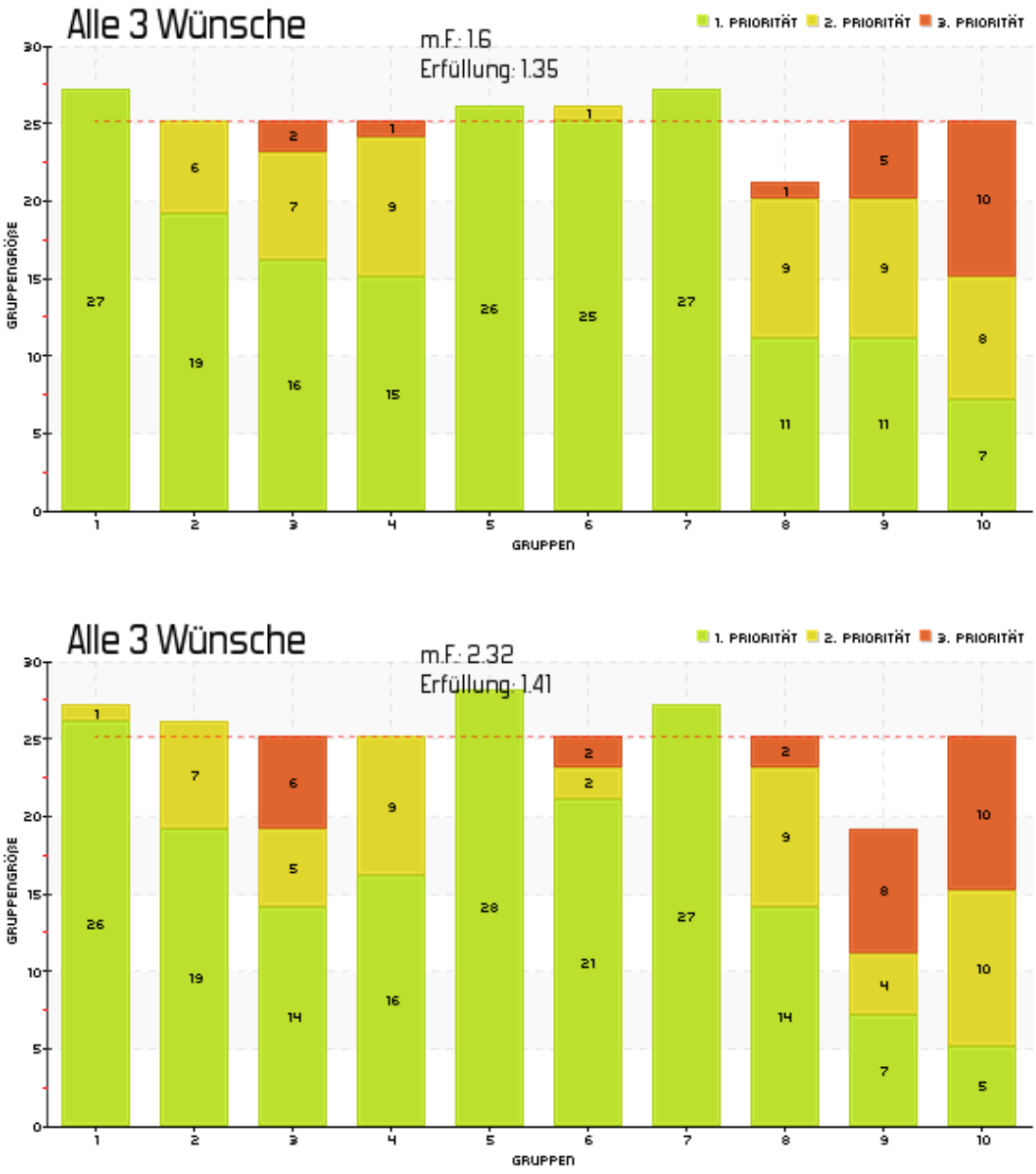


Abbildung 5.19: PRG1 WS 2009/10: Einteilung anhand aller Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

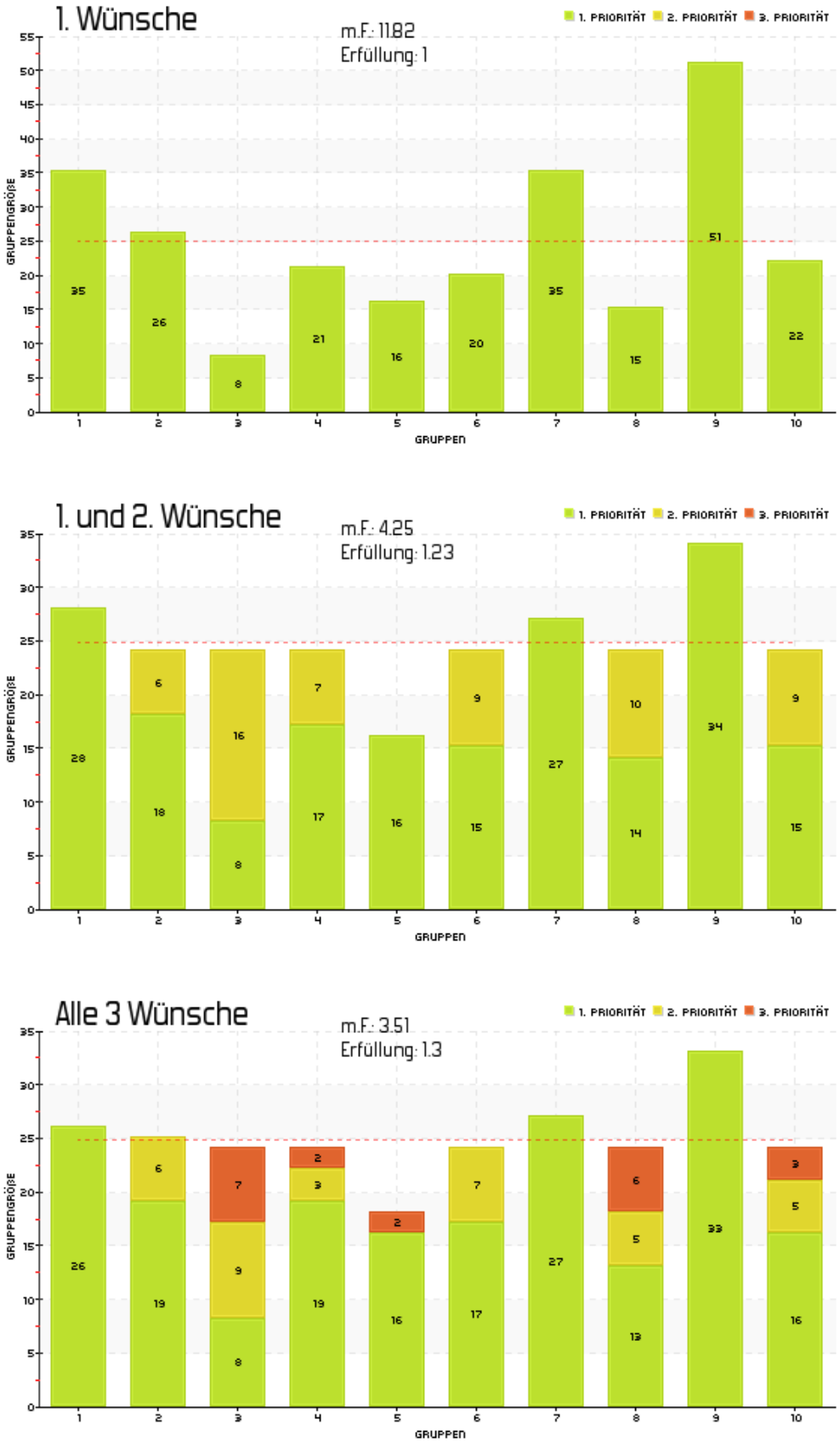


Abbildung 5.20: PRG1 WS 2010/11
Einteilung anhand 1. Wünsche (oben), anhand 1. und 2. Wünsche (mittig) und anhand aller Wünsche (unten)

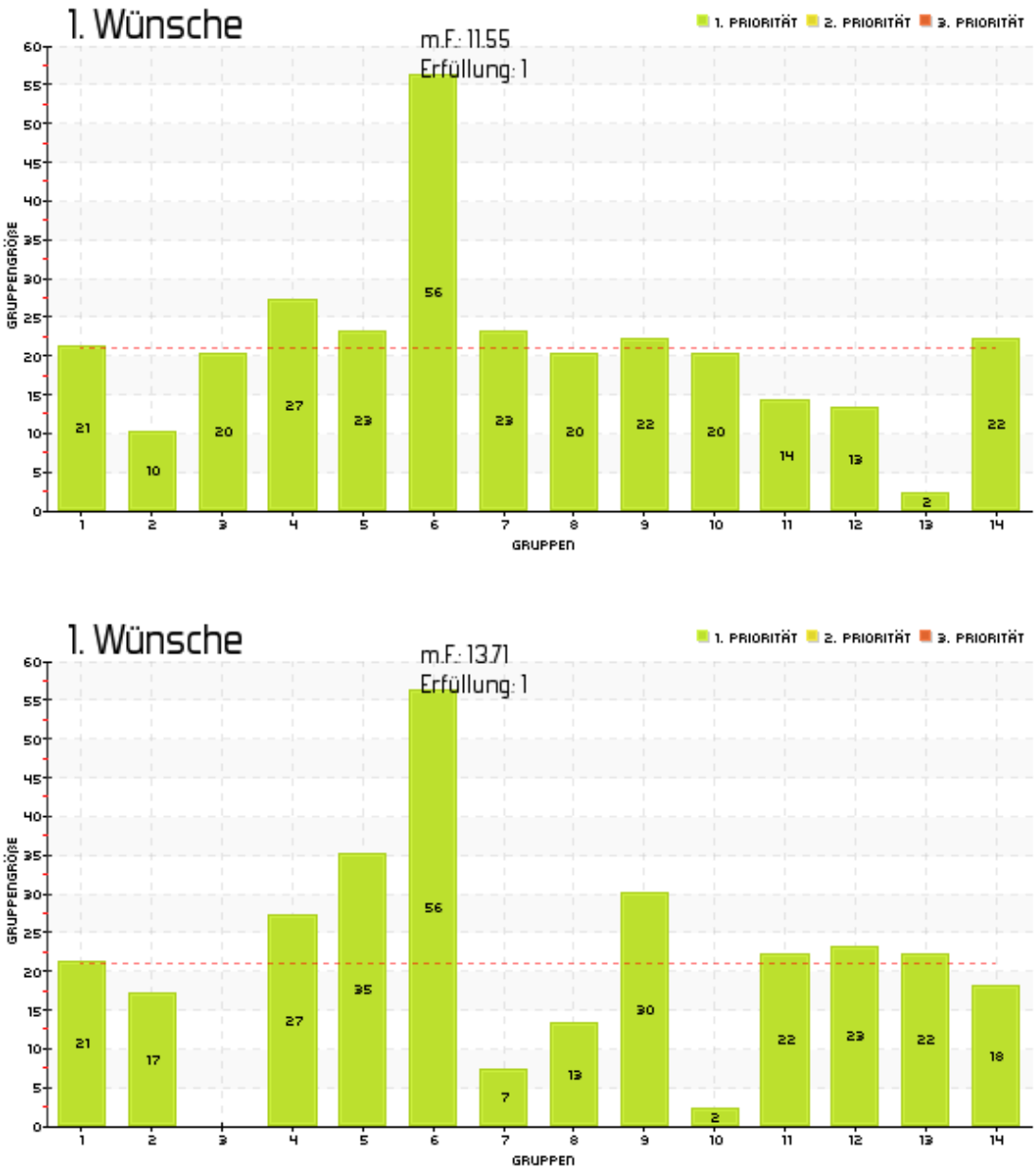


Abbildung 5.21: PRG1 WS 2011/12: Einteilung anhand 1. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

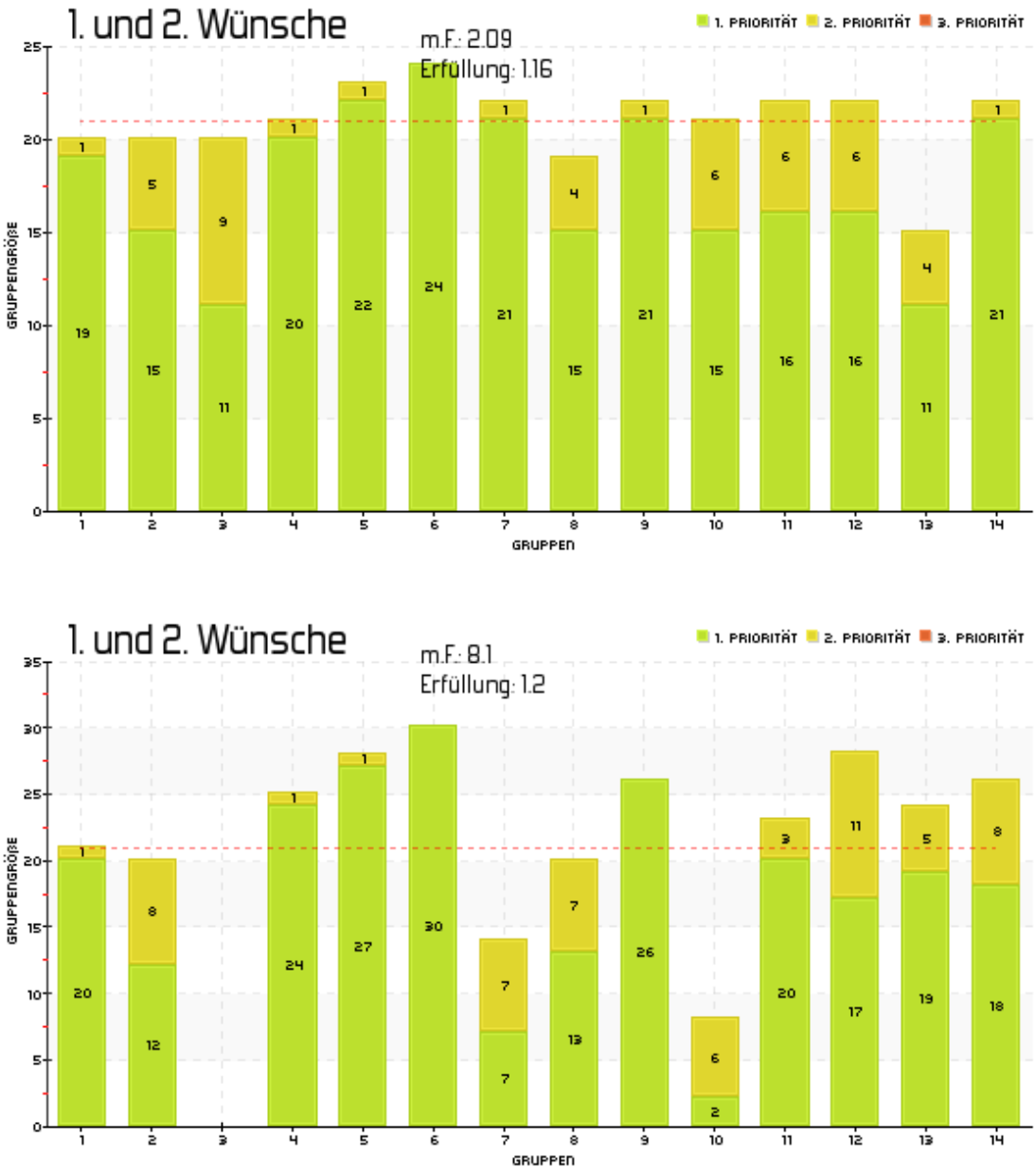


Abbildung 5.22: PRG1 WS 2011/12: Einteilung anhand 1. und 2. Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

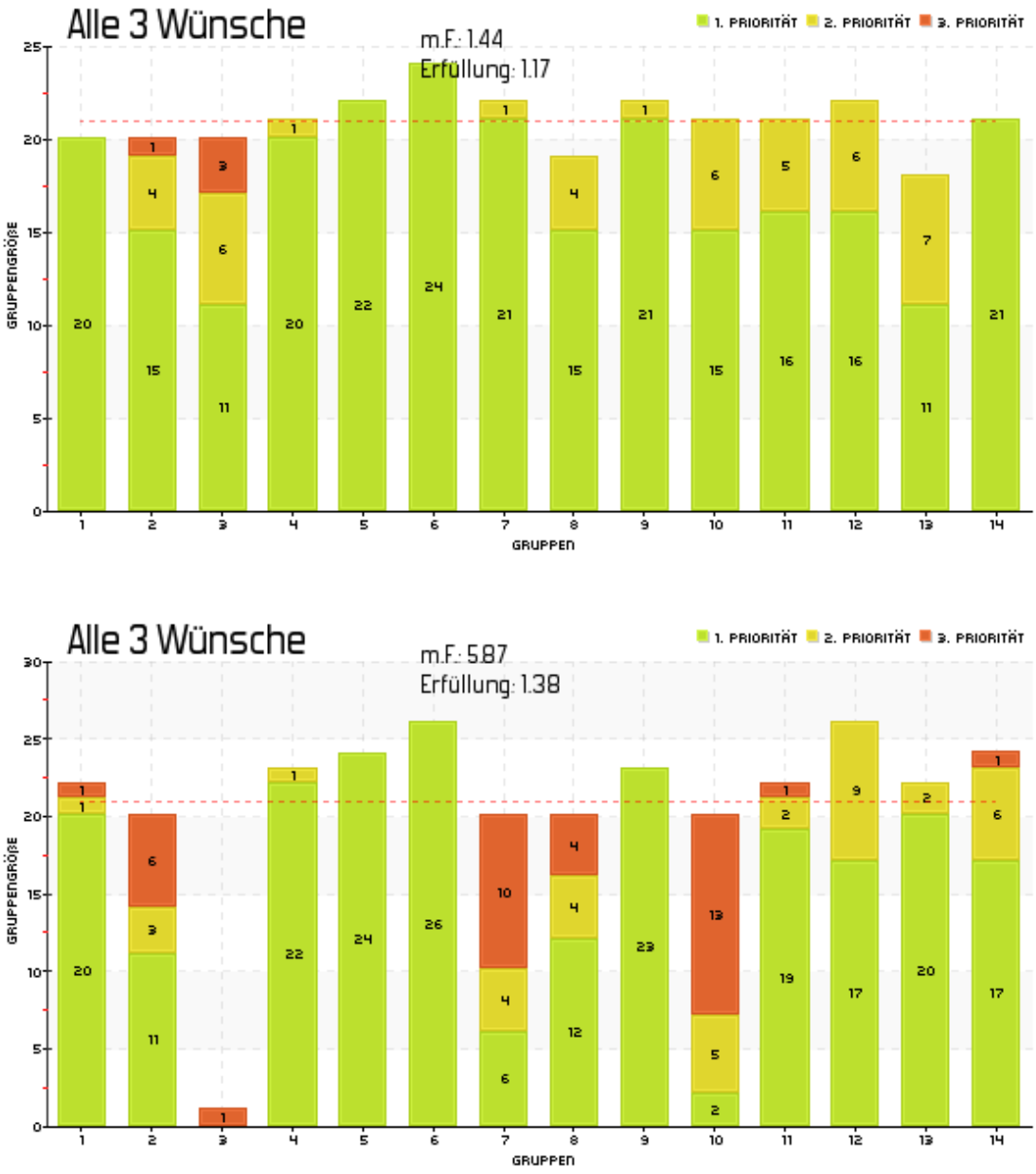


Abbildung 5.23: PRG1 WS 2011/12: Einteilung anhand aller Wünsche
mit (oben) und ohne (unten) Parallelitätserkennung

Kapitel 6

Ausblick

Obwohl die vorgeschlagene Heuristik gute und praxistaugliche Ergebnisse liefert, sind diese Ergebnisse immer noch nicht optimal.

Eine deutliche Verbesserung wäre möglich, wenn beim Schritt der Füllung von Gruppen (siehe Listing 4.3) die Abarbeitung nicht nach den Gruppennummern, sondern nach den theoretisch maximal erreichbaren Gruppengrößen erfolgen würde. Dabei müssten die kleinsten Gruppen zuerst gefüllt werden.

Das würde länger dauern, die Verschlechterung der Zeit um einige Prozent wäre allerdings aufgrund bisherig geringer Dauer akzeptabel (siehe Abschnitt 5.5). Aus Zeitgründen wurde diese kleine Verbesserung leider nicht implementiert.

In Zukunft ist zu erwarten, dass globale Campus-Verwaltungssysteme, wie beispielsweise das in der Goethe-Universität Frankfurt momentan geplante GInKo[1], sich soweit entwickeln, dass sie befriedigend implementierte Gruppeneinteilungsfunktionalitäten anbieten werden. Das könnte allerdings einige Jahren dauern.

Als Folge werden dedizierte Gruppeneinteilungssysteme, wie der in Rahmen dieser Bachelorarbeit implementierte Prototyp, überflüssig.

Die entwickelte Heuristik könnte allerdings auch für die Campus-Verwaltungssysteme der Zukunft interessant sein, zumindest als eine Grundlage für die Weiterentwicklung.

Wenn die Forschung keine Möglichkeiten der deterministischen Berechnung NP-schwerer Probleme in polynomialer Zeit findet, werden solche Heuristiken auch in weiteren Jahren für die Lösung des Gruppeneinteilungsproblems verwendet.

[1] **G**oethe Universität **In**formations- und **Ko**mmunikationssystem

Kapitel 7

Zusammenfassung

In dieser Arbeit wurde beschrieben, dass das Gruppeneinteilungsproblem aus theoretischer Sicht als ein NP-schweres Problem gesehen werden kann. Daher ist dieses wahrscheinlich nicht deterministisch in polynomialer Zeit lösbar. Als Folge braucht jedes gute Gruppeneinteilungssystem eine geeignete Heuristik für ein zufriedenstellendes Ergebnis.

Die gängigsten in Deutschland, insbesondere an der Goethe-Universität Frankfurt, verwendeten Gruppeneinteilungssysteme wurden analysiert. Festgestellt wurde, dass alle davon offensichtliche Mängel aufweisen. Daher ist die Entwicklung eines System ohne diese Mängel sinnvoll.

Nachdem die theoretisch passenden Algorithmen betrachtet wurden, wurde festgestellt, dass eine neue spezielle Heuristik mit geringer Komplexität sinnvoll wäre.

Eine neue Heuristik wurde entwickelt. Es wurde bestätigt, dass sie mit allen betrachteten Anforderungen zu den Gruppeneinteilungssystemen kompatibel ist.

Anschließend wurde ein Gruppeneinteilungssystem vorgestellt, das alle im Kapitel 3.3 für die Evaluation der Gruppeneinteilungssysteme eingeführten Kriterien erfüllen kann.

Das Laufzeitverhalten des Prototyps wurde analysiert und es wurde festgestellt, dass dieses praxistauglich ist. Anhand der Ergebnisse praktischer Tests mit den Anmeldedaten aus vergangenen Jahren wurde gezeigt, dass die vorgestellte Heuristik gute Ergebnisse liefert, was die praktische Verwendbarkeit des vorgestellten Algorithmus bestätigt hat.

Abschließend wurde ein Ausblick in die Zukunft der Gruppeneinteilung gegeben. Es wurde beschrieben, dass Gruppeneinteilungssysteme künftig wahrscheinlich nicht separat von den großen Verwaltungssystemen angeboten werden. Das Problem der effizienten, praxistauglichen Implementierung wird höchstwahrscheinlich auch weiterhin vorhanden sein.

Literaturverzeichnis

[1] S. Lin. *Computer solutions of the traveling salesman problem*, volume 44. Bell Syst. J., Boston, 1965.

[2] Jaikumar R. Fisher, M.L. and L. Van Wassenhove. *A multiplier adjustment method for the generalized assignment problem*. Management Science 32/9, 1986.

[3] Dirk G. Cattrysse and Luk N. Van Wassenhove. *A Survey of Algorithms for the Generalized Assignment Problem*. European Journal of Operational Research 60, North-Holland, 1990.

[4] R. J. Dakin. *A tree-search algorithm for mixed integer programming problems*, volume 8. The Computer Journal, 1965.

[5] M. Hönig. *Entwicklung und Implementierung eines Werkzeuges zur Lösung des Zuordnungsproblems auf Basis der Simplexmethode am Beispiel der Einstellung in den hessischen Schuldienst nach dem Ranglistenverfahren*. Phd thesis edition, 2002.

[6] H. Feltl. *Ein genetischer Algorithmus für das Generalized Assignment Problem*. Vienna, Austria, phd thesis edition, 2003.

[7] Florian Gnägi. *The LMS OLAT REST API and the OLAT mobile client*. 2011. http://www.slideshare.net/gnaegi/the-lms-olat-rest-api-and-the-olat-mobile-client [Zugegriffen am 04.09.2012].

[8] *Web services Roadmap*. 2012. http://docs.moodle.org/dev/Web_services_Roadmap [Zugegriffen am 04.09.2012].