**Entwicklungsprozess eines Verteilungsalgorithmus für Praktikumsanmeldungen an Hochschulen im Rahmen eines Praktikums für die Veranstaltung Software-Entwicklung 2**

**Grundproblematik**

Im aktuellen Anmeldungsprozess der HAW müssen sich immatrikulierte Studenten jedes Semester für Veranstaltungen anmelden und explizit angeben für welche Praktikumsgruppe sie dies tun möchten. Die Verantwortung einen konfliktfreien Stundenplan zu erhalten wird hierbei dem Studenten überlassen. Jeder Student muss sich zu Beginn jeder Anmeldephase also darüber informieren, ob die Praktikumsgruppe, die er belegen möchte, nicht dazu führt, dass er eine andere belegte Gruppe nicht mehr besuchen kann, weil die Gruppen Terminüberschneidungen haben. Die Aufgabe des Verteilungsalgorithmus besteht jetzt darin, diese Selbstverantwortung dem Studenten zu entziehen und sie zu automatisieren. Studenten sollen sich nur noch für ein Praktikum an sich anmelden und sich keine Gedanken mehr darüber machen, welche Praktikumsgruppe sie genau wählen müssen, um keine Überschneidungen zu erhalten.

Jetzt ist die Realität noch ein wenig komplizierter, als das Grundproblem ohnehin schon bereits ist. Die Praktika werden in der Regel nicht alleine, sondern in Teams bestehend aus meistens zwei (in seltenen Fällen sogar bis zu 10) Studenten bestritten. Deswegen soll es eine Möglichkeit geben, sich mit einem Team für eine Veranstaltung anmelden zu können. Auch hier muss die Zuteilung des Teams zu einer Praktikumsgruppe automatisiert geschehen.

Während der Entwicklung eines solchen Algorithmus wird schnell klar, dass es hierfür keine optimale Lösung geben kann. Selbst im aktuellen Anmeldeprozess bei dem die Verantwortung den Studenten selbst übergeben wird (auch, wenn wir davon ausgehen, dass sich kein Student fehlerhaft beim Anmelden verhält), kann es sein, dass obwohl eigentlich eine mögliche Verteilung existiert, sich nicht alle Studenten konfliktfrei für eine Praktikumsgruppe anmelden können. Der unten beschriebene erste Versuch eines Greedy-Algorithmus simuliert dies (betrachtet nur Einzelanmeldungen). Nimmt man die Teamanmeldungen hinzu lässt sich das Problem auf das Stundenplan-Problem übertragen bei dem man versucht eine geeignete Konstellation zu finden, so dass kein Lehrer zum Beispiel zwei Klassen gleichzeitig unterrichten kann. Das Stundenplanproblem wurde bisher noch nicht algorithmisch gelöst und ist sogar beweisbar nicht in polynomialer Zeit lösbar. Dasselbe gilt für diesen Verteilungsalgorithmus. Man kann nur versuchen mit einer guten Wahl von Heuristiken und Einschränkungen eine ansatzweise gute Lösung zu finden.

Um sich diesem Problem zu nähern und möglicherweise irgendwann zu einer passablen Lösung zu gelangen, ist es sinnvoll, das Problem zunächst zu vereinfachen und Kompromisse einzugehen

**Die erste Algorithmus-Idee – Ein Greedy-Algorithmus** (Stand: 20.11.2015)

Der Greedy-Gedanke besagt, dass man Schritt für Schritt versucht einen gültigen Zustand zu erhalten, um so zu gewährleisten, dass auch nach dem letzten Schritt, das Ergebnis korrekt sein muss. Übertragen auf den Verteilungsalgorithmus bedeutet das also, dass man in jeder Iteration für jeden Studenten versucht, immer eine konfliktfreie Praktikumsgruppe zu finden. Konfliktfrei bedeutet, dass ein Zuweisen einer weiteren Praktikumsgruppe nicht dazu führen kann, dass ein Konflikt zu bisher zugewiesenen Praktikumsgruppen auftritt. Dies simuliert eigentlich nur den realen Anmeldeprozess und wie sich ein Student entscheiden würde. Die erste Version dieses Algorithmus behandelt derzeit nur Einzelanmeldungen. Die Teamanmeldungen, so der Gedanke, müssten später eh in der Überprüfung von gültigen, konfliktfreien Praktikumsgruppen auf Einzelanmeldungen runtergebrochen werden. Da es mit dieser Lösung nicht möglich ist, zu gewährleisten, dass jedem Studenten auch eine konfliktfreie Praktikumsgruppe zugewiesen werden kann, werden diese Studenten in dieser Algorithmus-Version als „nicht-zuteilbar“ markiert und nicht weiter behandelt. Die Verantwortung, was mit diesen Studenten geschehen soll, wird der Hochschule überlassen. Dies ist ebenfalls eine Simulation der realen Situation.

**Pseudocode**

*Für jede Veranstaltung (v)*

* *Für jeden angemeldeten Studenten (s) // Random, um*

*// keinem Vorteile zu schaffen, nur weil sich*

*// ein Student zuerst angemeldet hat*

* + *Praktikumsgruppe p = findeKonfliktfreiePraktikumsGruppe(s, v)*
  + *Wenn eine Gruppe gefunden wurde:*
    - *p.registriereStudent(s)*
  + *Ansonsten markiere den Studenten als „nicht zuteilbar“*

*/\*\**

*\* Prüft, ob es eine konfliktfreie Praktikumsgruppe für den Stunden für eine Veranstaltung gibt und liefert*

*\* diese dann gegebenenfalls zurück.*

*\*/*

*findeKonfliktfreiePraktikumsGruppe(Student s, Veranstaltung v){*

***for****(Gruppe g : v.getGruppen()){*

***if****(g.plätzeFrei()* ***&&*** *keinKonfikt(g,s){ // wenn die Gruppe noch freie Plätze hat und*

***return*** *g; // die Gruppe keinen Konflikt mit dem*

*} // Studenten besitzt. D.h. eine Zuweisung*

*} // dieser Gruppe führt nicht dazu, dass der*

***return*** *null; // Student mehrere Gruppen zur gleichen*

*} // Zeit besuchen müsste*

­

**Simulationsdaten**

Es haben sich im Verlauf der Anmeldungsphase 9 Studenten für folgende beiden Veranstaltungen angemeldet:

**Veranstaltungen**

* Logik und Berechenbarkeit
  + Montags von 08:15 Uhr bis 11:30 Uhr
    - Gruppe:1, Termine: (43,46,49,54), Anzahl Plätze: 3
    - Gruppe:2, Termine: (44,47,50,55), Anzahl Plätze: 3
    - Gruppe:3, Termine: (45,48,51,56), Anzahl Plätze: 3
* Rechner und Maschinennahe Programmierung
  + Montags von 08:15 Uhr bis 11:30 Uhr
  + Drei Praktikumsgruppen mit je 3 Plätzen
    - Gruppe:1, Termine: (45,48,51,56), Anzahl Plätze: 3
    - Gruppe:2, Termine: (43,46,49,54), Anzahl Plätze: 3
    - Gruppe:3, Termine: (44,47,50,55), Anzahl Plätze: 3

**Ablauf des Algorithmus**

1.) Student:2 Gruppen:[Logik und Berechenbarkeit Gr.: 1]

2.) Student:3 Gruppen:[Logik und Berechenbarkeit Gr.: 1]

3.) Student:9 Gruppen:[Logik und Berechenbarkeit Gr.: 1]

4.) Student:1 Gruppen:[Logik und Berechenbarkeit Gr.: 2]

5.) Student:7 Gruppen:[Logik und Berechenbarkeit Gr.: 2]

6.) Student:4 Gruppen:[Logik und Berechenbarkeit Gr.: 2]

7.) Student:6 Gruppen:[Logik und Berechenbarkeit Gr.: 3]

8.) Student:8 Gruppen:[Logik und Berechenbarkeit Gr.: 3]

9.) Student:5 Gruppen:[Logik und Berechenbarkeit Gr.: 3]

10.) Student:2 Gruppen:[Logik und Berechenbarkeit Gr.: 1, Rechner und Maschinennahe Programmierung Gr.: 1]

11.) Student:5 Gruppen:[Logik und Berechenbarkeit Gr.: 3, Rechner und Maschinennahe Programmierung Gr.: 2]

12.) Student:8 Gruppen:[Logik und Berechenbarkeit Gr.: 3, Rechner und Maschinennahe Programmierung Gr.: 2]

13.) Student:3 Gruppen:[Logik und Berechenbarkeit Gr.: 1, Rechner und Maschinennahe Programmierung Gr.: 1]

14.) Student:4 Gruppen:[Logik und Berechenbarkeit Gr.: 2, Rechner und Maschinennahe Programmierung Gr.: 1]

15.) Student:7 Gruppen:[Logik und Berechenbarkeit Gr.: 2, Rechner und Maschinennahe Programmierung Gr.: 2]

16.) Für Student:1 und Kurs Rechner und Maschinennahe Programmierung konnte keine freie Gruppe mehr gefunden

werden.

17.) Student:9 Gruppen:[Logik und Berechenbarkeit Gr.: 1, Rechner und Maschinennahe Programmierung Gr.: 3]

18.) Student:6 Gruppen:[Logik und Berechenbarkeit Gr.: 3, Rechner und Maschinennahe Programmierung Gr.: 3]

Liste aller Studenten, die nicht verarbeitet werden konnten: [Student:1]

**Ergebnisse der einzelnen Praktikumsgruppen**

Logik und Berechenbarkeit Gr.: 1 Teilnehmerzahl: 3

Logik und Berechenbarkeit Gr.: 2 Teilnehmerzahl: 3

Logik und Berechenbarkeit Gr.: 3 Teilnehmerzahl: 3

Rechner und Maschinennahe Programmierung Gr.: 1 Teilnehmerzahl: 3

Rechner und Maschinennahe Programmierung Gr.: 2 Teilnehmerzahl: 3

Rechner und Maschinennahe Programmierung Gr.: 3 Teilnehmerzahl: 2

**Auswertung:**

*Student 1* hatte nur noch die Wahl für „Rechner und Maschinennahe Programmierung“ in Praktikumsgruppe 3 zu gehen, da alle anderen Praktikumsgruppen zu diesem Zeitpunkt bereits konfliktfrei belegt wurden. Da Praktikumsgruppe 3 zu einem Konflikt geführt hätte, konnte keine gültige Gruppe gefunden werden. (siehe Schritt 16.) Alle Studenten haben sich vernünftig verhalten und haben bis auf *Student 1* auch konfliktfreie Stundenpläne erhalten. Bei der Suche nach einer gültigen Praktikumsgruppe in „Rechner und Maschinennahe Programmierung“ für *Student 1* wären die einzigen möglichen Gruppen Praktikumsgruppe 1 oder 2 gewesen. Da diese allerdings schon voll belegt waren, konnte keine gültige Gruppe gefunden werden und der Student musste als „nicht zuteilbar“ markiert werden.

Dieses Ergebnis zeigt, dass obwohl sich der Algorithmus bei der Suche nach einer gültigen Praktikumsgruppe vernünftig verhalten hat, dennoch keine optimale Lösung gefunden werden konnte. Und das, obwohl es eine geben würde, nämlich:

Student 1 Gruppen [Logik und Berechenbarkeit Gr.: 1, Rechner und Maschinennahe Programmierung Gr.: 1]

Student 2 Gruppen [Logik und Berechenbarkeit Gr.: 1, Rechner und Maschinennahe Programmierung Gr.: 1]

Student 3 Gruppen [Logik und Berechenbarkeit Gr.: 1, Rechner und Maschinennahe Programmierung Gr.: 1]

Student 4 Gruppen [Logik und Berechenbarkeit Gr.: 2, Rechner und Maschinennahe Programmierung Gr.: 2]

Student 5 Gruppen [Logik und Berechenbarkeit Gr.: 2, Rechner und Maschinennahe Programmierung Gr.: 2]

Student 6 Gruppen [Logik und Berechenbarkeit Gr.: 2, Rechner und Maschinennahe Programmierung Gr.: 2]

Student 7 Gruppen [Logik und Berechenbarkeit Gr.: 3, Rechner und Maschinennahe Programmierung Gr.: 3]

Student 8 Gruppen [Logik und Berechenbarkeit Gr.: 3, Rechner und Maschinennahe Programmierung Gr.: 3]

Student 9 Gruppen [Logik und Berechenbarkeit Gr.: 3, Rechner und Maschinennahe Programmierung Gr.: 3]

**Was kann man jetzt tun?**

Man hat jetzt im Prinzip drei Möglichkeiten, um eine bessere Lösung zu erhalten. Die erste wäre, dass man alle Permutationen durchgeht (keine gute Idee). Die zweite wäre, dass man versucht *Student 1* mit einem möglichen Studenten aus Praktikumsgruppe 1 oder 2 zu tauschen. Der Aufwand hierfür wäre aber hoch, besonders wenn Teams dazu kommen. Die letzte Möglichkeit wäre, dass man versucht feste Abhängigkeiten von Gruppen zu implementieren in der Art: „Wenn ein Student in Logik und Berechenbarkeit in Gruppe 1 eingefügt wird, dann muss er bei Veranstaltungen aus demselben Semester, die zeitgleich stattfinden, möglichst auch in Gruppe 1.“ Auf diese Art und Weise funktioniert der Stundenplan an der HAW. Wenn sich ein Student konstant auf diese Weise anmelden würde, würde es, siehe oben, eine Lösung geben. Dafür müssten diese Abhängigkeiten vorher bekannt sein. (oder nicht? – gilt noch zu überprüfen)

**Heuristiken – Die Rettung? – Ein erster Gedankengang**

Um zu wissen, welchen Studenten wir für einen Tausch in Erwägung ziehen könnten, wäre es möglicherweise sinnvoll den Studenten Heuristiken zu geben und sie zu kategorisieren. Die Heuristiken wären insofern sinnvoll, da man sich somit Rechenleistung sparen könnte und die Suche somit wesentlich schneller laufen könnte. So wäre es zum Beispiel denkbar, dass jeder Student eine Wertung erhält, die sich aus der Anzahl an Veranstaltungen für die er sich angemeldet hat und bei denen es potenzielle Überschneidungen gibt und aus der Anzahl an unterschiedlichen Teams in denen er Mitglied ist zusammensetzt. Unterschiedliche Teams sind Teams, die nicht nur ausschließlich aus denselben Teammitgliedern bestehen. Bestehen die Teams aus den gleichen Mitgliedern, wird das Netz aus Verknüpfungen dadurch nicht vergrößert und muss somit nicht aufaddiert werden. Außerdem werden hierfür auch nur Teams betrachtet, die zu eben solchen Veranstaltungen gehören die Überschneidungen haben. Veranstaltungen bei denen es nicht zu Konflikten kommen kann sind zunächst uninteressant.

Sobald für einen *Studenten x* für eine *Veranstaltung v* keine Gruppe gefunden werden konnte, muss eine Wertung vorgenommen werden, anhand der man dann schauen kann, ob ein Tausch mit einem anderen *Studenten y* ungefährlich wäre. Aus diesem Grund wird dieser Wert von jetzt an DANGER genannt. Ebenso ermöglicht es dieser Gedanke einem ganzen *Team t* eine Wertung zu geben, wenn man mal ein ganzes Team verschieben möchte oder muss. Der DANGER-Wert von *t* ergibt sich dann aus der Addition der DANGER-Werte der einzelnen Teammitglieder. Um das ein wenig anschaulicher zu machen, sind hier ein paar Beispiele:

1. Ist *Student y1* nur für eine einzige Veranstaltung angemeldet, so hat er den DANGER-Wert 0. Er kann beliebig in eine andere Praktikumsgruppe verschoben werden, ohne dass Gefahr besteht, dass es zu einem Konflikt für ihn kommt. – **Erste Wahl**
2. *Student y2 ist* nur für eine einzige Veranstaltung angemeldet und Mitglied eines Teams. Er hat also einen DANGER-Wert von 1 und ist immer noch ein guter Kandidat, um in eine andere Gruppe verschoben zu werden – **Zweite Wahl**
3. *Student y3* hat sich nur für „Logik und Berechenbarkeit“ und für „Rechner und Maschinennahe Programmierung“ angemeldet und ist keinem Team beigetreten. Da die Praktikumsgruppen dieser beiden Veranstaltungen potenzielle Überschneidungen haben, würde der DANGER-Wert des Studenten also 1 betragen. Da er in keinem Team Mitglied ist, kann das ignoriert werden. – **Zweite Wahl**
4. *Student y4* hat sich für zwei Veranstaltungen angemeldet, dessen Praktikumsgruppen potenzielle Überschneidungen haben und ist für jede der beiden Veranstaltungen Mitglied in einem anderen Team. *Student y* erhält also einen DANGER-Wert von 3. (1 potenzieller Konflikt zwischen zwei Veranstaltungen und Mitglied in 2 verschiedenen Teams) In diesem Fall ist es schon sehr viel aufwendiger *Student y* in eine andere Gruppe zu verschieben, da es sehr viel schwerer wird, die Teamerhaltung zu gewährleisten, ohne dass es für das Team zu einem Konflikt kommt. Dieser Student ist schon keine gute Wahl mehr – **Dritte Wahl**
5. *Team t* soll verschoben werden. *t* besteht aus *Student x* mit einem DANGER-Wert von 2 und *Student y* mit einem DANGER-Wert von 1 und hat damit also einen DANGER-Wert von 3. Für ein Team ist das dennoch eine gute Wertung. Ich hab mich zu diesem Zeitpunkt allerdings noch für keine Klassenkategorien für Teams entschieden. Dieses Team wäre aber vermutlich eine gute Wahl, um es zu verschieben, da die Wertung relativ gering ist. Teams könnten je nach Größe und Vernetzung der einzelnen Mitglieder sogar einen DANGER-Wert weit über 10 erhalten.

Für die erste Algorithmus-Idee (die einfache Greedy-Variante), die aktuell nur Einzelanmeldungen betrachtet, wäre es durchaus denkbar, dass die Einführung dieser Heuristiken bereits ein sehr gutes Ergebnis liefert. Für Teams kann dies allerdings nicht geliefert werden, da dort je nach der Anzahl der Vernetzungen der einzelnen Teammitglieder, eine optimale Lösung ausgeschlossen werden muss und das Team möglicherweise aufgelöst werden muss, um für alle Studenten einen konfliktfreien Stundenplan zu schaffen.

**Wie kann man jetzt den ersten Algorithmus dahingehend erweitern, dass die eben entworfenen Ideen umgesetzt werden können?**

Zunächst muss man sich überlegen, wann genau die DANGER-Werte berechnet werden sollten. Auch hier gäbe es zunächst zwei Möglichkeiten:

1. Dies kann entweder in dem Moment geschehen, wenn ein Student getauscht werden müsste
   * Vorteil: es müssten kurzfristig wesentlich weniger Berechnungen durchgeführt werden, nämlich nur die DANGER-Werte der Studenten, die in möglichen Praktikumsgruppen Plätze belegen.
2. oder einmal für alle Kursanmeldungen eines Kurses
   * Vorteil: man könnte Studenten mit hohen DANGER-Werten versuchen zuerst zuzuordnen, damit es im besten Fall gar nicht erst zu einem Tausch kommen müsste.
   * Nachteil: Sehr hoher Rechenaufwand

**Aktuelle Ideen (01.12.2015):**

* Die DANGER Werte taugen nichts mehr, sobald alle Studenten/Teams denselben Wert haben.
  + Führt dazu, dass getauscht werden muss wie blöde.
* Es gilt den Grad der Vernetzung während der Anmeldephase herauszufinden, um dem entgegen zu wirken.
  + Mathematisch auf Grund der DANGER-Werte?
* Einzelanmeldungen in Zukunft als Teams der Größe 1 behandeln?
* Viele Fehler entstehen bereits durch den User während der Anmeldephase und müssen auch bereits dort behandelt werden:
  + Student meldet sich für zu viele Veranstaltungen an einem Termin an, so dass eine konfliktfreie Belegung gar nicht erst möglich wird. Beispiel: Student x meldet sich für 4 Veranstaltungen an, die zeitgleich stattfinden, aber jeweils nur drei Praktikumsgruppen besitzen. -> **Anmeldung nicht möglich! Warnung an den User!**
  + Zwei Vorlesungen für die sich ein Student anmeldet finden zur selben Zeit statt. -> **nur eine Warnung an den User.**
  + Eine Vorlesung und ein Praktikum überschneiden sich. **-> Warnung an den User!**
  + **Idee zur Lösung:**
    - **Kreisüberprüfung durchführen!**

**Aktuelle Ideen (04.12.2015):**

* Wir sind am Verzweifeln
* Es bleiben nach stundenlangen Überlegungen und Untersuchungen eigentlich nur noch die folgenden Möglichkeiten:
  1. Wir können einen konfliktfreien Stundenplan für alle Studenten ermöglichen (sofern sie sich nicht für mehr Veranstaltungen die potentielle Konflikte haben anmelden, als Praktikumsgruppen bereitgestellt werden). Die Teamerhaltung kann dann aber nicht mehr eingehalten werden.
  2. Wir können versuchen so viele Teams wie möglich zu erhalten, so dass sie auch noch konfliktfreie Stundenpläne besitzen. Die restlichen Teams können aber nicht weiterverarbeitet werden. Dies ist nicht wirklich möglich, da es möglicherweise nicht mal möglich ist die Teams aufzulösen und die Studenten in andere für sie konfliktfreie Gruppen einzufügen, da alle anderen Gruppen bereits voll sind. Die einzige Möglichkeit wäre in diesem Fall das Auflösen anderer Teams, was im Schlimmsten Fall dazu führen könnte, dass die Teamerhaltung so minimal ist, dass es der Aufwand nicht wert wäre. Die Verantwortung, was mit denen geschieht, die nicht zugewiesen werden konnten, ohne dass es zu einem Konflikt für sie kommt, muss somit der Hochschule übergeben werden.
  3. Wir können die Teamerhaltung gewährleisten, wenn wir Konflikte ignorieren.
  4. Man könnte die Bedingungen der Teamerstellung einschränken:
     + Nur ein Team pro Veranstaltung und allen Veranstaltungen die mit denen in Konflikt stehen. Das heißt: Wenn sich Student a zusammen mit Student b für DB anmelden möchte und dafür ein Team bildet, dann darf sich Student a auch nur alleine oder zusammen mit Student b als Team bei RMP anmelden. (Sofern RMP mit DB natürlich im Konflikt steht)
     + Auf diese Weise wäre es denkbar, dass die Teamerhaltung gewährleistet werden kann und die Stundenpläne konfliktfrei bleiben.

**TODO 15.12.2015**

* Architektur ein wenig überarbeiten!
  + Den AllocationPlan anpassen
    - Einzelanmeldungen rausnehmen! -> Es gibt nur noch mindestens Teams der Größe 1
    - Die AbstractAllocationPlanAlgorithm Klasse rausnehmen/verändern und die Funktionalitäten in die AllocationPlan-Klasse stecken
    - Den Code überarbeiten
    - Den Tausch auf Teambasis implementieren, ansonsten wie in SecondExtendedAllocationPlanAlgorithm weitermachen
    - GUI bauen! :D
      * Button für das Ausführen des Algorithmus
      * Übersichtliche Ansicht der Ergebnisse
        + Allgemeine Statistik und für jedes einzelne Fach

Wie viele Studenten haben sich angemeldet?

Wie viele Teams haben sich gebildet?

Wie viele gleiche Teams haben sich für Kurse angemeldet?

Wie viele Teams wurden erhalten?

Wie viele Studenten haben einen konfliktfreien Stundenplan?

Welche Studenten konnten gar nicht eingefügt werden?

Laufzeit des Algorithmus

* + - * Verschiedene Werkzeuge um die Ergebnisse zu bearbeiten:
        + Manuelles Verschieben von Studenten, die nicht bearbeitet werden konnten in andere Praktikumsgruppen und somit auch
        + ein manuelles Erweitern von Praktikumsgruppengrößen
      * Speichern der Ergebnisse als CSV-Datei

**Stand 05.01.2016 (4:30Uhr morgens)**

Der Algorithmus:

Letztendlich hat sich eine einfache Greedy-Variante durchgesetzt. Die Ergebnisse sind wie erwartet nicht perfekt, aber brauchbar. Im Durchschnitt erhält der Algorithmus ca. 70% der Teams und um schafft es ca. 90%-100% der Registrierungen zu verwerten.

*\* For every Course c  
\* For every registrated Team team  
\* Group g = findGroup(team,c);  
\* if (g not null)*

*\* registerTeam(g,team);  
\* else  
\* destroy Team and mark the TeamRegistrations as unfinished  
\**

*\* For every unfinished TeamRegistration teamReg  
\* Group g = findGroup(teamReg,c); //  
\* if (g not null)  
\* registerTeamRegistration(g,teamReg);  
\* else  
\* mark teamReg as unMatchable*

Die Idee:

Der Algorithmus versucht so viele Teams wie möglich zu erhalten und fügt im Greedy-Verfahren ein Team nacheinander konfliktfrei in eine Praktikumsgruppe. Kommt es zu einem Konflikt und es gibt keine mögliche Gruppe, wird das Team stumpf zerstört und zunächst zurückgelegt. Die Teams werden anhand ihrer Größe abgearbeitet. Die größten Teams kommen zuerst und danach die kleineren Teams. (Kleinere Teams kann man leichter „in Lücken stopfen“.) Zu Beginn jeder Iteration über die Kurse wird die Teamliste geshuffelt, um kein Team bevorzugt zu behandeln. Auf diese Weise entsteht ein Nicht-Terminismus der sich auf sonderbare Weise positiv auswirkt auf die Ergebnisse, da er einem ermöglicht verschiedene Lösungen zu generieren, die in häufigen Fällen sogar besser sein können, als ohne das Shuffeln.

Sind die Teams abgearbeitet, wird das ganze nochmal mit den übergebliebenen TeamRegistrations gemacht. Kommt es hier wieder zum Konflikt, gibt es anscheinend keine Lösung und die TeamRegistration wird als notMatchable markiert. Die weitere Behandlung muss im Nachhinein erfolgen.

Um eine bessere Lösung zu bieten kam jetzt die Idee auf, durch Interaktion mit einem Menschen, das Ergebnis noch zu verarbeiten und somit zu verbessern. Aus diesem Grund ist eine erste GUI entstanden, die neben einer ausführlichen Darstellung der Ergebnisse auch ermöglicht, das Ergebnis zu verbessern. In der aktuellen Version sind die Funktionen noch nicht sehr umfangreich. Es ist nur möglich, die Konflikte entweder automatisch vom System beheben zu lassen (hierbei werden allerdings die Gruppengrößen verändert) oder aber manuell einen Studenten einer Gruppe zuzuteilen.

Dieser Ansatz scheint Potential zu haben und sollte weiter verfolgt werden.