

„Optimaler Sitzplan im Klassenraum auf Basis evolutionärer Algorithmen“

Studienarbeit II

für die Prüfung zum
Bachelor of Science

der Angewandten Informatik
an der Dualen Hochschule Baden-Württemberg - Karlsruhe

von
Michael Sprauer

September 2013

Bearbeitungszeitraum	6. Semester
Matrikelnummer, Kurs	5147809, TAI10B2
Ausbildungsfirma	Thales Defence & Security Systems GmbH, Pforzheim
Betreuer	Prof. Dr. H. Braun

I Erklärung

gemäß § 5 (2) der „Studien- und Prüfungsordnung DHBW Technik“ vom 18. Mai 2009:

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Karlsruhe, Mai 2013

MICHAEL SPRAUER

II Abstrakt

In der Schule müssen Lehrer regelmäßig einen Sitzplan erstellen, um in der Klasse eine optimale Lernatmosphäre zu gewährleisten. Dabei sind viele – zum Teil widersprüchliche – Anforderungen zu berücksichtigen. Beispielsweise müssen einige Schüler in der ersten Reihe sitzen, da sie eine Sehschwäche haben oder besonderer Betreuung durch den Lehrkörper bedürfen. Außerdem sollen nur die Schüler nebeneinander sitzen, die sich nicht vom Lernen abhalten. Besser sind Sitznachbarn, die das Lernen fördern. Damit insgesamt das Lernen in der Klasse optimal läuft, muss auch der Geräuschpegel niedrig sein. Wer sich also zu gut versteht, darf auch nicht nebeneinander sitzen. Diese Herausforderungen implizieren bereits, dass es keinen perfekten Sitzplan gibt. Diese Studienarbeit verwendet evolutionäre Algorithmen, um den optimalen Sitzplan mit den gegebenen Einschränkungen und Parametern zu finden.

III Abstract

Creating a seating plan is a conventional task for every teacher. A well-defined seating plan is mandatory for an effective learning atmosphere in every classroom. The requirements are partly conflicting. For example some of the students have to sit in the front row, because they have debility of sight or need special mentoring from the teacher. Other students can't be placed next to each other, because they distract each other. In the best case the neighbours support each other. To facilitate the best possible learning environment, the noise level should be very low. Chatting students shouldn't sit side by side. All these challenges already imply that there is no such thing like a perfect seating plan. This thesis uses genetic algorithms to find the best possible solution considering all the given constraints and parameters.

IV Inhaltsverzeichnis

1	Einleitung und Motivation.....	1
1.1	Zielsetzung	1
1.2	Wirtschaftliche Aspekte	1
1.3	Gliederung der Arbeit	2
2	Grundlagen: Evolutionäre Algorithmen	3
2.1	Repräsentation.....	4
2.2	Zielfunktion / Fitnessfunktion	5
2.3	Greedy.....	6
2.4	Mutationsoperatoren	7
2.4.1	Rekombination.....	7
2.4.2	Mutation	8
2.5	Selektion.....	9
2.6	Strategie	10
2.7	Haltebedingung.....	11
3	Implementierung.....	12
3.1	Entwicklungsumgebung	12
3.2	Software-Architektur	14
3.3	Codeausschnitte.....	15
3.3.1	Bewertungsfunktion	15
3.3.2	Selektion	17
3.4	Codequalität.....	18
3.5	Arbeitsaufwand.....	19
4	Bedienung.....	20
4.1	Einstellungen.....	20
4.2	Klassen erzeugen, laden und speichern.....	21
4.3	Konfigurationen erzeugen, laden und speichern.....	22
4.3.1	Generation	23
4.3.2	Mutation	25
4.3.3	Gewichtung.....	26
4.4	Klassenraum.....	28
4.4.1	Relationen der Schüler.....	29
4.4.2	Relation zur Tafel	30
4.5	Fitness-Funktion	30
5	Evaluation	32
5.1	Mutation durch Swap.....	32
5.2	Mutation durch Inversion	34
5.3	Auswahl der Mutationsoperatoren	35
5.4	Strategien	39
5.5	Selektion.....	40

6	Zusammenfassung und Ausblick	42
6.1	Verbesserungsmöglichkeiten	42
6.2	Fazit	43

1 Einleitung und Motivation

In der Schule müssen Lehrer regelmäßig einen Sitzplan erstellen, um in der Klasse eine optimale Lernatmosphäre zu gewährleisten. Dabei sind viele – zum Teil widersprüchliche – Anforderungen zu berücksichtigen. Beispielsweise müssen einige Schüler in der ersten Reihe sitzen, da sie eine Sehschwäche haben oder besonderer Betreuung durch den Lehrkörper bedürfen. Außerdem sollen nur die Schüler nebeneinander sitzen, die sich nicht vom Lernen abhalten. Noch besser sind Sitznachbarn, die das Lernen fördern. Damit insgesamt das Lernen in der Klasse optimal läuft, muss auch der Geräuschpegel niedrig sein. Wer sich also zu gut versteht, darf auch nicht nebeneinander sitzen. Diese Herausforderungen implizieren bereits, dass es keinen perfekten Sitzplan gibt. Einen brauchbaren Plan zu erstellen, in dem möglichst viele Aspekte berücksichtigt werden, erfordert einen erheblichen Zeitaufwand.

1.1 Zielsetzung

Ziel der Arbeit ist es, einen optimalen Sitzplan zu finden, der die gegebenen Rahmenbedingungen und Parameter berücksichtigt. Dabei werden die folgenden Kriterien berücksichtigt:

- Entfernung zur Tafel
- Beziehung zum direkten Nachbar
- Beziehung zum übernächsten Nachbar
- Beziehung zum Vorder-/Hintermann
- Beziehung zu den Schülern, die diagonal sitzen
- Schüler, die ihre Position nicht verändern dürfen

1.2 Wirtschaftliche Aspekte

Die Zeit zur Erstellung eines Sitzplans bedeutet für den Lehrer in der Regel einen Aufwand von mindestens einer Stunde. Je nach Verhalten der Klasse oder Eignung des Sitzplans ist dieser Vorgang mehrmals pro Schuljahr

notwendig. Diese Zeit könnte effektiver zur besseren Vorbereitung des Unterrichts verwendet werden.

1.3 Gliederung der Arbeit

Diese Arbeit ist grob in zwei Teile unterteilt. Das Kapitel 2 beschäftigt sich mit den theoretischen Grundlagen des evolutionären Algorithmus. Dann folgt in Kapitel 3 die Beschreibung der konkreten Realisierung der gestellten Aufgabe. In Kapitel 4 wird die Bedienung der Anwendung erläutert. Abschließend evaluiert Kapitel 5 die erzielten Ergebnisse und Kapitel 6 gibt einen Ausblick auf Untersuchungsmöglichkeiten um die Ergebnisse oder die Performance weiter zu verbessern.

2 Grundlagen: Evolutionäre Algorithmen

Dieses Kapitel soll die Grundlagen der Studienarbeit erläutern. Als Verfahren zur Lösung des gestellten Problems eignen sich die evolutionären Algorithmen besonders. Dieses Verfahren wurde zwischen 1970 und 1980 von verschiedenen Wissenschaftlern unabhängig voneinander als effiziente Weise der Optimierung von numerischen Problemen erkannt. In Deutschland wurde das Thema durch Schwefel [1] und Rechenberg [2] unter dem Namen „*Evolutionsstrategie*“ bekannt. Auf der anderen Seite des Atlantiks veröffentlichten Goldberg und Holland [3] ähnliche Gedanken unter dem Titel „*genetische Algorithmen*“. Insbesondere bei nicht linearen Optimierungsproblemen muss man abwägen zwischen Rechenaufwand und Nutzen.

Die evolutionären Algorithmen bilden die Evolutionstheorie Darwins nach. Dort unterscheidet man zwischen Genotyp und Phänotyp. Der Phänotyp entspricht dabei dem konkreten Exemplar einer Spezies. Der Genotyp ist die genetisch kodierte Information über alle Merkmale und Eigenschaften dieser Art. Der Genotyp mutiert durch äußere Einflüsse bzw. wird durch die Fortpflanzung rekombiniert. Der neu entstandene Genotyp bzw. der dazugehörige Phänotyp wird nun einem Fitnesstest unterzogen. Überlebt dieser Genotyp bzw. kann mehr Nachkommen erzeugen als andere Genotypen, spricht dies für seine Fitness. Damit werden nur die Vererbungslinien weiter verfolgt, die eine bessere Fitness haben als andere.

Diese Überlegung wird nun auf das gestellte Problem übertragen. Der Genotyp wird im Folgenden auch als Repräsentation bezeichnet und in Kapitel 2.1 genauer beschrieben. Der Fitnesstest wird durch die Fitnessfunktion bzw. Zielfunktion durchgeführt (Kapitel 2.2). Darauf folgt die Erläuterung der initialen Erstellung des ersten Genotyps durch eine sogenannte Greedy-Funktion in Kapitel 2.3. Das Kapitel 2.4 beschreiben die Mutation und Reproduktion. Se-

lektion wird im Kapitel 2.5 behandelt und im Kapitel 2.6 geht es um die Strategie zur Auswahl der Eltern. Abschließend beschreibt der letzte Teil 2.7 das Ende dieses iterativen Prozesses. Alle Abschnitte in diesem Kapitel 2 betrachten die Evolutionären Algorithmen immer im Kontext der gestellten Aufgabe.

2.1 Repräsentation

Die Repräsentation bezeichnet die Kodierung eines konkreten Sitzplans. Das bedeutet, dass in dieser Kodierung alle Informationen über den Sitzplan enthalten sein müssen. Auf diese Repräsentation wird auch die Fitnessfunktion angewendet. Die folgenden Elemente müssen aus der Darstellung hervorgehen:

- Auf welche Position im Klassenraum bezieht sich die Angabe
- Welcher Schüler ist gemeint

Damit ergibt sich für einen rechteckigen Klassenraum mit $B * H$ Tischen die folgende Repräsentation – in diesem Fall eine Permutationscodierung:

$$\begin{aligned} \Pi = & \Pi_{1,1} \Pi_{1,2} \Pi_{1,3} \dots \Pi_{1,H} \\ & \Pi_{2,1} \\ & \dots \\ & \Pi_{B,1} \Pi_{B,2} \Pi_{B,3} \dots \Pi_{B,H} \end{aligned}$$

Listing 1: Permutationscodierung - $\Pi_{i,j}$ ist der Schüler an der Stelle (i,i)

In der Permutationscodierung kommt jeder Schüler genau einmal vor. Bei unterbesetzten Klassenräumen kann es jedoch vorkommen, dass Plätze zwischen den Schülern unbesetzt bleiben können. Diese Eigenschaft kann man sich unter Umständen zu Nutze machen, indem Puffer zwischen Schülern ein-

gebaut werden, um die Fitness zu verbessern. In der Regel werden die freien Plätze durch den Algorithmus jedoch eher in der hinteren Reihe angeordnet.

2.2 Zielfunktion / Fitnessfunktion

Die Zielfunktion beschreibt die Fitness einer gegebenen Repräsentation, gibt also eine Bewertung über die Qualität ab. Man kann sich die Funktion als eine mehrdimensionale Trajektorie vorstellen, die einen Lösungsraum aufspannt. Die verschiedenen Lösungen werden durch die Dimensionen definiert und der Wert an dieser Stelle entspricht der Fitness. Die Zielfunktion besteht aus zwei Teilen, die jeweils gewichtet und summiert werden:

- Abstand zur Tafel
- Abstand zu den Nachbarn

In der folgenden Abbildung 1 sind die Gewichtungsfaktoren jeweils unterschiedlich eingefärbt. Für die Realität ist es – wie in der Bewertung – nicht relevant, ob ein Schüler links oder rechts neben einem anderen Schüler sitzt. Ebenso werden die Schüler vorn und hinter dem Schüler, diagonal zu dem Schüler und zwei Plätze neben dem aktuellen Schüler gleich gewichtet.



Abbildung 1: Gewichtung der Umgebung eines Schülers - Jeder Farbe kann ein Gewichtungsfaktor zugewiesen werden. Schüler, die direkt nebeneinander sitzen, egal ob rechts oder links, haben den gleichen Gewichtungsfaktor.

Die übrigen Schüler werden zur Bestimmung der Fitness und damit für die Qualität des Sitzplans nicht berücksichtigt.

Der optimale Chromosomensatz kann entweder eine möglichst kleine oder eine möglichst große Fitness haben, abhängig davon wie man das Problem formuliert bzw. die Fitnessfunktion definiert. Hier ist die Fitnessfunktion so definiert, dass der beste Sitzplan den nominell höchsten Wert hat. Die Beziehungen werden addiert und mit dem entsprechenden Faktor aus Abbildung 1 multipliziert. Damit ergibt sich eine umso bessere Fitness je mehr positive Beziehungen sich nebeneinander befinden.

2.3 Greedy

Um eine Mutation zu erzeugen oder zwei Gene zu rekombinieren, muss zunächst ein initialer Genstring erzeugt werden. Diese Aufgabe wird – in

Anlehnung an das Knapsackproblem - auch *Greedy* genannt. Bei dem Knapsack müssen viele Elemente unterschiedlicher Größe in einen begrenzten Speicher (dem knappen Sack) so einsortiert werden, dass möglichst viele hinein passen. Dabei fängt man üblicherweise mit dem größten Element an. Wenn man den Gedankengang auf das vorliegende Problem überträgt, dann erhält man gleich eine recht gute Lösung. Der Algorithmus dazu besteht aus drei Schritten:

1. Alle Schüler, die einen festen Sitzplatz zugewiesen bekommen haben, werden im Raum platziert.
2. Die übrigen Schüler werden nach ihrer Tafelpräferenz sortiert.
3. Der Reihe nach werden die Schüler auf die übrigen Plätze verteilt beginnend vorne links mit dem Schüler mit der höchsten Tafelpräferenz. Bei Schülern mit der gleichen Tafelpräferenz wird zufällig ein Schüler ausgewählt.

Für diesen Genstring wird nun die Fitness bestimmt.

2.4 Mutationsoperatoren

Um nun von diesem initialen Gen zu einem besseren Gen zu kommen, im Sinne der Fitnessfunktion (siehe 2.2) muss das Gen verändert werden – also Mutation – oder es müssen zwei Elemente ein Neues bilden – also Rekombination. Diese beiden Verfahren sollen nun kurz erläutert werden.

2.4.1 Rekombination

Bei der Rekombination kommen zwei mehr oder weniger zufällig ausgewählte Gene zusammen und bilden gemeinsam ein neues Gen. Dabei gibt es verschiedene Varianten, nach denen der neue Genstring gebildet werden kann:

1. Uniform Crossover [4]
2. One Point Crossover

3. Multipoint Crossover [5]
4. Partially Maped Crossover (PMX)
5. Order Crossover
6. Cycle Crossover [6]

Je nach ausgewählter Variante weist das neue Gen mehr oder weniger Ähnlichkeiten zu seinen beiden Eltern auf, kann also hinsichtlich der Bewertung besser oder schlechter sein. Eine neue Bewertung durch die Fitnessfunktion muss also erfolgen. Diese Rekombinationsoperatoren sind sehr anfällig dafür, nicht überlebensfähige Nachkommen zu bilden und wurden daher in der aktuellen Version nicht implementiert. Die Nachkommen werden ausschließlich über Mutation erzeugt.

2.4.2 Mutation

Mutation ist eine Operation die auf ein einzelnes Gen wirkt. Auch hier können verschiedene Varianten eingesetzt werden.

1. Swap
2. Inversion
3. Insert
4. Splice [7]

An dieser Stelle werden nur die ersten beiden Varianten erläutert, die in diesem Projekt auch eingesetzt wurden.

Der **Swap**-Operator wählt zufällig zwei Schüler aus und vertauscht ihre Sitzposition. Die neue Repräsentation unterscheidet sich nicht so stark von der vorherigen. Diese Eigenschaft ist wichtig, um sich der optimalen Sitzordnung langsam zu nähern und nicht immer wieder am Ziel vorbei zu schießen.

Der **Invert**-Operator vertauscht die Reihenfolge einer ganzen Sitzreihe. Dabei verändert sich in der Mitte der Reihe nur wenig, während die äußeren

Sitzplätze sehr starken Änderungen unterworfen sind. Das hat zur Folge, dass die neue Bewertung sehr stark von der vorherigen abweichen kann. Für eine effiziente Annäherung an die optimale Sitzordnung sollten nicht zu viele Nachkommen durch Invertierung erzeugt werden, da hier sehr starke Schwankungen zu erwarten sind.

2.5 Selektion

Die so erzeugten Nachkommen werden in den Genpool der aktuellen Generation aufgenommen. Dort werden die Gene miteinander verglichen. Um die Qualität der Nachkommen weiter zu verbessern, dürfen nicht alle Elemente im Genpool auch Nachkommen bzw. gleich viele Nachkommen erzeugen. Damit werden Fehlentwicklungen verhindert. Dieser Auswahlprozess wird Selektion genannt. Zur Selektion stehen ebenfalls verschiedene Möglichkeiten zur Verfügung, die nun kurz vorgestellt werden.

1. Nur die n Besten (Best-N)
2. Nach Rang (Rank-Based)
3. Wettbewerb (Tournament)

Die erste Methode (**Best-N**) ist auch gleichzeitig die einfachste. Hier werden alle Nachkommen nach absteigender Fitness sortiert und die N Besten daraus zur Mutation ausgewählt. N wird durch eine Konfigurationsoption in der Oberfläche festgelegt (siehe Kapitel 4.3.1).

Auch bei der **Rank-Based**-Selection werden die Nachkommen sortiert und die N besten ausgewählt. Im Unterschied zur Best-n-Selection darf aber nicht jeder gleich viele Nachkommen erzeugen. Die Erzeugung der Nachkommen erfolgt absteigend. Der Beste darf die meisten Nachkommen erzeugen. Der Zweitbeste einen weniger, und so weiter. Je nachdem wie viele Nachkommen erzeugt werden sollen, kann es so passieren, dass die letzten Eltern der N Besten keine Mutationen bilden dürfen.

Die Methode **Tournament** wählt aus allen Elementen zufällig eine bestimmte Anzahl aus. Davon darf sich dann das beste Element Reproduzieren. Damit kann man recht zuverlässig lokale Optima verlassen, allerdings kommt man nicht so leicht in ein lokales oder globales Optimum. Die Fitness ist am Ende also nicht so gut, wie sie sein könnte.

2.6 Strategie

Nun stellt sich noch die Frage welche Chromosomen für eine Mutation herangezogen werden können. Ausgehend von dem ersten Element, das durch den Greedy-Algorithmus (Kapitel 2.3) erzeugt wurde, werden anschließend die vorgegebene Anzahl Nachkommen mutiert (Kapitel 2.4) und daraus die Eltern selektiert (Kapitel 2.5). Für den Genpool der nächsten Generation können nun zwei Mengen gebildet werden:

1. Nachkommen mit (+) ihrer Elterngeneration
2. Nachkommen ohne (,) ihre Elterngeneration

Die Nachkommen werden in der Regel mit μ gekennzeichnet und die Elterngeneration mit λ . Üblicherweise ist die Schreibweise Folgende:

1. $(\mu + \lambda)$
2. (μ, λ)

Für die nächste Mutationsphase steht jetzt eine neue Menge zur Verfügung, die ebenfalls die Eltern enthalten kann. Diese beiden Optionen wurden auch im Programm implementiert.

Wenn man die Eltern nicht mit in die neue Generation aufnimmt, dann besteht die Möglichkeit, dass keiner der mutierten Nachkommen besser ist als ihr Elternchromosom und damit die neue Generation insgesamt schlechter abschneidet als die vorherige. Wenn man die Eltern allerdings mit aufnimmt, dann bleibt der Algorithmus unter Umständen in einem lokalen Optimum stehen und kann sich nicht weiter verbessern.

2.7 Haltebedingung




Zum Schluss muss auch noch eine Abbruchbedingung formuliert werden. Während es bei einer möglichst kleinen Fitness sogar die Möglichkeit gibt, den perfekten Wert „0“ zu erreichen, ist diese Grenze bei steigender Fitness nicht gegeben. Das führt zu einer potentiell endlosen Schleife, in der immer weiter versucht wird ein noch besseres Ergebnis zu erzielen. Man muss also nach einer definierten Anzahl von Generationen abbrechen.

3 Implementierung

Zur Implementierung wurde Java verwendet, da es heute eine sehr große Verbreitung hat, durch viele Werkzeuge unterstützt wird und sich gut zur Umsetzung dieser Aufgabe eignet. Die Entwicklungsumgebung dafür wird im Kapitel 3.1 beschrieben. Anschließend folgt die Beschreibung der Softwarearchitektur (Kapitel 3.2) und es werden einige interessante Abschnitte aus dem Quellcode herausgenommen und besprochen (Kapitel 3.3).

3.1 Entwicklungsumgebung

Als IDE (Integrated Development Environment) kommt Eclipse zum Einsatz. Eclipse lässt sich durch Plugins erweitern und an die individuellen Bedürfnisse anpassen. Für dieses Projekt kamen die folgenden Plugins zum Einsatz:

EclEmma Java Code Coverage 	Zeigt die Testabdeckung direkt in Eclipse an. So lässt sich einfach erkennen, welche Stellen von Tests abgedeckt sind und wo der Code evtl. noch nicht die erforderliche Codequalität aufweist.
Maven Integration for Eclipse 	Integration von dem Application-Lifecycle-Management-Tool Maven. Mit Maven können Abhängigkeiten verwaltet werden und der Buildprozess kann parametrisiert und automatisiert werden.
Sonar 	Sonar zeigt verschiedene Metriken zur Codequalität direkt in der IDE an. Dadurch werden Probleme an ihrer Quelle sichtbar und können umgehend behoben werden.



EGit 	Mit EGit kann das Versionierungswerkzeug git direkt in Eclipse eingebunden werden. Geänderte Elemente werden angezeigt und lassen sich committen oder rückgängig machen. Außerdem kann man sich die Historie einer Datei anzeigen lassen uvm.
WindowBuilder 	Der WindowBuilder erleichtert die Erstellung von grafischen Oberflächen mit Swing und SWT. In dem WYSIWYG-Editor können die grafischen Elemente direkt auf einer Form platziert und parametrisiert werden.

Tabelle 1: Verwendete Eclipse Plugins

Neben diesen Plugins wurden auch zwei Bibliotheken verwendet:

- **JUnit** als Unit-Testing-Framework
- **JFreeChart** zur Darstellung des Diagramms

Zur Nachverfolgung der Änderungen im Quelltext und an der Thesis wird git eingesetzt. Der folgende Screenshot (Abbildung 2) zeigt den Verlauf der Entwicklung. Jede Änderung wird in einem sogenannten Commit zusammengefasst und mit einer Bezeichnung versehen. So lässt sich der Projektfortschritt leicht erkennen und fehlerhafte Änderungen können rückgängig gemacht werden. Außerdem dient diese Methode als Backup für korrupte Dateien oder fehlerhafte Änderungen.

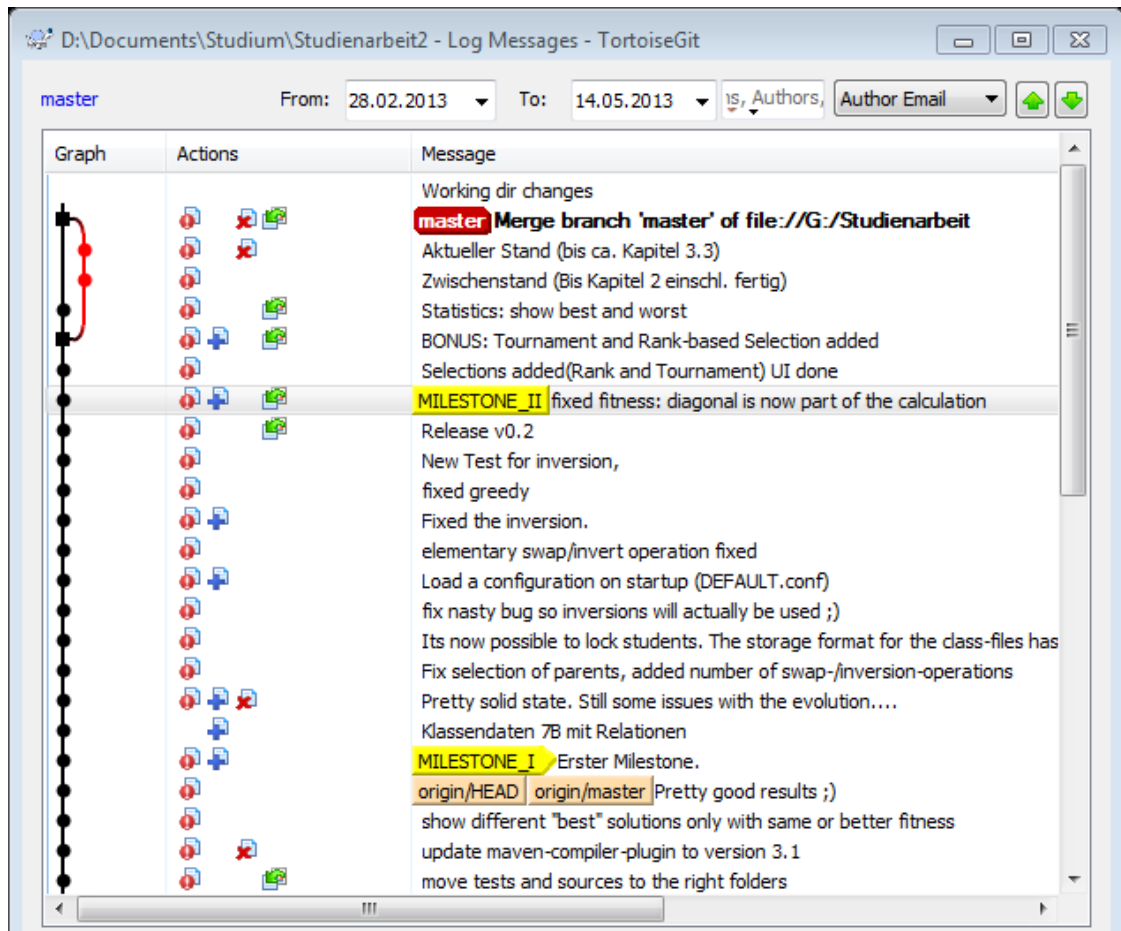


Abbildung 2: Git Versionslog - Die Änderungen werden zusammengefasst und mit einer kurzen Beschreibung in die Historie eingefügt.

Die gelben Markierungen sind Tags. Dadurch werden Milestones in dem Projekt markiert, um auf ältere Auslieferungsversionen zurück zu kommen und dort ggf. Fehler zu reproduzieren.

3.2 Software-Architektur

Als grobe, grundlegende Architektur kommt für dieses Projekt das MVC-Muster zum Einsatz.

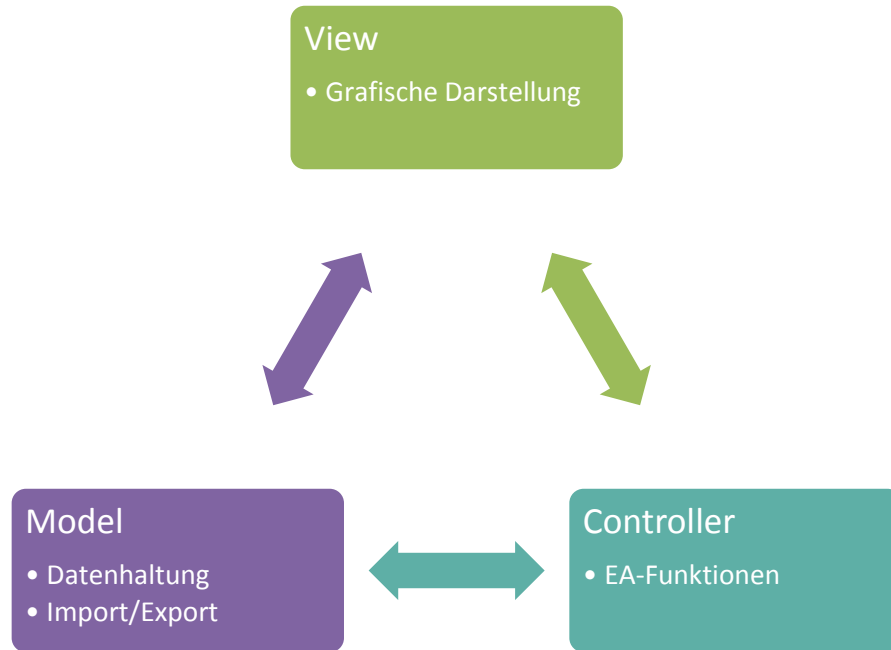


Abbildung 3: MVC-Muster - Dieses Muster ist eine übliche Architektur für Applikationen. Datenhaltung, Geschäftslogik und Darstellung sind in eigenen Modulen.

Das Modell besteht aus einer In-Memory-Datenbank, die eine Anzahl von Studenten und die Abmessungen des Klassenraums, sowie weitere Metadaten speichert.

3.3 Codeausschnitte

In diesem Kapitel sollen einzelne Codesequenzen besprochen werden, die beispielhaft für die Lösung der Aufgabe stehen. Den Anfang macht die Bewertungsfunktion (Kapitel 3.3.1), dann folgt ein Ausschnitt aus der Selektion (Kapitel 3.3.2).

3.3.1 Bewertungsfunktion

Die Fitnessfunktion ist das zentrale Element in der Fachlogik. Hier werden die Relationen mit der Gewichtungsfunktion verrechnet und zu einem Wert verdichtet.

```
private double fitnessFor(Student student, Point position) {
    double fitness = 0;
    fitness += gene.getConfig().getWeightingPriority() *
        getPriorityValueOf(student, position.y);
    fitness += getRelationsFor(student, position);
    return fitness;
}

private double getRelationsFor(Student student, Point position) {
    double fitness = 0;
    final Configuration config = gene.getConfig();
    fitness += config.getWeightingRight() *
        student.relationTo(gene.studentAt(rightOf(position)));
    fitness += config.getWeightingRight2() *
        student.relationTo(gene.studentAt(rightOf(rightOf(position))));
    fitness += config.getWeightingBottom() *
        student.relationTo(gene.studentAt(under(position)));
    fitness += config.getWeightingDiagonal() *
        student.relationTo(gene.studentAt(under(rightOf(position))));
    fitness += config.getWeightingDiagonal() *
        student.relationTo(gene.studentAt(under(leftOf(position))));
    return fitness;
}
```

Listing 2: Berechnung der Fitness

Die beiden Methoden finden sich in `class` `OperationFitness` im Modul `net.sprauer.Sitzplaner.EA.operations`. Für jeden Schüler wird die Methode `fitnessFor` aufgerufen und zur Gesamtfitness addiert. In der Variable `config` wird die gesamte Konfiguration für den EA hinterlegt. Nun wird jeweils der Gewichtungsfaktor für die Position mit der Relation des Schülers multipliziert. Der Schüler hat direkten Zugriff auf die Datenbank und liefert die gesuchte Relation bzw. gibt 0 zurück, wenn keine Relation vorhanden ist. Die Hilfsfunktionen `rightOf` und `under` liefern jeweils die Position mit einem veränderten X bzw. Y-Wert. In dem Gen sind die Positionen aller Schüler codiert. So kann man einfach mit `gene.studentAt` nach einer Position fragen und bekommt den entsprechenden Schüler bzw. einen `null`-Wert.

Damit hat man in einer Iteration über alle Schüler die Gesamtfitness des Chromosomensatzes berechnet.

3.3.2 Selektion

Ein anderer interessanter Teil des Projekts ist die Selektion der Population aus den mutierten Nachkommen.

```
private List<Chromosome> selecteTheBestAndKillTheRest() {
    List<Chromosome> parents = new ArrayList<Chromosome>();
    if (configuration.getStrategy() == Strategy.Tournament) {
        tournamentSelection(parents);
    } else {
        int index = 0;
        double fitness = Double.MAX_VALUE;
        do {
            Chromosome chromosome = population.get(index);
            double delta = Math.abs(fitness - chromosome.getFitness());
            if (delta > 0.5) {
                parents.add(chromosome);
                fitness = chromosome.getFitness();
            }
            index++;
        } while (parents.size() < Math.min(population.size(),
            configuration.getParents()) && index < population.size());
    }

    clear();
    if (configuration.isStrategiePlus()) {
        add(parents);
    }
}
```

Listing 3: Selektion der neuen Elterngeneration aus der vorhandenen Population

Das Listing zeigt, wie die Elterngeneration in der variable `parents` zusammengestellt wird. Wenn es sich um eine Tournament-Selektion handelt, werden die Eltern in einer separaten Funktion `tournamentSelektion` ausgewählt. Sonst werden solange Elemente aus der sortierten Population in die Elternliste eingefügt, bis die eingestellte Anzahl an Elternelementen vorhanden ist oder alle Elemente aus der Population eingefügt wurden.

Wenn man verhindert, dass gleiche Elemente als Eltern in Frage kommen, dann erhält man bessere Ergebnisse, da ähnliche oder gleiche Eltern auch zu ähnlichen oder gleichen Nachfahren führen würden. Jedes Element mit jedem anderen zu vergleichen wäre allerdings viel zu langsam. Daher

werden die Fitnesswerte miteinander verglichen und nur Elemente mit einer Abweichung größer als 0.5 zur Reproduktion zugelassen.

3.4 Codequalität

Wie bereits im Kapitel 3.1 erwähnt, wird in diesem Projekt JUnit eingesetzt. Das Schreiben von Tests dient dazu die Codequalität des Programmcodes zu verbessern. In der folgenden Abbildung 4 sind die einzelnen Module des Systems aufgelistet und nach ihrer Abdeckung sortiert. Das bedeutet: Je mehr ein Modul durch Tests abgesichert ist, desto höher ist die Abdeckung und desto weiter oben erscheint es in der Liste.

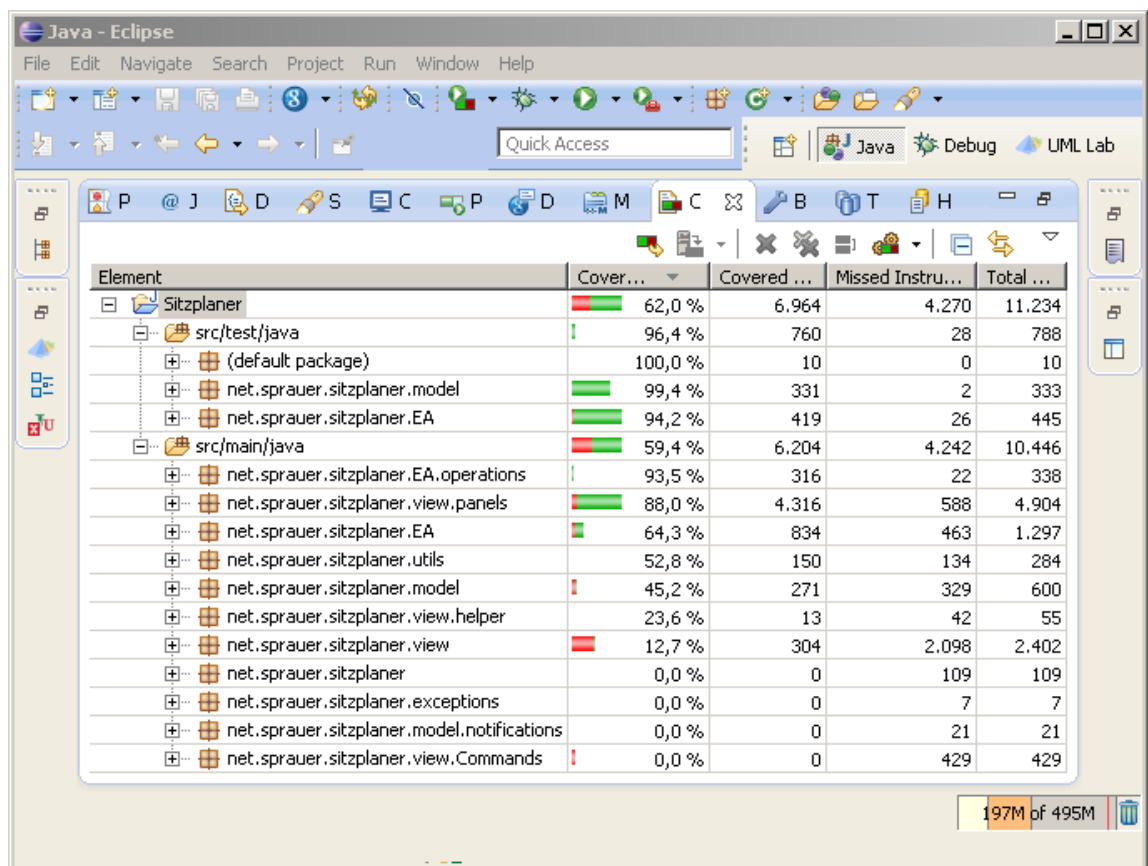


Abbildung 4: Abdeckung des Codes durch Unit-Tests - Der Balken zeigt die absteigende Abdeckung je Modul in Prozent an. GUI Module haben typischerweise einen sehr niedrigen Grad der Codeabdeckung.

Insgesamt erreicht das Projekt eine Abdeckung von 62 Prozent. Für ein sehr GUI-lastiges Projekt ist das ein recht guter Wert. Es ist deutlich erkennbar, dass die View-Module die geringste Abdeckung haben. Das ist dadurch erklärbar, dass Oberflächen nur schwer automatisiert getestet werden können. Das Ergebnis einer Berechnung kann direkt mit dem Soll-Wert verglichen werden und daher sehr einfach verifiziert werden. Bei einer Oberfläche können Anzeigeelemente nicht so einfach verifiziert werden. Das Diagramm beispielsweise, wird in ca. 50 Zeilen parametrisiert und bekommt die Werte für die Kurven zugewiesen. Das Ergebnis dieser Operation lässt sich aber nicht kontrollieren, ohne einen Test für die externe Bibliothek zu schreiben.

Die höchste Abdeckung haben die wichtigsten Bereiche: die EA Module, die Hilfsklassen (utils) und das Datenmodell (model).

3.5 Arbeitsaufwand

Was ebenfalls in dieser Grafik (Abbildung 4) sichtbar wird, ist die Verteilung der Komplexität bzw. des Aufwands. Der fachliche Anteil (das EA-Modul) fällt mit ca. 1500 Instruktionen recht moderat aus. Davon sind über 1100 Instruktionen über Tests abgesichert. Der Anteil für die GUI-Logik ist mit knapp 7500 Instruktionen das größte Modul. Ähnlich verteilt sich auch der geschätzte Arbeitsaufwand:

- GUI-Logik: ca. 70% Arbeitsaufwand
- Fachlogik: ca. 15% Aufwand
- Rest: Datenhaltung und Verwaltung, Import/Export

4 Bedienung

Das Programm ist grob in drei Bereiche gegliedert, die in der folgenden Abbildung 5 farbig hervorgehoben sind.

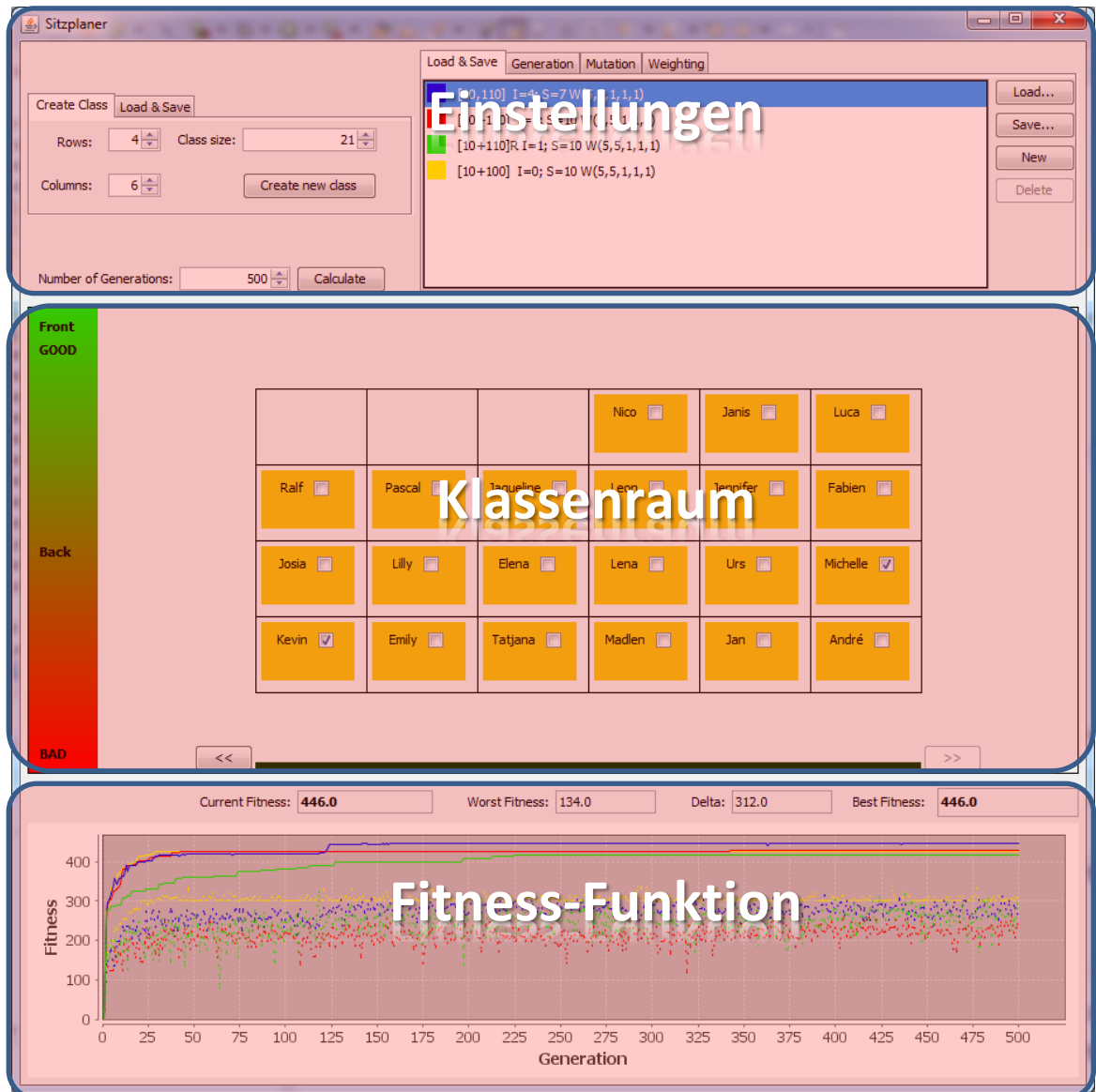


Abbildung 5: Die drei Bereiche des Sitzplaners

4.1 Einstellungen

Die Einstellungen zur Bedienung des Programms können in diesem ersten Abschnitt gesetzt werden.

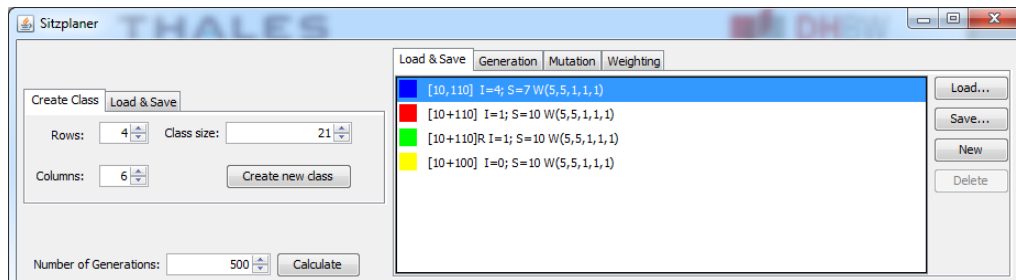
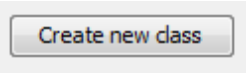
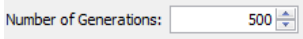
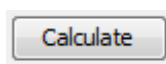


Abbildung 6: Einstellungen des Programms - Links der Bereich zum Erzeugen der Klasse und rechts die Parametrisierung des Evolutionären Algorithmus.

4.2 Klassen erzeugen, laden und speichern

In der folgenden Tabelle 2 werden die Einstellungsmöglichkeiten und Funktionen zum Erzeugen einer Klasse und zum Berechnen des Sitzplanes erläutert. Diese befinden sich in Abbildung 6 auf der linken Seite.

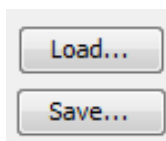
<p>Create Class</p> 	<p>Eine neue Klasse erzeugen. Die Abmessungen des Klassenraums kann in Anzahl der Zeilen (Rows) und Anzahl der Spalten (Columns) angegeben werden. Die Klassengröße – also die Anzahl der Schüler in der Klasse – wird in Class Size eingestellt. Mit Klick auf „Create new class“ wird die Klasse im Raum verteilt. Das Ergebnis ist im Klassenraum sichtbar.</p>
<p>Number of Generations</p> 	<p>Anzahl der Generationen, die bei Klick auf „Calculate“ erzeugt werden. Während der Berechnung wird über diesem Button ein Fortschrittsbalken angezeigt, der die Anzahl der berechneten Ergebnisse darstellt. Außerdem verändert sich der Text von „Calculate“ zu „Cancel“. Damit kann die laufende Berechnung abgebrochen werden.</p>
<p>Calculate</p>	<p>Erzeugt die notwendige Anzahl an Generationen und Chromosomen und errechnet die Fitness-</p>



Werte für jeden Chromosomensatz. Das beste Ergebnis wird im Klassenraum angezeigt.

Tabelle 2: Einstellungsmöglichkeiten Erzeugen und Berechnen - zur Erzeugung von Klassen, Einstellung der Anzahl der Generationen und Berechnung des Sitzplans


Load & Save



Über den Button „**Load**“ kann eine gespeicherte Klasse in das Programm geladen werden. „**Save**“ speichert eine Klasse. In der Datei werden nur die Beziehungen zueinander und nicht die Position der Schüler gespeichert. Ein neues Berechnen und Verändern ab dem gespeicherten Stand ist möglich.

Tabelle 3: Einstellungsmöglichkeiten Laden und Speichern

4.3 Konfigurationen erzeugen, laden und speichern

Die Liste auf der rechten Seite in Abbildung 6 zeigt alle geladenen Konfigurationen. Zwischen diesen Konfigurationen kann per Mausklick hin- und her geschaltet werden. Wird eine Konfiguration selektiert, so werden ihre Fitnesswerte über dem Diagramm dargestellt, der zugehörige Chromosomensatz wird in dem Klassenraum dargestellt und die Parameter werden in den Reitern „**Generation**“, „**Mutation**“ und „**Weighting**“ übernommen. In der Liste sind die wichtigsten Einstellungen in Textform kodiert. Diese Einstellung  [80+880] I=1; S=10 W(5,5,1,1,1) gibt an, dass aus einer Population von 880 die besten 80 als Eltern ausgewählt werden und aus jedem Elternchromosom dann ein Nachkomme durch Inversion und 10 Nachkommen durch Swap erstellt werden. Außerdem ist die Priorität mit 5 angegeben, die Gewichtung für den direkten Nachbarn beträgt 5 und der Reihe nach wird der übernächste, diagonale und davor Sitzende Nachbar auf eins gesetzt.

Die Einstellungen zum Laden, Speichern und Erzeugen von Konfigurationen finden sich in Abbildung 6 auf der rechten Seite und werden in der folgenden Tabelle 4 beschrieben.

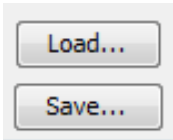

<p>Load & Save</p> 	<p>Über die Buttons „Load“ und „Save“ können vordefinierte Parameter des Evolutionären Algorithmus geladen und gespeichert werden. Die Liste links von diesen Buttons zeigt alle geladenen Einstellungen.</p>
<p>New & Delete</p> 	<p>Durch einen Klick auf „New“ wird die aktuell selektierte Konfiguration dupliziert und erscheint als neuer Eintrag mit neuer Farbe in der Liste. Für jede Einstellung werden zwei Kurven im Statistik-Panel angezeigt. Die Farben werden automatisch der Reihenfolge nach zugewiesen. Der Button „Delete“ löscht die selektierte Konfiguration. Es muss jedoch immer mindestens eine Konfiguration geladen sein.</p>

Tabelle 4: Bearbeiten der Konfigurationen-Liste - Alle Einstellungen, die die Berechnung des Sitzplans beeinflussen (wie z.B. die Gewichtungsfaktoren und die EA-Parameter) können in eine Datei gespeichert werden.

4.3.1 Generation

Der Evolutionäre Algorithmus wird hauptsächlich durch die Definition der Generation und der Mutation parametrisiert. Die Generation legt fest aus welchen Elementen die aktuelle Population besteht.

Abbildung 7: Einstellungen zur Bildung der Population

Die Abbildung 7 zeigt die möglichen Parameter und die zugehörige Tabelle 5 erläutert die einzelnen Optionen.

<p>Strategy</p>	<p>Bei der μ, λ-Strategie kommen nur die Nachfahren der vorhergehenden Population in Betracht. Mit dieser Strategie kann man lokale Optima besser wieder verlassen, die Fitness kann sich dadurch aber zwischendurch verschlechtern. Wählt man dagegen die $\mu + \lambda$-Strategie werden auch die Eltern der vorherigen Population mit einbezogen. Alle nachfolgenden Generationen können nie schlechter als das Elternelement werden. Damit verbessert man die Fitness kontinuierlich auf ein (lokales) Optimum hin. In Kapitel 2.6 findet sich die Theorie dazu.</p>
<p>Parents</p>	<p>Dieser Wert bestimmt den μ-Parameter – also die Anzahl der Chromosomen, die Nachkommen erzeugen dürfen. Je nach eingestellter Selektionsmethode werden beispielsweise nur die besten 4 der vorherigen Generation zur Reproduktion bzw. Mutation zugelassen.</p>
<p>Normal</p>	<p>Jedes Element der Population darf gleich viele</p>

<input checked="" type="radio"/> Normal	Nachkommen erzeugen.
<input checked="" type="radio"/> Rank	Das beste Element der Elterngeneration darf am meisten Nachkommen erzeugen. Der zweitbeste einen weniger, und so weiter. Damit ergibt sich eine linear absteigende Funktion. Es kann vorkommen, dass die letzten Eltern keine Nachkommen erzeugen dürfen (siehe Kapitel 2.5).
<input checked="" type="radio"/> Tournament 8	Bei dieser Methode wird das eine beste Chromosom aus 8 zufällig selektierten Chromosomen als Elternchromosom ausgewählt. Eine nähere Beschreibung hierzu findet sich in Kapitel 2.5.

Tabelle 5: Einstellungen zur Bildung der Population

4.3.2 Mutation

In dem Reiter „Mutation“ können alle Einstellungen zur Mutation der Nachkommen vorgenommen werden. Wenn eine neue Generation entsteht, wird jedes Elternelement geklont und dann mit den hier vorgegebenen Einstellungen mutiert. Die Abbildung 8 zeigt einen kurzen Text für die einstellbaren Parameter an, die in der Tabelle 6 weiter erläutert werden.

From 4 parents

create 7 descendants per parent using **swap** with 2 swap operations,

create 4 descendants per parent using **inversion** with 1 inversion operations.

Resulting in a total of 28 children using **swap**

and 16 children using **inversion**

The new generation has a population of 44

Abbildung 8: Einstellungen zur Mutation der Chromosomen

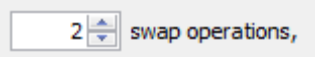

Swap	Für jedes Elternchromosom wird die hier angegebene Anzahl an Nachkommen durch Vertauschen von zwei Schülern erzeugt.
Swap operations 	Wenn ein Nachkomme durch Swap erzeugt werden soll, so werden für diesen Chromosomensatz die hier angegebene Anzahl an Swapvorgängen durchgeführt. Welcher Wert hier zu guten Ergebnissen führt, wird in Kapitel 5.1 behandelt.
Inversion	Jedes Elternelement wird sooft geklont, wie es hier angegeben ist. Dann werden die geklonten Nachkommen mit einer Inversion mutiert. Bei der Inversion werden eine oder mehrere zufällig selektierte Sitzreihen umgedreht. Der Schüler am linken Rand sitzt nun am rechten Rand, der zweite von links auf dem zweiten Platz von rechts, und so weiter.
Inversion operations 	Wie viele Inversionsoperationen je Nachkommen durchgeführt werden sollen, wird mit diesem Parameter angegeben. Eine Evaluation für die optimale Anzahl von Inversionen findet sich in Kapitel 5.2.

Tabelle 6: Einstellungen zur Mutation der Chromosomen

4.3.3 Gewichtung

Die Gewichtungsparemeter legen fest, welchen Einfluss die Sitzposition und die Umgebung eines Schülers auf die Fitness haben. Wie in Abbildung 9 zu sehen ist, können 5 verschiedene Gewichtungsfaktoren vergeben werden.

The screenshot shows a seating arrangement interface. At the top, there are three columns of numbers: '1', '1', and '1'. Below these, there is a grid of student positions. The central student is labeled 'Student' and has a weight of 5. Other students have weights of 1. Below the grid, there is a 'Priority' field set to 5 and a label 'distance to blackboard'.

Abbildung 9: Gewichtung der Relationen

Der Schüler, der sich in der Mitte befindet, wird durch seine Relationen zu anderen Schülern definiert. Je nach Erfahrung mit dem Sitzplan und abhängig von der Klasse können jedoch unterschiedliche Faktoren für die Relationen wichtig sein. Diese Faktoren können hier eingestellt werden. Wie bereits in 2.2 beschrieben, wird hier nicht unterschieden, ob ein Schüler rechts oder links neben einem anderen sitzt. Daher gibt es für die direkten Nachbarn nur einen Gewichtungsfaktor. Die Tabelle 7 bezeichnet die einzelnen Felder:

Student	Direkter Nachbar	Übernächster Nachbar
Vor oder hinter dem Schüler	Diagonal von dem Schüler	

Tafelpriorität

Tabelle 7: Einstellung der Gewichtung für die Relationen

Wenn die Gewichtung geändert wird, ergeben sich bei einer Neubewertung des Sitzplans auch höhere bzw. niedrigere Fitnesswerte. Daher sind, nach einer Änderung in diesem Teil des Programms, die Fitnesswerte nicht mehr direkt miteinander vergleichbar. Für einen Vergleich der Parameter, wie er in Kapitel 5 gemacht wurde, müssen diese Einstellungen bei allen Durchläu-

fen gleich sein. Details zur Implementierung der Fitnessfunktion, die dieses Verhalten begründen, finden sich in Kapitel 3.3.1.

4.4 Klassenraum

Der Klassenraum ist in einem Rechteck angeordnet. Der gelbe Bereich markiert einen Tisch. An jedem Tisch sitzt ein Schüler sitzen. Die Tafel ist unten im Bild zu sehen. Auf der linken Seite ist eine Legende eingezeichnet. Wenn man mit der Maus über die Schüler fährt, wird für jeden Schüler die Beziehung zu allen anderen Schülern angezeigt. Dabei entspricht grün einer guten Beziehung (diese Schüler sollten nebeneinander sitzen) und rot entspricht einer schlechten Beziehung (wenn diese Schüler nebeneinander sitzen, wird der Unterricht gestört oder die Beiden lenken sich gegenseitig ab).



Abbildung 10: Klassenraum mit Tafel, Schülern und einer Legende

Nach der Berechnung eines Sitzplans stehen unten neben der Tafel zwei Buttons zur Verfügung. Der linke Button ruft Sitzpläne aus der aktuellen Generation auf, die schlechter sind als der aktuelle. Wenn keine schlechteren Sitzpläne mehr verfügbar sind, wird der Button ausgegraut. Über den rechten Button kann dann zu besseren Sitzplänen geblättert werden. Wenn der beste Sitzplan angezeigt wird, wird der rechte Button inaktiv.

4.4.1 Relationen der Schüler

Alle Eigenschaften der Schüler lassen sich in der Klassenraumansicht anzeigen und festlegen. Dabei wird zunächst zwischen zwei verschiedenen Relationsarten unterschieden:

1. Beziehung zwischen zwei Schülern
2. Beziehung des Schülers zur Tafel

Wenn die Maus über einem Schüler schwebt, werden seine Beziehungen angezeigt. In der Abbildung 11 ist die Maus über Lilly und ihre Relationen werden sichtbar. Hier ist beispielsweise die Relation zur Tafel=2. Das bedeutet, Lilly sollte nach Möglichkeit näher an der Tafel sitzen. Dieser Wert lässt sich nicht nur an der Linie zur Tafel ablesen, sondern ist auch als weißer Balken in der Legende eingezeichnet.



Abbildung 11: Die Relationen einer Schülerin werden angezeigt.

Außerdem werden die Relationen zu verschiedenen Mitschülern wie zum Beispiel Jennifer(-5=Rot) und Ralf (4=Grün) angezeigt. Diese Relationen lassen sich auch verändern. Wenn Lilly beispielsweise nicht neben Ralf sitzen soll, dann kann Lilly angeklickt werden, mit gedrückter linker Maustaste auf Ralf gezogen werden und dort die linke Taste losgelassen werden. Über das

Mausrad kann nun die Relation verändert werden. Bei einer Drehung des Rads vom Benutzer weg, wird dieser Wert größer (Richtung grün). In die andere Richtung wird der Wert kleiner (Richtung rot). Der Wert wird an der Kante zwischen Lilly und Ralf angezeigt, ist in der Legende über den weißen Balken sichtbar und zusätzlich wird der Tisch von Jan in der entsprechenden Farbe eingefärbt. Ist der gewünschte Wert eingestellt, dann muss die Maus den Tisch von Jan verlassen. Damit wird die Bearbeitung abgeschlossen und der Wert übernommen. Die Beziehungen zwischen zwei Schülern ist immer dieselbe. Wenn Lilly einen Wert von 4 zu Ralf hat, dann hat Ralf auch einen Wert von 4 zu Lilly. Auf diese Weise lassen sich alle Beziehungen innerhalb der Klasse eintragen.

4.4.2 Relation zur Tafel

Wenn die Relation zur Tafel geändert werden soll, dann muss der entsprechende Schüler auf die Tafel gezogen werden. Auch hier ist nun die Bestimmung des Wertes über das Mausrad möglich. Beendet und übernommen wird die Eingabe durch einen Klick mit der linken Maustaste. Der Wertebereich für diese Relation liegt zwischen 0 und 5. Ein negativer Wert kann also nicht vergeben werden.

In Abbildung 11 ist die Relation von Lilli zur Tafel an der Kante zwischen ihrem Sitzplatz und der Tafel zu sehen. Für sie wurde ein Wert von 2 vergeben. Sie sollte also nicht ganz hinten sitzen. Aber wenn in der ersten Reihe kein Platz mehr ist, verliert die Gesamtbewertung (je nach Gewichtung) nicht so viel.

4.5 Fitness-Funktion

Der letzte Abschnitt der Oberfläche zeigt die Fitness-Funktion und damit die historische Entwicklung der Qualität der Sitzordnung.



Abbildung 12: Verlauf der Fitness und Fitness der aktuellen Repräsentation

In diesem Abschnitt ist auch der numerische Fitnesswert oben rechts eingeblendet. Wie die Abbildung 12 zeigt, steigt die Fitness in den ersten 25 Generationen stark an. Ab der 50. Generation verbessert sich die Fitness nur noch minimal. Aktuell wird ein Klassensatz angezeigt mit einer Fitness von 435. Der beste Wert ist mit 449 angegeben. In dieser Population ist der beste Sitzplan mit 449 und der schlechtesten mit 82 errechnet worden. Das macht ein Delta von 367.

5 Evaluation

In diesem Abschnitt werden die verschiedenen Parameter an einigen Beispielen untersucht. Damit sollte sich ein Parametersatz ergeben, der unter möglichst geringer Rechenzeit optimale Ergebnisse liefert. Als Datensatz für die Berechnung wurde ein 4 mal 6 großer Klassenraum erstellt und mit 21 Schülern besetzt. Damit bleiben 3 Plätze frei. Außerdem wurden einige Relationen gesetzt und zwei Schüler wurden in ihrer Platzwahl auf ihren aktuellen Platz beschränkt. Diese Datei ist auf dem Datenträger im Ordner „Klassendaten“ zu finden.

Zunächst sollen die beiden Mutationsoperatoren Swap und Inversion untersucht werden.

5.1 Mutation durch Swap

Für den Swap-Operator kann man einstellen, wie viele Swaps je Nachkommen durchgeführt werden sollen. Dieser Wert sollte nicht zu hoch sein, da sonst die Rechenzeit steigt und am Ende eventuell zwei bereits getauschte Schüler wieder zurück getauscht werden. Er darf aber auch nicht zu niedrig sein, sonst müssen zu viele Generationen erzeugt werden um vom Ausgangspunkt zum Zielpunkt zu kommen. Daher soll nun untersucht werden, welcher Wert optimal ist. Dazu werden verschiedene Werte eingestellt und die Ergebnisse verglichen. Die Einstellungen finden sich in Tabelle 8.

Konfiguration	Parameter	Blau	Rot	Grün	Gelb	Schwarz
Swap	Strategy	+	+	+	+	+
	Selection	Normal	Normal	Normal	Normal	Normal
	Parents	10	10	10	10	10
	Swaps	10	10	10	10	10
	Swap operations	1	2	3	4	5
	Inversion	0	0	0	0	0
	Inversion operations	0	0	0	0	0
	Descendants	110	110	110	110	110

Tabelle 8: Evaluation – Parameter: Wie viele Swaps je Nachkommen?

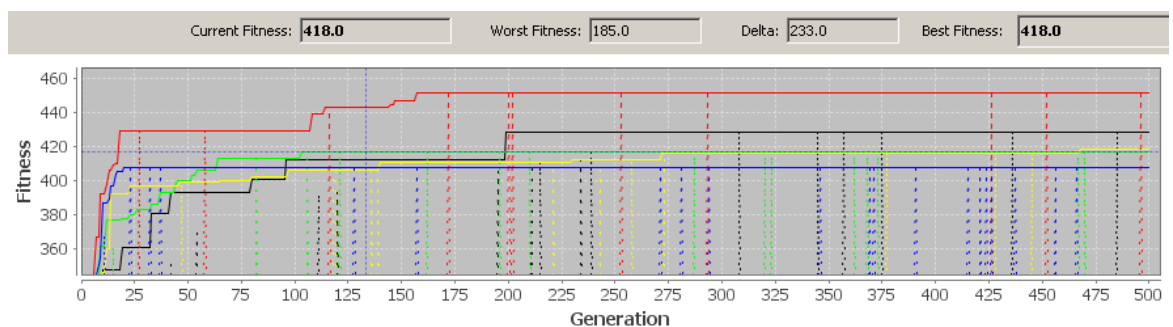


Abbildung 13: Wie viele Swaps je Nachkommen - Den besten Wert mit 451 erreicht die rote Kurve mit 2 Swaps je Nachkommen. Deutlich wird: Je mehr Swaps desto langsamer steigt die Fitness.

Die Antwort ist nicht so eindeutig, wie der Screenshot es vermuten lässt, denn es gibt auch Fälle in denen mit 4 oder 5 Swaps die besten Ergebnisse erzielt wurden. Deutlicher ist dagegen der Zusammenhang zwischen

Anfangssteigung und Swap-Anzahl. Je mehr Swaps eingestellt werden, desto langsamer steigt die Fitness.

Mit 2 bis 3 Swaps je Nachkommen kommt man schnell zu guten Ergebnissen. Wenn man ausreichend viele Generationen erzeugt (hier über 1.000), kann man auch mit 6 oder mehr Swaps den besten Sitzplan erzeugen, allerdings nicht so zuverlässig. Mit nur einem Swap je Nachkommen stagniert die Fitness bereits nach ca. 50 Generationen bei einer durchschnittlichen Fitness.

5.2 Mutation durch Inversion

Dieselbe Evaluation wird nun für die Inversion durchgeführt. Bei der Inversion werden eine oder mehrere zufällig ausgewählte Reihen invertiert.

Konfiguration	Parameter	Blau	Rot	Grün	Gelb	Schwarz
Inversion	Strategy	+	+	+	+	+
	Selection	Normal	Normal	Normal	Normal	Normal
	Parents	10	10	10	10	10
	Swaps	0	0	0	0	0
	Swap operations	0	0	0	0	0
	Inversion	10	10	10	10	10
	Inversion operations	1	2	3	4	5
	Descendants	110	110	110	110	110

Tabelle 9: Evaluation – Parameter: Wie viele Inversionen je Nachkommen?

Hier stellt sich bereits nach drei Generationen für alle Varianten ein konstanter Wert ein, der für eine gerade Anzahl an Inversionsvorgängen bei

239 liegt und für eine ungerade Anzahl bei 240. Diese Werte ergeben sich aus der sehr geringen Anzahl an Reihen in der Beispielklasse. Jede Inversion kann nur aus 4 verschiedenen Reihen wählen. Die Wahrscheinlichkeit bei geraden Inversionsvorgängen, die eben vorgenommene Inversion rückgängig zu machen ist sehr hoch.

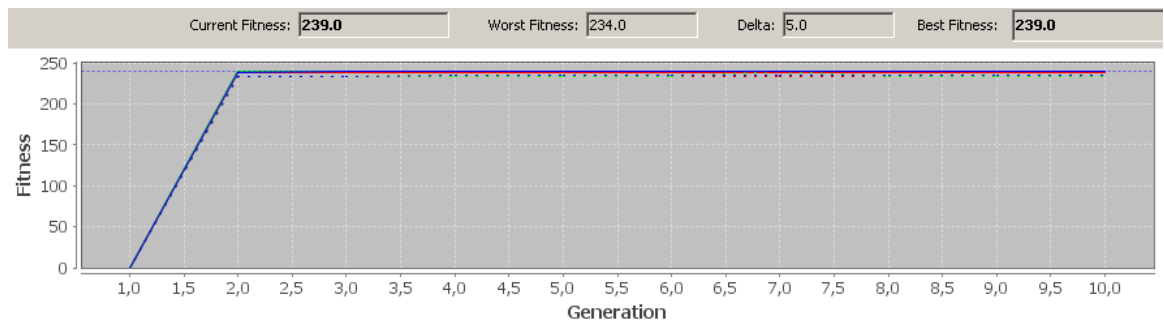


Abbildung 14: Wie viele Inversionen je Nachkommen?

Da die Ergebnisse für eine Inversion fast identisch mit den Ergebnissen für 2 oder mehr Inversionen sind, empfiehlt es sich diesen Wert aus Effizienzgründen auf 1 zu setzen.

5.3 Auswahl der Mutationsoperatoren

Nachdem gezeigt wurde, welche Parameter für die Mutation durch Swap bzw. durch Inversion optimal sind, soll nun untersucht werden, welcher der beiden Operatoren besser ist. Als Konfiguration werden die Werte wie in Tabelle 10 eingestellt.

Konfiguration	Parameter	Blau	Rot
11*Inversion (Blau) verglichen mit 10*Inversion + 1*Swap (Rot)	Strategy	+	+
	Selection	Normal	Normal
	Parents	10	10
	Swaps	0	1
	Swap operations	N/A	2
	Inversion	11	10
	Inversion operations	1	1
	Descendants	110	110

Tabelle 10: Evaluation – Parameter: Wird Swap benötigt?

Mit dieser Parametrierung kommen beide Konfigurationen auf 110 Nachkommen und hätten damit die gleiche Chance den besten Nachkommen zu bilden. In Abbildung 15 wird jedoch deutlich, dass bereits nach der 2. Generation in der blauen Konfiguration keine Verbesserung mehr eintritt. Die rote Konfiguration, bei der je Generation ein Nachkomme durch Swap erzeugt wurde, stagniert nach ca. 100 Generationen auf einer Fitness zwischen 400 bis 420. Ohne Swaps kommt die Fitness nicht über 240 hinaus.

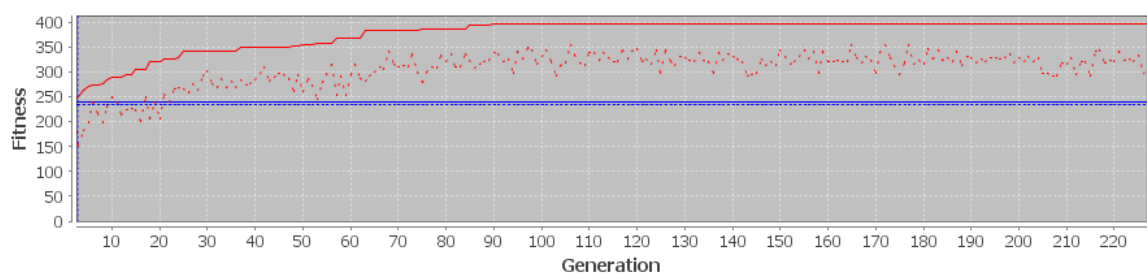


Abbildung 15: Wird Swap benötigt? - Die blaue Linie wird ausschließlich mit durch Inversion erzeugten Nachkommen gebildet. Der rote Verlauf entsteht durch eine Kombination von 10 Inversions-Nachkommen und einem Swap-Nachkommen.

Dieses Ergebnis überrascht nicht wirklich, denn allein durch invertieren einer Sitzreihe kommt nicht genug Variabilität in Sitzordnung. Insgesamt gibt es nur $4! = 24$ verschiedene Möglichkeiten die Reihen zu invertieren. Bei 10 Nachkommen ist die Wahrscheinlichkeit nach 2 Generationen bereits bei 83%, dass das optimale Ergebnis darunter ist.

Nun könnte man vermuten, die Inversion ist überflüssig. Diese These wird in der zweiten Evaluation untersucht. Dabei werden nur die Vorgaben für Swap und Inversion vertauscht. Die Parametrierung enthält Tabelle 11.

Konfiguration	Parameter	Blau	Rot
11*Swap (Blau) verglichen mit 10*Swap + 1*Inversion (Rot)	Strategy	+	+
	Selection	Normal	Normal
	Parents	10	10
	Swaps	11	1
	Swap operations	2	2
	Inversion	0	10
	Inversion operations	N/A	1
	Descendants	110	110

Tabelle 11: Evaluation – Parameter: Wird Inversion benötigt?

Hier fällt das Ergebnis deutlich knapper aus. Die Abbildung 16 zeigt die Verläufe der Fitness mit diesen Parametern.

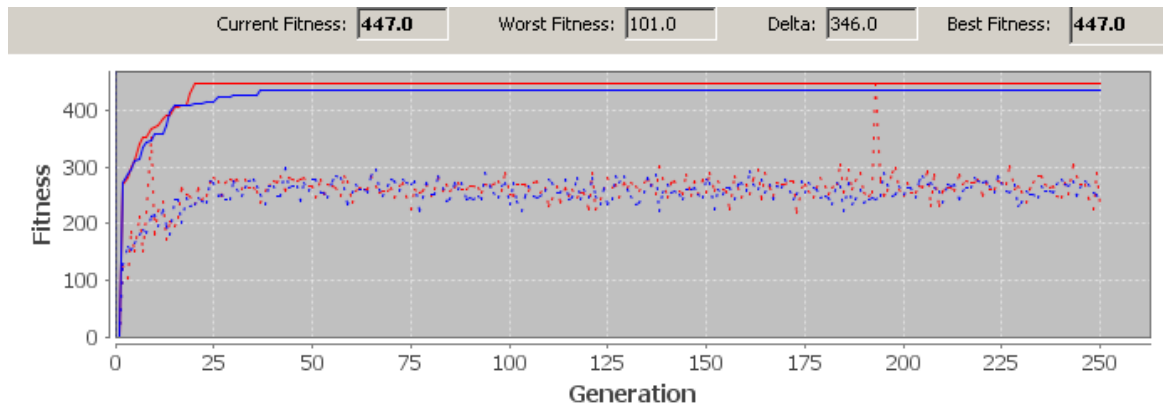


Abbildung 16: Wird Inversion benötigt? - Nach etwa 50 Generationen erreichen beide Konfigurationen ihr Maximum. Die Kombination Swap+Inversion liegt jedoch etwa 19 Punkte über der Variante ohne Inversion.

In mehreren Messungen wurden die Ergebnisse überprüft. Die folgende Tabelle 12 zeigt die Ergebnisse nach mehreren Durchläufen.

Durchlauf	1	2	3	4	5	6	7	8	9	10	Ø
Rot	431	442	441	432	428	413	419	449	426	449	433
Blau	421	437	425	417	440	429	419	428	406	414	423,6

Tabelle 12: Vergleich Swap mit und ohne Inversion nach 10 Durchläufen - Im Mittel sind die Swapmutationen in Kombination mit der Inversion ca. 5 Punkte besser. Es gibt jedoch auch Fälle in denen das Ergebnis ohne Inversion besser ist.

Offensichtlich kommt man nur mit der Swapmutation ohne die Verwendung der Inversion auf annähernd gleich gute Ergebnisse. Im Mittel liegt jedoch die Kombination Swap + Inversion 5 Punkte über den Ergebnissen ohne Inversion. Daraus kann man schließen, dass die optimale Konfiguration einen deutlich größeren Anteil an Swaps als Inversionen enthalten sollte.

5.4 Strategien

Zur Erstellung der Population stehen zwei Strategien zur Auswahl, die beide in Kapitel 2.6 vorgestellt wurden:

- $(\mu + \lambda)$ – Mit Eltern
- (μ, λ) – Ohne Eltern

Ein Vergleich soll nun zeigen, welche der beiden Strategien zu einem besseren Ergebnis führt. Dazu wurde der Algorithmus mit den Einstellungen aus Tabelle 11 parametrisiert.

Konfiguration	Parameter	Blau	Rot
+ - Strategie Verglichen mit , - Strategie	Strategy	+	,
	Selection	Normal	Normal
	Parents	10	10
	Swaps	10	10
	Swap operations	2	2
	Inversion	1	1
	Inversion operations	1	1
	Descendants	110	110

Tabelle 13: Evaluation – Parameter zum Vergleich zwischen +-Strategie und , - Strategie

Die beiden Strategien liegen nahezu gleich auf. Erst nach etwa 140 Generationen kann die , - Strategie in einigen Fällen noch eine Verbesserung erreichen, wie es in Abbildung 17 zu sehen ist.

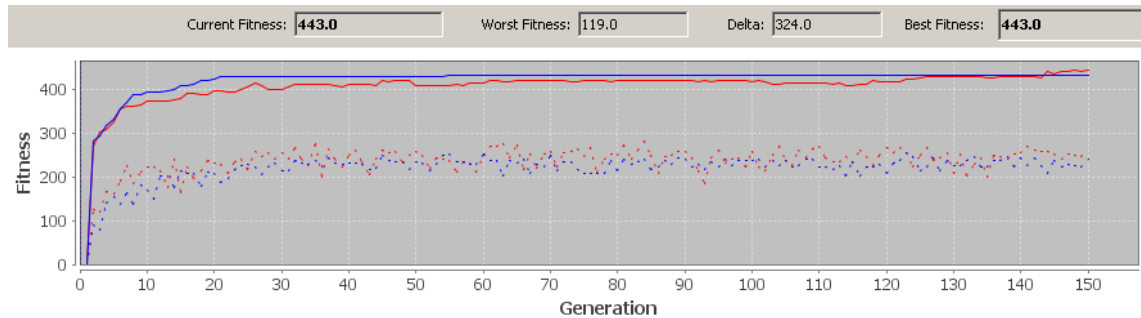


Abbildung 17: Evaluation - Vergleich zwischen +-Strategie und ,-Strategie. Erst nach ca. 140 Generation kann die ,-Strategie eine Verbesserung erreichen.

Die ,-Strategie erreicht in diesem Beispiel einen Höchstwert von 443, während die +-Strategie auf 434 bleibt. Eine eindeutige Empfehlung kann jedoch auch hier nicht gegeben werden, da es in vielen Fällen auch zu anderen Ergebnissen kommt. Generell lässt sich beobachten, was die Theorie vorausagt: Die +-Strategie verschlechtert sich nicht, und die ,-Strategie kann sich auch verschlechtern, erreicht dadurch aber hin und wieder zum Teil deutlich bessere Ergebnisse.

5.5 Selektion

Die Theorie zur den Selektionsmethoden wurde bereits in Kapitel 2.5 beschrieben. Hier soll es nun darum gehen, welche der drei Möglichkeiten die besten Ergebnisse liefert.

- Best-N
- Rank
- Tournament

Dazu wurde der EA, wie in Tabelle 14 dargestellt, parametrisiert.

Konfiguration	Parameter	Blau	Grün	Rot
Best-N-, Rank- und Tournament-Selektion im Vergleich	Strategy	+	+	+
	Selection	Normal	Rank	Tournament (Größe=8)
	Parents	10	10	10
	Swaps	10	10	10
	Swap operations	2	2	2
	Inversion	1	1	1
	Inversion operations	1	1	1
	Descendants	110	110	11

Tabelle 14: Evaluation - Parameter: Best-N- vs. Rank- vs. Tournament-Selektion



Abbildung 18: Evaluation - Vergleich Best-N-, Rank- und Tournament-Selektion

Das Ergebnis für Tournament fällt deutlich aus: Ist der Auswahlbereich zu klein, dann ist das Ergebnis, wie hier, sehr zufällig. Wenn der Auswahlbereich zu groß ist, verhält es sich wie eine normale „-Strategie mit einer sehr kleiner Population.

Die beiden anderen Varianten (Rank und Best-N) liegen etwa auf gleicher Höhe. Dass die normale Best-N-Selektion nach 1500 Generationen noch einen Sprung macht, muss nicht immer so sein.

6 Zusammenfassung und Ausblick

Dieses Kapitel soll aufzeigen welche Teilaspekte des Projekts im Sinne der Anforderungen, oder auch darüber hinaus, nicht optimal gelöst werden konnten.

Das Ziel des Projekts war es Evolutionäre Algorithmen zur Erstellung eines Sitzplans zu verwenden. In den Kapiteln 2 und 3 wurden die Grundlagen und die Implementierung erläutert. Kapitel 4 befasste sich mit der Beschreibung des fertigen Programms. Der Abschnitt 5 zeigte, dass die gestellte Aufgabe voll erfüllt wurde. Es ist mit Hilfe des Programms möglich einen Klassenraum in Rasterform zu modellieren und mit Schülern zu füllen. Außerdem können die geforderten Einschränkungen und Beziehungen modelliert werden und zum Schluss ein optimaler Sitzplan errechnet werden. Diese Daten lassen sich über eine File-Schnittstelle aus dem Programm exportieren und wieder in das Programm laden.

Zusätzlich dazu sind verschiedene Einstellungen möglich um die Erzeugung bzw. die Effizienz der Erzeugung zu beeinflussen und die Konfiguration zur Erzeugung zu speichern. Um einen schnellen Überblick über die Qualität des Sitzplans zu bekommen wurde ein Diagramm eingebaut, das den Verlauf der Qualität darstellt.

6.1 Verbesserungsmöglichkeiten

Zu den eingangs gestellten Anforderungen sind im Laufe des Projekts noch einige zusätzliche Ideen entstanden, die aus Zeitmangel jedoch nicht umgesetzt werden konnten.

Um die Berechnung auch von großen Klassen mit vielen Beziehungen möglichst schnell zu ermöglichen, könnte man die Fitness der einzelnen Chromosomen differentiell errechnen. Bei einer Änderung durch Swap wür-

den die Werte der beiden Schüler errechnet, von der Gesamtfitness abgezogen, an der neuen Position wieder berechnet und zur Gesamtfitness addiert.

Als weitere Optimierung wäre eine Anpassung des Algorithmus auf mehrere Kerne denkbar. Dazu muss der rechenaufwändige Teil so umformuliert werden, dass er parallel läuft und möglichst unabhängig voneinander ist. Diese Änderung ist relativ einfach umsetzbar, indem die Fitnessberechnung auf mehrere Kerne verteilt wird. Eine Übersicht über verschiedene Verfahren dazu gibt [8]. Diese Optimierung wurde in der aktuellen Version 0.3.5 noch eingebaut.

Die höchste Leistung, die mit heutigen handelsüblichen Rechnern möglich ist, stellt die Verwendung der Grafikkarte dar. Sie ist, im Gegensatz zu der CPU, genau auf die parallele Berechnung ausgelegt. Eine Untersuchung dazu wird gerade von Andreas Bartschat im Rahmen seiner Studienarbeit vorgenommen.

6.2 Fazit

Wie diese Studienarbeit zeigt, bieten die Evolutionären Algorithmen eine praktische und einfache Möglichkeit komplizierte Probleme zu lösen, für die es ohne weiteres keine Lösung gibt oder bei denen es nur unter sehr großem Aufwand möglich ist Lösungen zu errechnen. Die praktische Umsetzung dieses Problems hat mir sehr anschaulich die Herangehensweise zur Implementierung gezeigt. Am Ende war ich sehr überrascht, dass selbst für große Klassen mit vielen Beziehungen sehr schnell eine brauchbare Lösung errechnet werden konnte. Das Herumspielen mit den Parametern brachte mir ein tieferes Verständnis der EA und hat mir sehr viel Spaß gemacht.

I Glossar

EA: Evolutionäre Algorithmen

GUI: Graphical User Interface

IDE: Integrated Development Environment

MVC: Model View Controller

WYSIWYG: What You See Is What You Get

II Abbildungsverzeichnis

Abbildung 1: Gewichtung der Umgebung eines Schülers	6
Abbildung 2: Git Versionslog	14
Abbildung 3: MVC-Muster	15
Abbildung 4: Abdeckung des Codes durch Unit-Tests	18
Abbildung 5: Die drei Bereiche des Sitzplaners	20
Abbildung 6: Einstellungen des Programms	21
Abbildung 7: Einstellungen zur Bildung der Population	24
Abbildung 8: Einstellungen zur Mutation der Chromosomen	25
Abbildung 9: Gewichtung der Relationen	27
Abbildung 10: Klassenraum mit Tafel, Schülern und einer Legende	28
Abbildung 11: Die Relationen einer Schülerin werden angezeigt	29
Abbildung 12: Verlauf der Fitness und Fitness der aktuellen Repräsentation	31
Abbildung 13: Wie viele Swaps je Nachkommen	33
Abbildung 14: Wie viele Inversionen je Nachkommen?	35
Abbildung 15: Wird Swap benötigt?	36
Abbildung 16: Wird Inversion benötigt?	38
Abbildung 17: Evaluation - Vergleich zwischen +-Strategie und ,-Strategie	40
Abbildung 18: Evaluation - Vergleich Best-N-, Rank- und Tournament-Selektion	41

III Tabellenverzeichnis

Tabelle 1: Verwendete Eclipse Plugins	13
Tabelle 2: Einstellungsmöglichkeiten Erzeugen und Berechnen.....	22
Tabelle 3: Einstellungsmöglichkeiten Laden und Speichern	22
Tabelle 4: Bearbeiten der Konfigurationen-Liste	23
Tabelle 5: Einstellungen zur Bildung der Population.....	25
Tabelle 6: Einstellungen zur Mutation der Chromosomen	26
Tabelle 7: Einstellung der Gewichtung für die Relationen	27
Tabelle 8: Evaluation – Parameter: Wie viele Swaps je Nachkommen?	33
Tabelle 9: Evaluation – Parameter: Wie viele Inversionen je Nachkommen?	34
Tabelle 10: Evaluation – Parameter: Wird Swap benötigt?	36
Tabelle 11: Evaluation – Parameter: Wird Inversion benötigt?	37
Tabelle 12: Vergleich Swap mit und ohne Inversion nach 10 Durchläufen	38
Tabelle 13: Evaluation – Parameter zum Vergleich zwischen +-Strategie und ,-Strategie .	39
Tabelle 14: Evaluation - Parameter: Best-N- vs. Rank- vs. Tournament-Selektion	41

IV Listings

Listing 1: Permutationscodierung - Π_i, j ist der Schüler an der Stelle (i,i)	4
Listing 2: Berechnung der Fitness.....	16
Listing 3: Selektion der neuen Elterngeneration aus der vorhandenen Population.....	17

V Literaturverzeichnis

- [1] H.-P. Schwefel, "Evolutionsstrategie und numerische Optimierung," Technische Universität Berlin, 1975.
- [2] I. Rechenberg, "Evolutionsstrategie–Optimierung technischer Systeme nach Prinzipien der biologischen Evolution," 1973.
- [3] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, 1988.
- [4] G. Syswerda, "Uniform crossover in genetic algorithms," 1989.
- [5] W. M. Spears and K. A. De Jong, "An analysis of multi-point crossover," DTIC Document, 1990.
- [6] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley, *A comparison of genetic sequencing operators*. Citeseer, 1991.
- [7] B. R. Fox and M. B. McMahon, "Genetic operators for sequencing problems," *Foundations of genetic algorithms*, vol. 1, pp. 284–300, 1991.
- [8] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs paralleles, re-seaux et systems repartis*, vol. 10, no. 2, pp. 141–171, 1998.