

# Implementation eines Ancestral Repair-Operators für Genetische Algorithmen

## Laborpraktikum

an der  
Fakultät für Informatik  
Otto-von-Guericke Universität Magdeburg

Name: Christian Schlager  
Geburtstag: 18. Oktober, 1980  
Geburtsort: Horb am Neckar, Deutschland  
Matrikelnummer: 162953

Betreuer: Dr. Christian Borgelt  
Frank Rügheimer

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>2</b>
<b>2</b>	<b>Überlegungen</b>	<b>4</b>
2.1	Genetische Algorithmen . . . . .	4
2.2	Backtracking . . . . .	5
2.3	RNS-Cache . . . . .	6
<b>3</b>	<b>Implementierung</b>	<b>7</b>
3.1	Genetischer Algorithmus . . . . .	7
3.2	Der Ancestral Repair-Operator . . . . .	10
3.3	Programmierung mit MFC . . . . .	11
3.4	Bedienung . . . . .	11
<b>4</b>	<b>Ergebnisse</b>	<b>14</b>

# Kapitel 1

## Einleitung

Die Vererbungslehre wurde unter anderem durch den Augustinerchorherr Gregor Mendel begründet, der 1866 in [Men66] die nach ihm benannten Mendel'schen Regeln formulierte. Inzwischen ist die moderne Genetik so weit vorangeschritten, dass mit Hilfe von DNS-Analysen Ausnahmen dieser Regeln bewiesen werden können. Während sich die meisten dieser Ausnahmen auf spezifische Gene beschränken, wurden auch Hinweise auf einen generellen Mechanismus zur Vererbung von Erbinformation außerhalb des Genoms gefunden.



Abbildung 1.1: Arabidopsis thaliana

Dieses Laborpraktikum basiert auf [Mag02]. In dem *Nature*-Artikel zur genetischen Forschung wird die Pflanzenart *Arabidopsis thaliana* vorgestellt, deren ungewöhnliche Nachkommenverteilung die Überlegung angestoßen hat,

dass eine extragenomische Weitergabe von Erbmaterial existiert. Der in der Folge entdeckte sogenannte RNS-Cache enthält RNS-Sequenzen, die von den Großeltern oder weiteren Vorfahren stammen und die Verteilung von Merkmalen unter den Nachkommen beeinflussen.

Das implementierte Programm und der vorliegende Bericht untersuchen die Tauglichkeit dieses Mechanismus für die Verwendung in genetischen Algorithmen.

# Kapitel 2

## Überlegungen

### 2.1 Genetische Algorithmen

Genetische Algorithmen bilden die natürliche Evolution nach. Die Entwicklung einer Art erfolgt durch die Anpassung an eine sich verändernde Umgebung. Die Fähigkeit zu überleben kann als die Information gesehen werden, die eine Art während ihrer Entwicklung sammelt und die ihre Mitglieder als Erbinformation in sich tragen.

Um genetische Algorithmen als Optimierungsverfahren zu benutzen, muss eine Kodierung vorliegen, die Lösungskandidaten für das Optimierungsproblem als Erbinformation beschreibt. Da Chromosomen, Träger der Erbinformation, aus einem einzigen fortlaufenden DNS-Strang bestehen, eignet sich eine Zeichenkette oder eine eindimensionale Folge von Zahlen am ehesten für diese Aufgabe.

Genetische Algorithmen werden vor allem für Probleme eingesetzt, für die keine geschlossene Lösung vorliegt und konkurrieren mit klassischen Suchstrategien wie dem A\*-Algorithmus, der Tabu-Suche oder dem Gradientenabstiegsverfahren.

Ein genetischer Algorithmen besteht aus den folgenden Schritten:

1. **Initialisierung:** Erzeugen (engl. generate) der Population mit unterschiedlichen Individuen (Lösungskandidaten).
2. **Evaluation:** Für jede Lösung wird anhand einer Evaluierungsfunktion die Fitness des Individuums bestimmt.

3. **Selektion:** Anhand eines Auswahlverfahrens werden Individuen aus der Population ausgewählt. Dabei werden Individuen mit höherer Fitness mit einer höheren Wahrscheinlichkeit ausgewählt.
4. **Rekombination:** die Allele verschiedener Individuen werden gemischt und aus den neuen Parametern eine neue Generation von Individuen erzeugt.
5. **Mutation:** Die Chromosomen der Individuen in der neuen Generation werden zufällig verändert.
6. **Neue Generation:** Nach einem bestimmten Verfahren wird die neue Population aus der Menge der alten Individuen und der Menge der mutierten Nachfolger gebildet. Der Algorithmus wird anschließend ab Schritt 2.1 wiederholt, oder nach einem vordefinierten Abbruchkriterium beendet und das fitteste Individuum als Lösung ausgegeben.

## 2.2 Backtracking

An dieser Stelle soll der Suchalgorithmus Backtracking untersucht werden, da die lokale Suche unter den Vorfahren eines Individuums der Rücknahme von Schritten beim Backtracking ähnelt.

Backtracking geht nach dem Versuch-und-Irrtum-Prinzip (*trial and error*) vor, d.h. es wird versucht, eine erreichte Teillösung schrittweise zu einer Gesamtlösung auszubauen. Wenn absehbar ist, dass ein Lösungszweig nicht zu einer endgültigen Lösung führen kann, kehrt der Algorithmus zu einer vorangegangenen Abzweigung zurück und es werden alternative Zweige probiert. Backtracking und die zugrundeliegende Tiefensuche haben im schlechtesten Fall eine exponentielle Laufzeit. Bei großer Suchtiefe und hohem Verzweigungsgrad dauert die Suche somit oft sehr lange. Weil bei der Ancestral Repair aber nur eine begrenzte Zahl an Vorfahren durchsucht werden soll, eignet sich das Backtracking für diese Aufgabe trotzdem.

Zur Verfeinerung und Verringerung der Zeitkomplexität eines Backtracking-Algorithmus können folgende Methoden verwendet werden:

- Heuristiken
- Akzeptanz von Näherungslösungen und Fehlertoleranz
- Durchschnittliche Eingabemenge

Da genetische Algorithmen ebenfalls auf Heuristiken und der Akzeptanz von Näherungslösungen basieren, ist eine Kombination der beiden Lösungsfindungsalgorithmen erfolgversprechend. Eine mögliche Kombination von genetischen Algorithmen und dem Backtracking-Algorithmus ist die Verwendung des RNS-Cache.

## 2.3 RNS-Cache

Der RNS-Cache wird als Kellerstack für einzelne Gene interpretiert. Bei einer Rekombination oder Mutation des Genmaterials werden die aktuellen Allele in den Kellerstack kopiert und an das neue Individuum übergeben. Danach erfolgt für jedes Individuum eine Evaluierung der Fitness und es werden gegebenenfalls Allele ausgetauscht.

Der Operator, der den Kellerstack benutzt, soll Ancestral Repair genannt und im Anschluss an die Mutation (Schritt 2.1) ausgeführt werden. Da jedes Individuum nur seinen eigenen Kellerstack zur Verfügung hat, wird die Reparatur nur mit Hilfe der Fitnessfunktion ausgeführt. Es werden also keine Vergleiche zwischen verschiedenen Individuen angestellt. Stattdessen werden die aktuellen Allele mit Vorgängerallelen rekombiniert und anhand der Fitnessfunktion festgestellt, ob eine fittere Kombination gefunden werden kann. Es findet also noch zusätzlich zur intergenomischen Rekombination eine intragenomische Rekombination statt.

# Kapitel 3

## Implementierung

### 3.1 Genetischer Algorithmus

Als Grundlage für die Implementierung wird der genetische Algorithmus aus [Mic96] verwendet. Wie man im Listing 3.1 erkennen kann, stand bei der beispielhaften Implementierung des Algorithmus Klarheit und leichte Verständlichkeit im Vordergrund. Die Schritte der iterierten Berechnung jeder neuen Generation entspricht den in 2.1 erwähnten Schritten.

```
1 void main(void)
2 {
3     initialize();
4     evaluate();
5     keep_the_best();
6     while(generation < MAXGENS)
7     {
8         select();
9         crossover();
10        mutate();
11        report();
12        evaluate();
13        elitist();
14
15        generation++;
16    }
17 }
```

Listing 3.1: Main Loop der Beispielimplementierung

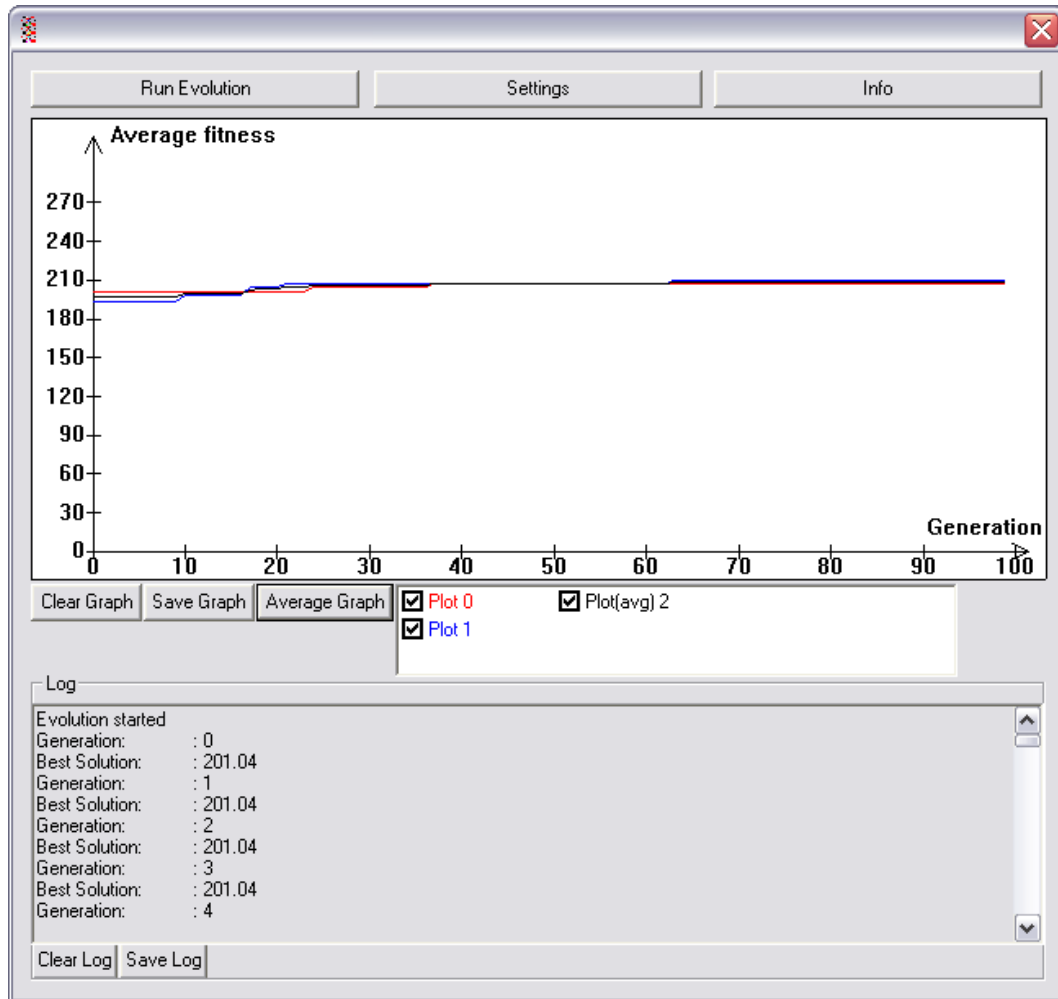


Abbildung 3.1: Das Anwendungsfenster

Für die Durchführung des Laborpraktikums wurde die Implementierung in einen objektorientierten Anwendungsrahmen überführt und um Fehlerabfragen, Einstellungsdialoge und eine graphische Ausgabe zur Verbesserung der Ergebnisdarstellung und Benutzerfreundlichkeit erweitert.

Im CAlgorithm-Objekt wird ein dynamisch erweiterbarer Vektor aus der C++ Standardbibliothek benutzt, um die Population abzuspeichern. Das ermöglicht die Änderung der Populationsgröße zur Laufzeit. Jedes Individuum wird durch die in Listing 3.2 gezeigte Struktur repräsentiert.

```
1 struct genotype
2 {
3     double gene[NVARS];
4     double ancestor[NVARS] [NANCESTORS];
5     double fitness;
6     double upper[NVARS];
7     double lower[NVARS];
8     double rfitness;
9     double cfitness;
10 };
```

Listing 3.2: Genotype-Struktur zur Repräsentation der Individuen

## 3.2 Der Ancestral Repair-Operator

Das Listing 3.3 zeigt den implementierten Operator. Die Verwendung des RNS-Cache wird als Reparatur-Operation interpretiert. Dazu werden bei Veränderungen durch Mutations- und Crossoveroperationen Sicherheitskopien der Allele im erwähnten Kellerstack abgelegt. Die Kommentare im Listing erläutern die Funktionsweise des Operators.

```
1 void CAlgorithm::AncestralRepair()
2 {
3     int mem, i, k;
4
5     for (mem = 0; mem < m_iPopSize; mem++)
6     {
7         // Zum Vergleich der Fitness mit dem tatsächlichen
8         // Chromosom wird eine Kopie erstellt
9         genotype temp = m_aPopulation[mem];
10
11        // Die Funktion durchläuft alle Gene und Vorfahren
12        for (i = 0; i < NVARs; i++)
13        {
14            for (k = 0; k < NANCESTORS; k++)
15            {
16                // Das Allel des Vorfahren wird wiederhergestellt
17                temp.gene[i] = temp.ancestor[i][k];
18
19                // Die Fitness des wiederhergestellten Allels wird errechnet
20                double x[NVARs + 1];
21                for (i = 0; i < NVARs; i++)
22                    x[i + 1] = temp.gene[i];
23                temp.fitness = (x[1] * x[1]) - (x[1] * x[2]) + x[3];
24
25                // Wenn das Chromosom mit dem wiederhergestellten Allel
26                // fitter ist als das tatsächliche, wird es ausgetauscht
27                if (temp.fitness > m_aPopulation[mem].fitness)
28                    m_aPopulation[mem] = temp;
29                else
30                    temp = m_aPopulation[mem];
31            }
32        }
33    }
34 }
```

Listing 3.3: Quellcode für den Ancestral Repair-Operator

## 3.3 Programmierung mit MFC

Die MFC-Bibliothek (MFC steht für Microsoft Foundation Classes) ist eine der wesentlichen Komponenten zur Entwicklung von Anwendungen auf der Windows-Plattform. Diese Sammlung verschiedener C++-Klassen kapselt einen großen Bereich der Win32-API und stellt einen leistungsfähigen Anwendungsrahmen für Anwendungen zur Verfügung.[TL99]

Aufgrund ihrer Tiefe und Komplexität ist die MFC schwierig zu erlernen und anzuwenden. Obwohl die Erstellung einer MFC-Anwendung durch Anwendungsassistenten unterstützt wird, ist die Programmierung der grundlegenden Dialogstruktur und von einzelnen Steuerelementen wie Radio Buttons und Checkboxes unverhältnismäßig zeitintensiv und aufwendig.

Zur Programmierung der Diagrammanzeige wird das Graphics Device Interface (GDI) benutzt, eine weitere Komponente der Windows Entwicklungsumgebung. Das Interface bietet Funktionen zum Zeichnen, Textdarstellung und Farbauswahl zur Verfügung und ist daher für graphische Darstellungen unter MFC die offensichtliche Wahl.

## 3.4 Bedienung

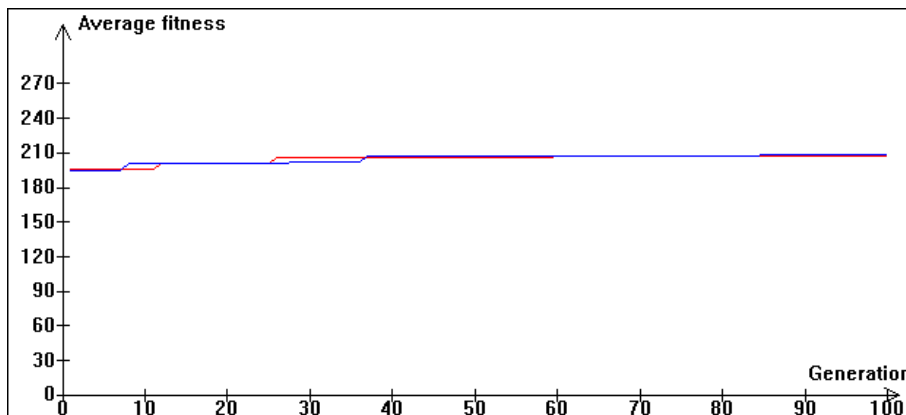


Abbildung 3.2: Anzeige der durchschnittlichen Fitness über der Laufzeit

Das Kernstück der Anwendung ist die in 3.2 dargestellte Graphenanzeige. Hier wird für jeden Lauf des Algorithmus die Veränderung der durchschnittlichen Fitness über den Generationen angezeigt. Die Auswahl der Plotfarbe

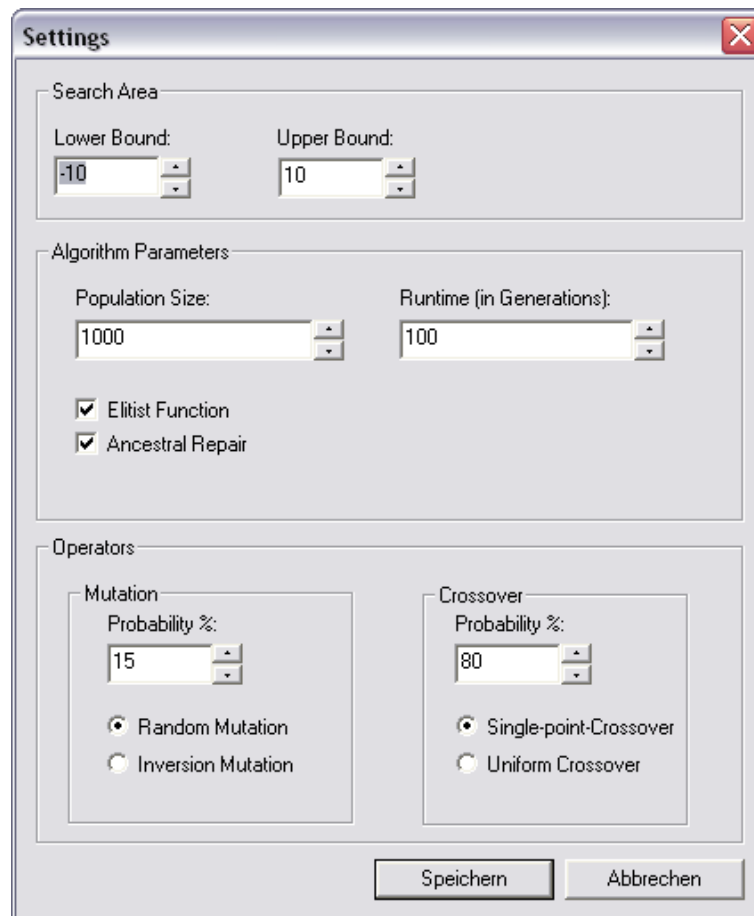


Abbildung 3.3: Das Dialogfenster für Einstellungen

erfolgt interaktiv. Die Skalierung und Verschiebung auf der y-Achse erfolgt automatisch. Damit die Anzeige bei unterschiedlicher Laufzeit von Plots einheitlich bleibt, werden vorhandene Plots bei einer Veränderung der Laufzeit entfernt. Um eine empirische Aussage über die Leistung des Algorithmus machen zu können, kann der Durchschnitt von mehreren Plots mit dem Button *Average Plot* gebildet werden. Der Durchschnitt wird als neuer Plot der Graphenanzeige hinzugefügt. Die Anwendung bietet außerdem die Möglichkeit, die gesamte Graphenanzeige als Bitmap abzuspeichern.

Zusätzlich wird der Lauf des genetischen Algorithmus von der Ausgabe im Logfenster dokumentiert. Die Anwendung ermöglicht es, die Ausgabe des Logfensters in einer Textdatei abzuspeichern.

Im Dialog 3.3 werden alle Einstellungen für den Algorithmus vorgenommen.

Alle Steuerelemente sind limitiert, das heißt es gibt festgesetzte Ober- und Untergrenzen für die Werte. Folgende Einstellungen sind möglich:

- Grenzen des Suchbereichs
- Populationsgröße und Laufzeit
- Elitist-Funktion aktivieren/deaktivieren
- Ancestral Cache-Funktion aktivieren/deaktivieren
- Mutationswahrscheinlichkeit für den Mutationsoperator
- Auswahl von Random oder Inversion Mutation
- Crossoverwahrscheinlichkeit für den Crossoveroperator
- Auswahl von Single-Point oder Uniform Crossover

Die Einstellungen wurden so gewählt, dass Testläufe des genetischen Algorithmus mit verschiedenen Parametern eine Aussage über die Auswirkungen des Ancestral

# Kapitel 4

## Ergebnisse

Mit dem Schematheorem ist eine Aussage über die Effektivität des Ancestral Repair-Operators möglich, indem die Wahrscheinlichkeit bestimmt wird, dass die Passung von Chromosomen verlorengeht oder erhalten bleibt. Es ist leicht zu sehen, dass die Wahrscheinlichkeit des Passungsverlustes gering ist, da Allele in der Ancestral Repair Operation nur mit ihren Vorfahren ausgetauscht werden.

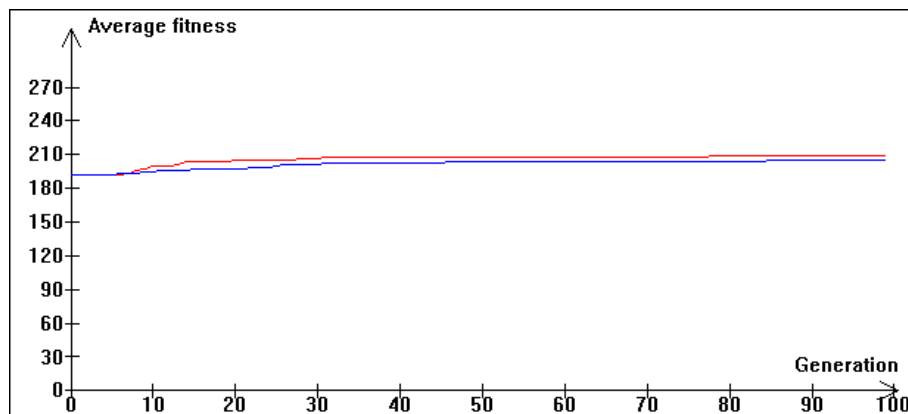


Abbildung 4.1: Die blaue Kurve zeigt den Verlauf bei aktivierter Ancestral Repair.

Die Graphik 4.1 zeigt, dass die Population mit aktivierter Ancestral Repair langsamer konvergiert. Das könnte man auf die Theorie zurückzuführen, dass Crossover- und Mutationsoperatoren problemabhängig sind. Laut Forschern wie zum Beispiel Eshelman ([ES89]) eignen sich bestimmte Operatoren für

eine bestimmte Klasse von Problemen eher als für andere. Es wäre interessant zu untersuchen, ob sich der Ancestral Repair-Operator für bestimmte Problemklassen eher eignet.

Da die Effektivität genetischer Rekombination auf dem Zusammenspiel verschiedener Mechanismen beruht, sollten auch genetische Algorithmen die Nachbildung der entsprechenden Mechanismen umfassen. Da einige Mechanismen wie der RNS-Cache erst in den letzten Jahren entdeckt wurden und die Entdeckung ihrer Funktionsweise und Wirkung noch aussteht, ist die Potentialsteigerung, die genetische Algorithmen durch diese Mechanismen erfahren können, noch nicht abzuschätzen.

# Literaturverzeichnis

- [ES89] L.J. Eshelman and J.D. Schaffer. *Biases in the Crossover Landscape*. Morgan Kaufmann Publishers Inc., 1989.
- [Mag02] Nature Magazine. Ancestral rns-cache. *Nature Magazine*, 2002.
- [Men66] Gregor Mendel. *Versuche über Pflanzenhybriden*. Verhandlungen des Naturforschenden Vereines in Brünn. Bd. IV, 1866.
- [Mic96] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag Berlin Heidelberg, 1996.
- [TL99] Viktor Toth and Dirk Louis. *Visual C++ 6 Kompendium*. Markt+Technik, 1999.