Universität Salzburg, Institut für Computerwissenschaften

PS Programmierpraktikum
SS 2000

Optimierte Objektpositionierung

Manfred Bernhard Mayr	mmayr@cosy.sbg.ac.at
Florian Angulanza	fangulan@cosy.sbg.ac.at
Christian Osterrieder	coster@cosy.sbg.ac.at
Markus Spitzlinger	mspitzl@cosy.sbg.ac.at

Projektbetreuer:
Dipl.Ing. Mag. Roland Schwaiger rschwaiger@cosy.sbg.ac.at

Version vom 2. Mai 2000

Zusammenfassung

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Genetischer Algorithmus

Nach [**?**] ist ein Genetischer Alogrithmus ein Algorithmus, der Strategien aus der Evolutionstheorie verwendet, um zu einem Problem eine möglichst gute Lösung zu finden. Bei dieser Heuristik muß man die Lösungen eines Problems als Chromosomen, aufgebaut aus einer Folge oder Menge von elementaren Bausteinen (Gene), formulieren; eine Menge solcher Chromosomen, eventuell erweitert um zusätzliche Attribute, heißt Population. Ausgehend von irgendeiner Anfangspopulation erzeugt man durch biologische Vererbungsmechanismen neue Chromosomen, bewertet diese mit Hilfe einer vorgegebenen Bewertungsfunktion (Fitnessfunktion) m und läßt gewisse Chromosomen überleben. Durch Wiederholung entsteht dann eine Folge von Populationen, in denen man hofft, eine besonders gute Lösung für das vorgegebene Problem zu entdecken. Als Vererbungsmechanismen verwendet man zumeist Kombinationen folgender Strategien:

- **Reproduktion:** Ein Chromosom wird unverändert an die nächste Population vererbt.
- **Mutation:** Ein oder mehrere Gene eines Chromosoms werden bei der Vererbung zufällig abgeändert.
- **Inversion:** Umdrehen eines Teilstücks eines Chromosoms und Vererbung des so modifizierten Chromosoms an die nächste Population.
- **Crossover:** Teilen zweier Chromosomen und Vererbung des über Kreuz zusammengesetzten Chromosoms. Hierbei muß man darauf achten, daß die Ergebnisse der Neukombination wieder gültige Chromosomen sind.
- **Blockung:** Zusammenfassen bestimmter Gene zu einer Einheit, deren Teile bei der weiteren Vererbung bei einer Inversion oder einem Crossover nicht mehr getrennt werden dürfen.
- **Entblockung:** Aufhebung einer Blockung.
- **Selektion:** Die bzgl. der Bewertungsfunktion besten Chromosomen überleben.

Kapitel 2

Projektbeschreibung

2.1 Einleitung

Es wird ein GA (Genetischer Algorithmus) benützt um Objektgruppierungen $o_1, ..., o_n$, in einer 2-dimensionalen Umgebung, mit Hilfe einer gegebenen Fitness-Funktion zu optimieren. Die verschiedenen Objekte o_i erhalten in einem vordefinierten Intervall $[l_i, u_i]$ Werte, gemessen in einem diskreten Zeitintervall, z. B. in einem Tag. Die 2-dimensionale Fläche ist in Teilflächen unterteilt, die einzelnen Teilflächen können

1. verboten sein,
2. zu einer vordefinierten Gruppe gehören,
3. besetzt sein, oder nicht bei einem Objekt o_i sein.

Ziel des Projektes ist, das Problem theoretisch zu untersuchen (ist es möglich dieses Problem zu klassifizieren?), eine GUI (grafische Benutzeroberfläche) und einen GA zu implementieren. Ein Ergebnis wird sein, Vorschläge für die Gruppierung in der Fläche zu machen (basierend auf einer allgemeingültigen Fitness-Funktion), z. B. maximal zuweisbare Werte der Objekte mit Rücksicht auf ein zugrundeliegendes Modell der verschiedenen Aufgaben.

Optimierte Objektpositionierung (OOP) ist ein Multiagentensystem, welches im wesentlichen aus zwei Komponenten besteht:

1. Agentenwelt
2. Objektwelt

Das Projekt unterteilt sich hierbei in natürlicher Weise in zwei Teilaufgaben:

1. Initialisierungsphase, welche die Agenten in der Welt simuliert und somit Daten für die zweite Phase generiert.
2. Optimierungsphase, wo mit Hilfe der Daten aus der Initialisierungsphase versucht wird die Objekte möglichst gut zu positionieren.

2.1.1 Agentenwelt

In der Initialisierungsphase werden die Agenten in der Welt W, mit den Dimensionen $w_x \in 1, ..., n$ und $w_y \in 1, ..., m$, in denen die Agenten und Objekte existieren, simuliert. Die Welt ist begrenzt durch Mauern, in denen Markierungen für Eingänge und Ausgänge enthalten sind. Agent $a_i \in A$ kann die Welt nach einer zugrundeliegenden Verteilung betreten und verlassen. Die Anzahl der Agenten in der Welt besitzt eine Obergrenze. Jeder Agent a_i besetzt ein gewisses Feld in der Welt welches

1. nicht besetzt von einem anderen Agenten ist,
2. nicht verboten ist,
3. nicht von einem Objekt besetzt ist.

Jeder Agent bekommt eine bestimmtes Guthaben v_i^a zugewiesen, welches erniedrigt wird sobald dieser vor einem Objekt anhält. Die Verminderung des Guthaben hängt dabei vom Objekt ab, welches der Agent gerade besucht und wird mit c_i^o festgehalten. Der Agent $\mathcal{A}$ kann dabei in der Welt zufällig herumwandern, folgt einem vordefinierten, oder evolvierten Schema. Das Ziel dabei ist alle Objekte mit einem $v_i^a > 0$ zu besuchen.

Die Agenten können dabei T Zeiteinheiten umherwandern und Objekte besuchen. Wir werden dies eine dynamische auf Generationen basierende Population nennen. Dynamisch deswegen, weil die Agenten die Welt jederzeit verlassen oder betreten können und auf Generationen basierend, weil nach $\mathcal{T}$ Zeiteinheiten eine andere Generation von Agenten die Welt bevölkert. Aktualisiert sollte die Simulation dabei synchron werden, z.B wenn eine Generation zum Zeitpunkt t gegeben ist hängt die Aktualiesierung füer den Zeitpunkt $t + 1$ nur vom Zeitunkt t ab. Alle Kollisionen mit anderen Agenten usw. werden basierend auf der jeweiligen Zeitstufe berechnet.

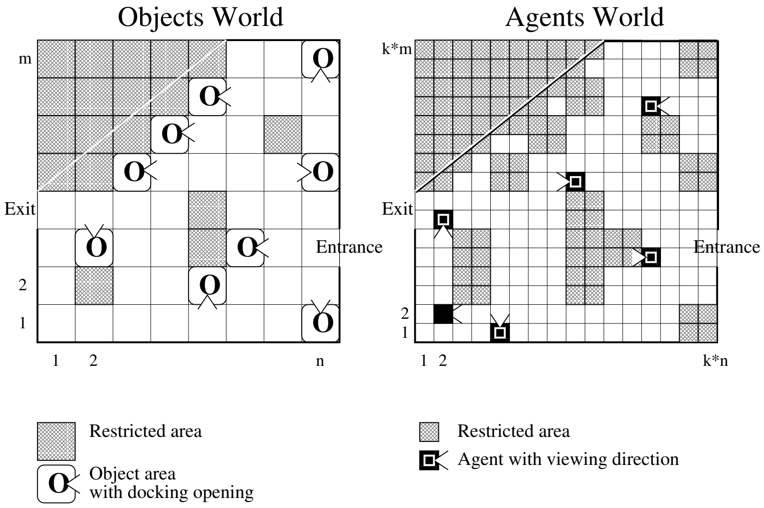


Abbildung 2.1: Objekt- und Agentenwelt

<ins>Beschreibung der Abbildung 2.1</ins>:

Links ist das Schema der Objektwelt dargestellt. Es wird ein Raster über die Objektwelt gelegt. Schwarze Rasterfelder bedeuten, dass kein Objekt (=Entity) hier positioniert werden darf. Ein weißes Rasterfeld bedeutet, dass dieses Feld leer ist. Objekte können nur auf weiße Felder gestellt werden. Objekte können nur vom Agenten einen Wert bekommen,wenn dieser genau vor der Öffnung des Objektes steht. Rechts ist das Schema der Agentenwelt abgebildet. Agenten sind für uns Klassen. Die Umsetzung der Bewegungen der Agenten erfolgt am Anfang ganz einfach mit Hilfe eines Zufallsgenerators. Die Anzahl der Agenten wird immer konstant gehalten, d.h. wenn ein Agent die Agentenwelt verläßt wird ein anderer nachgeschickt.

Um genug Daten für den Optimierungsphase sammeln zu können, wird die Grundeinstellung der Objekte nach einem Durchlauf verändert.

2.1.2 Objektwelt

Im Grunde hat die Objektwelt die gleiche Größe wie die Agentenwelt, aber sie ist viel ungenauer. Das ist deshalb so, weil die Agenten in der Regel nicht soviel Raum wie unsere Objekte benötigen. Objekte besetzen eine Einheit in der Welt. Objekte werden einem bestimmten Typen zugeordnet und folglich werden ihnen bestimmte Kosten c_i^o zugewiesen, die dem Agenten von seinem Wert abgezogen werden, falls das Objekt vom Agenten besucht wurde. Das Objekt sammelt die reduzierten Werte des Agenten und nach einem Durchlauf hat das Objekt den Wert v_i^o.

Nach jedem Durchlauf wird die Grundeinstellung der Objekte solange verändert, bis genug Daten für den Optimierungsphase gesammelt wurden. Einige der Objekte können durch andere Objekte aus dem Objektvorrat ersetzt werden.

2.2 Initialisierungsphase

Diese Phase dient dazu, die für den Optimierungslauf notwendigen Daten zu ermitteln und zu sammeln. Hier werden über einen Zeitraum von $p * \mathcal{T}$ Datensätze angelegt, wobei p z.B. 30 (Tage) und $\mathcal{T}$ z.B. 24 Stunden $*$ 60 Minuten sind. Es finden also insgesamt p Durchläufe statt, wobei vor jedem Durchlauf die Grundeinstellungen (Objekte, deren Verteilung, ...) verändert werden. Nach jedem Durchlauf wird festgehalten welches Guthaben die einzelnen Objekte an der bestimmten Postion haben. Durch die Veränderung der Grundeinstellungen vor jedem Experiment (=Durchlauf) entstehen genügend Vergleichsdaten für die Optimierungsphase.

Hierbei stellt sich jetzt die Frage, wie die einzelnen Agenten positioniert werden und nach welchem Schema sie sich bewegen. In unserem Fall wird dies zuerst ganz einfach mit Hilfe eines Zufallsgenerators realisert. Die Bewegung und die Postionierung sind Methoden, sodaß sie zu einem späteren Zeitpunkt einfach durch möglicherweise bessere ersetzt werden können.

Wichtig dabei ist auch, daß die Anzahl der Agenten konstant gehalten wird, d.h. daß wenn ein Agent die Welt betritt ein anderer sie verläßt und umgekehrt.

Eine weitere Frage ist, ob überwacht werden muß, daß die Objekte durch ihre Aufstellung die Agenten nicht behindern. Ein Beispiel wäre, daß etwa alle Objekte in einer Reihe stehen und kein Agent mehr zu einem Objekt kann, oder der Ein- oder Ausgang blockiert ist. Diese Problem müßte jedoch der GA lösen, da sich diese Aufstellung der Objekte als nicht sehr vorteilhaft klassifizieren müßte.

Wie ermittelt man welchen Fitnisswert ein Objekt hat wenn es zwei Einträge in der Liste gibt? z.B.: Objekt 1 hat am Platz 1 zuerst 50 Punkte und beim nächsten Mal 100 Punkte bekommen.

- Ermittelt man dies durch Zufall?
  - Dies ist sehr schlecht, da der GA zuerst einen Fitnisswert bekommt und beim nächsten Mal einen anderen. Welcher gilt?
- Maximum?
- Durchschnitt?

Dies ist noch zu klären.

Am Ende der Initialisierungsphase werden die gesammelten Daten zu einem Datensatz zusammengefügt und stehen anschliessend dem Optimierungsalgorithmus als Ausgangsbasis zur Verfügung.

Datum	Objekt	Wert
02.03.2000	1	1000
02.03.2000	2	3000
...	...	...

2.3 Optimierungsphase

In dieser Phase wird versucht, aus den Daten, welche aus der Initialisierungsphase gewonnen wurden (können auch empirische Daten sein) eine optimale Postitionierung der Objekte zu finden. Ziel dabei ist es eine Aufstellung zu finden, bei welcher die Summe der Werte aller Objekte möglichst hoch ist. Realisiert wird diese Optimierung mit Hilfe eines GA's.

2.3.1 Motivation warum ein GA verwendet wird

- Wahrscheinlich ist dieses Problem ein Assigment-Problem.
- Ein Assigment-Problem ist jedoch NP-Vollständig!
- Herkömlicher Lösungsansatz:
  - Gegeben ist die Liste der Objekte, ihre Postion und ihrer Wertigkeit:
  - z.B.:

Objekt	Positon	Wert
1	1	1000
1	2	300
2	1	900
2	2	100
...	...	...

  - Nun sucht man einfach aus der Liste den Besten heraus (in unserem Fall Objekt 1 auf Postion 1) und positioniert ihn dort. Das Ergebnis ist in diesem Beispiel folgendes:

Objekt	Positon	
1	1	1000
2	2	100
...	...	...

    Dies ergibt ein Gesamtergebnis von 1100.

    Besser wäre jedoch folgendes:

Objekt	Positon	
1	2	300
2	1	900
...	...	...

    Da das Gesamtergebnis hier 1200 ist!

2.3.2 Evolutionärer Zyklus

Wir bilden unser System anhand der Generationsbasierten Methode, d.h. es werden Populationen von Individuen gebildet, indem am Ende einer Generation Fitness-Werte bestimmt werden, und zwar in Bezug auf eine zuvor bestimmte Fitness-Funktion. In unserem Fall wird jedem Individuum (= Codierung einer Objektposition) die Summe aller Werte zugeschrieben, die von den Agenten dem Individuum zugewiesen werden.

Als nächstes stellt sich die wichtige Frage: Wie soll man die Objektposition in einen bit string codieren? In unserer Simulation wollen wir eine Objektposition durch zwei Chromosome codieren. Chromosom 1 (C_1) repräsentiert die verschiedenen möglichen Positionen von Objekten in der Objektwelt. All diese Positionen sind grundsätzlich nicht-gesperrte Felder. Da Objekt- und Agentenwelt in Matrix-Form organisiert sind, können wir die Objektwelt Zeile für Zeile auslesen und pro nichtgesperrtem Feld ein Bit vorsehen. In diesem Fall repräsentiert 1 das Vorhandensein und 0 das Nicht-Vorhandensein eines Objektes auf diesem bestimmten Feld. Das zweite Chromosom (C_2) codiert den Index der Objekte. C_2 kann grösser, gleich gross oder kleiner sein als C_1. Da wir den Index der Objekte codieren, müssen wir mehr als ein Bit für den Index verwenden, nämlich $\lceil \log_2(\#objects) \rceil$. Weiters müssen wir aufpassen, dass im Falle einer Mutation ein Objekt genau einmal in C_2 aufscheint. Daher müssen wir den Mutations-Operator auf folgende Weise abändern:

1. Bestimme in Bezug auf die Mutationswahrscheinlichkeit p_m einen Mutationsort auf C_2.
2. Wähle zufällig einen zweiten Ort auf C_2.
3. Tausche die beiden Elemente gegeneinander aus.

Die Verwendung des obigen Algorithmus garantiert, dass ein Objekt genau einmal in C_2 vorkommt. Bei C_1 können wir den Operator der Standard-Bit-Mutation verwenden.

In Abbildung 2.2 zeigen wir das crossing over für zwei verschiedene Individuen. Für Chromosom C_2 könnten wir den gleichen Mechanismus anwenden, aber wir müssen garantieren, dass ein Objekt genau einmal vorkommt.

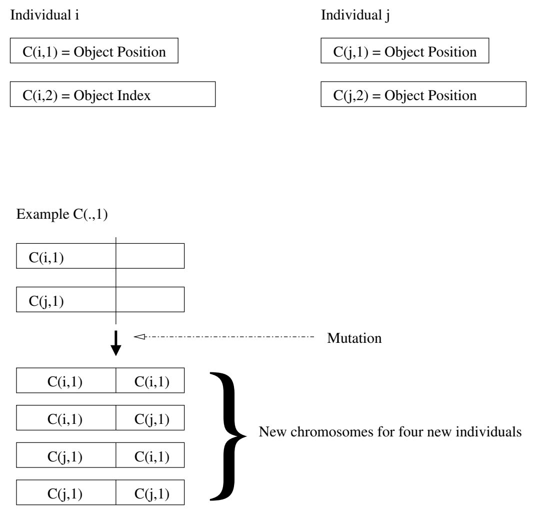


Abbildung 2.2: Crossing over zwischen Chromosom C_1 der Individuen i_i und i_j.

Diese Erfordernis impliziert deutlich die Modifikation des crossover-Operator auf nicht-triviale Weise. Daher beabsichtigen wir, das original C_2 der Individuen i_i und i_j zu kopieren und behalten einen ursprünglichen Satz. Somit bekommen wir wieder vier Chromosome.

Wie müssen wir die relevanten Objekte für Orte mit Wert = 1 in C_1 bestimmen? C_1 wird von links nach rechts ausgelesen. Wann immer eine 1 im bit string auftaucht, lesen wir den passenden Objektindex C_2 und erhöhen den Zähler für C_2.

2.4 Anwenderfälle

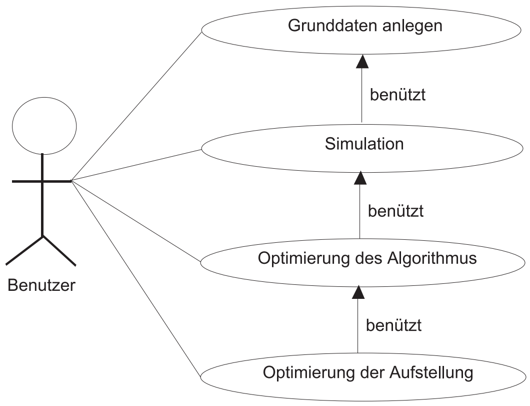


Abbildung 2.3: use-case-diagram

1. **Betriebsart Grunddaten anlegen**

   Unter Grunddaten wird verstanden, dass der Benutzer diese Daten auf alle Fälle eingeben muß.
   Grunddaten sind:

   (a) Die Dimension und die Rasterung der Superklasse Welt:
   Die Form der Welt muß immer die Form eines Rechtecks haben. Es müssen folgende Daten eingegeben werden:

i. Die Werte m und n.
Der Wert m gibt an wieviele Objekte in Y-Richtung - siehe Abbildung 2.1 - maximal nebeneinander aufgestellt werden können. Der Wert n gibt an wieviele Objekte in X-Richtung maximal nebeneinander aufgestellt werden können. Es wird angenommen, dass die Objektabmessung quadratisch ist.

ii. Eingabe von Wert k.
Der Wert k gibt das Verhältnis Agentenaktionsradius zu Objektabmessung an. Die Fläche die ein Agent in der Agentenwelt für sich alleine beansprucht, ist der Agentenaktionsradius.

(b) Festlegung der Sperrflächen.
Die Definition, wo eine Sperrfläche ist, erfolgt über die Rasterflächen in der Agentenwelt. Das heißt, es wird für jedes Feld festgelegt, ob es sich um eine Sperrfläche handelt oder nicht.

(c) Festlegung der Ein- und Ausgänge.
Die Definition der Ein- und Ausgänge erfolgt über zwei Punkte.

(d) Eingabe der Objekte.
Ein Objekt hat einen Namen und Kosten c_i^0.

Der Benutzer, der die Grunddaten eingibt, hat die Möglichkeit die Richtigkeit seiner Eingaben anhand einer graphischen Darstellung oder in Form einer Tabelle zu kontrollieren.

2. **Betriebsart Simulation**

Der Benutzer kann zwischen einer automatischen Generierung der Daten und einer benutzerdefinierten Eingabe der Daten wählen. Diese Auswahl beeinflusst die weitere Bedienung des Programmes:

benutzerdefinierte Eingabe:
Der Benutzer muss die Objekte in der Objektwelt positionieren. Diese Positionierung kann entweder mit einer grafischen Benutzeroberfläche oder mit einer Eingabeaufforderung mittels der Koordinaten der Objektwelt (m,n) erfolgen. Weiters muss vom Anwender die Ausrichtung der Objekte angegeben werden. Diese Ausrichtung orientiert sich nach den Himmelsrichtungen (N - oben, S - unten, W - links, O - rechts) und bezieht sich auf die Darstellung am Bildschirm.
Weiters wird vom Anwender die Agentenanzahl und ein Mittelwert des Budgets der Agenten angelegt.

automatische Generierung:
Das Programm nimmt die Positionierung der Objekte und deren Ausrichtung zufällig vor. Die Anzahl und das Budget der Agenten wird vom Programm angelegt. Das Programm generiert eine feste Anzahl von Experimentenlisten (30 Tabellen).

Das Experiment kann mittels einer grafischen Anzeige der Welt (inkl. aller sich darin befindlichen Sperrflächen, Objekte, Ein- und Ausgänge und Agentenbewegungen), am Bildschirm verfolgt werden. Diese grafische Anzeige erfolgt in vom Benutzer bestimmten Zeitintervallen (0 - 100 % der Experimentendauer).
Durch eine einfache Anzeige soll der Benutzer den Fortschritt der Simulation verfolgen können.
Die Simulation kann vom Benutzer gestartet, unter- und abgebrochen werden.
Das Experimentenergebnis wird in einer Tabelle angelegt und kann als solches vom Benutzer verändert, gespeichert, neu geladen und ausgegeben werden. Falls das System genügend Experimente zur Verfügung hat (30 Experimentenergebnisse) kann vom Benutzer eine Tabelle erzeugt werden, die das Simulationsergebnis darstellt. Um diese Datengenerierung vorzunehmen, können Experimentenergebnisse geladen werden. Falls zuvor die automatische Generierung der Daten ausgewält wurde, werden die Daten aus der automatischen Generierung verwendet, der Anwender kann in diesem Fall keine anderen Experimentenergebnisse laden.
Das Simulationsergebnis kann gespeichert, gelöscht und angezeigt werden.

3. **Betriebsart Optimierung der Objektaufstellung**

Als Basisdaten für die Optimierung der Objektaufstellung dienen die Grunddaten, d.h. die Dimensionierung der Objekt- und Agentenwelt und deren Raster, definierte Sperrflächen und Ein-/Ausgänge sowie die angegebenen Objekte o_i mit ihren Parametern (Name, Kosten).

Dem Benutzer soll es möglich sein, einen Optimierungsalgorithmus zu laden. Grundsätzlich steht als „default"-Wert ein Basisalgorithmus zur Verfügung. Der Benutzer kann nun den Optimierungsvorgang starten, in dem der gewählte Algorithmus eine möglichst gute Objektaufstellung berechnet. Nach Abschluss dieses Vorgangs kann sich der Benutzer die berechnete Objektaufstellung grafisch oder tabellarisch anzeigen lassen.

4. **Betriebsart Optimierung des Algorithmus**

   Es müssen mehrere Simulationsergebnisse bereitstehen, die dazu dienen, den Optimierungsalgorithmus zu verbessern und/oder fur gewisse Problemstellungen zu optimieren.

   Dem Benutzer soll es nun möglich sein, einen vorhandenen Optimierungsalgorithmus zu laden oder den Basisalgorithmus als Voreinstellung auszuwählen. Ausserdem kann der Benutzer Simulationsergebnisse, die in der Betriebsart Simulation errechnet worden sind, laden, um den Algorithmus für die Optimierung zu verbessern. Dabei soll eine History über bereits verwendete Simulationsergebnisse Auskunft geben. Diese History enthält Namen der Simulationsergebnisse sowie die zeitlichen Daten ihrer Verwendung für die Optimierung.

   Nach Vornahme dieser Einstellungen kann der Benutzer nun den Start der Algorithmus-Optimierung veranlassen. Das erfolgreiche Ende der Optimierung wird dem Benutzer mitgeteilt, dieser kann den neuen Algorithmus nun speichern.

2.5 Klassendiagramm

2.6 Snapshots der Benutzeroberflächen:

1. Darstellung des Hauptfensters:
   In diesem Hauptfenster wird die grafische Anzeige der Welt dargestellt. Links davon befindet sich eine Liste mit den bereits angelegten Objekten. Diese Objektliste wird in 4 Spalten unterteilt. Die erste Spalte zeigt die automatisch vergebene Nummerierung des Objektes. Die zweite Spalte zeigt den vom Benutzer eingegeben Namen des Objektes. Die dritte und vierte Spalte zeigt die Position und die Ausrichtung des Objektes an. Position und Ausrichtung der Objekte wird standardmäßig vom Programm definiert. Indem der Benutzer die Schaltfläche Auto abschaltet, kann die Position und Ausrichtung vom Benutzer verändert werden.
   Unter den Anzeigen für Objektliste und Welt befinden sich ein Schalter zurück und vor. Mit Hilfe dieser Schalter können die verschiedenen Simulationen betrachtet werden. Die Zahl zwischen den beiden Schaltern zeigt die Nummer der Simulation.
   Der Schalter optimiert zeigt die Objektliste und Welt der optimierten Aufstellung.

2. Darstellung der Auswahl Datei, aus der Menüleiste des Hauptfensters:
   Hier kann ein neues Projekt gestartet, ein Projekt geladen, gespeichert und das Programm beendet werden.

3. Darstellung der Auswahl Grunddaten - Welt, aus der Menüleiste des Hauptfensters:
   Mit dieser Auswahl kein eine neue Welt angelegt werden, eine bereits bestehende Welt bearbeitet und geladen werden.

4. Darstellung der Auswahl Grunddaten - Objekte, aus der Menüleiste des Hauptfensters:
   Hier kann die Objektliste geladen und bearbeitet werden.

5. Darstellung der Auswahl Grunddaten - Agenten, aus der Menüleiste des Hauptfensters:
   Hier kann die Agentenliste geladen und bearbeitet werden.

6. Darstellung der Auswahl Start, aus der Menüleiste des Hauptfensters:
   Mit dieser Auswahl kann eine Simulation und die Optimierung gestartet werden.

7. Darstellung der Auswahl Optionen, aus der Menüleiste des Hauptfensters:
   Unter Optionen kann ein Algorithmus geladen und optimiert werden. Weiters können hier diverse Einstellungen getroffen werden.

8. Eingabefenster für Daten der Welt:

9. Eingabefenster für Daten der Objektliste:
10. Eingabefenster für Daten der Agentenliste:
11. Eingabefenster für Einstellungen:
12. Eingabefenster für Algorithmusoptimierung:

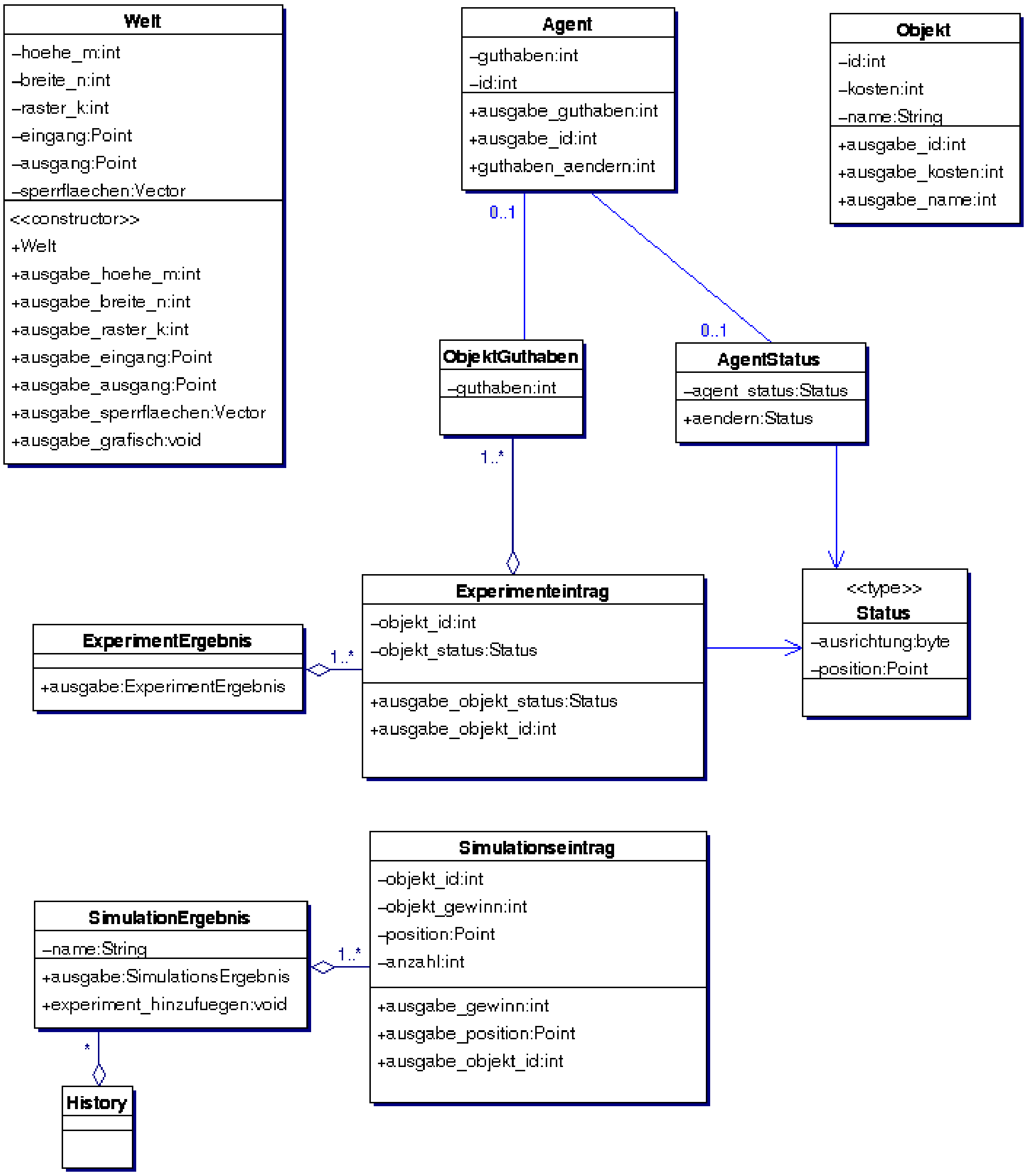


Abbildung 2.4: Klassendiagramm

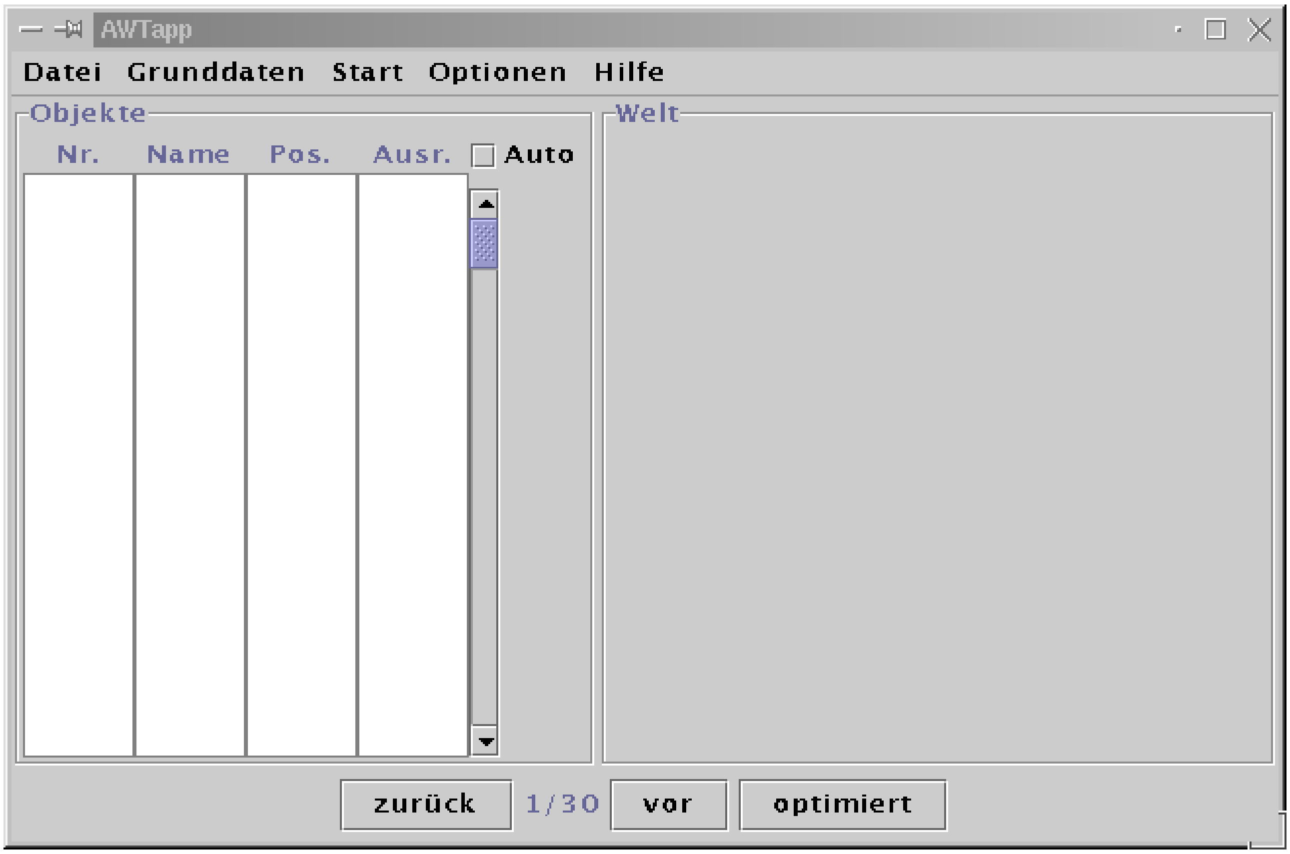


Abbildung 2.5: Hauptfenster

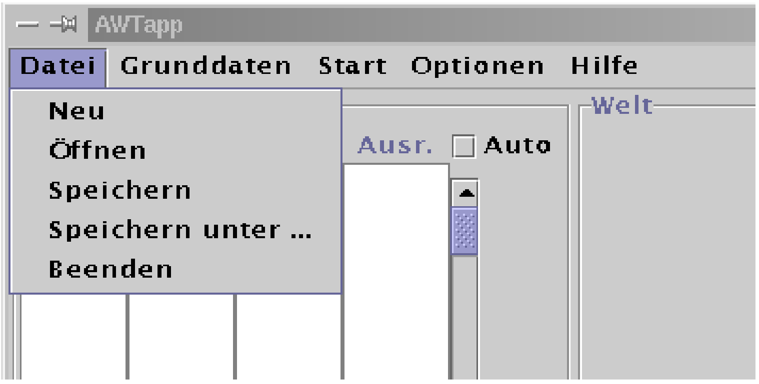


Abbildung 2.6: Menü: Datei

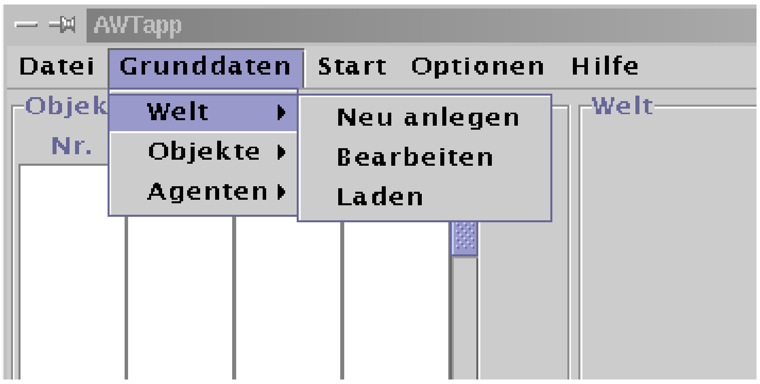


Abbildung 2.7: Menü: Grunddaten - Welt

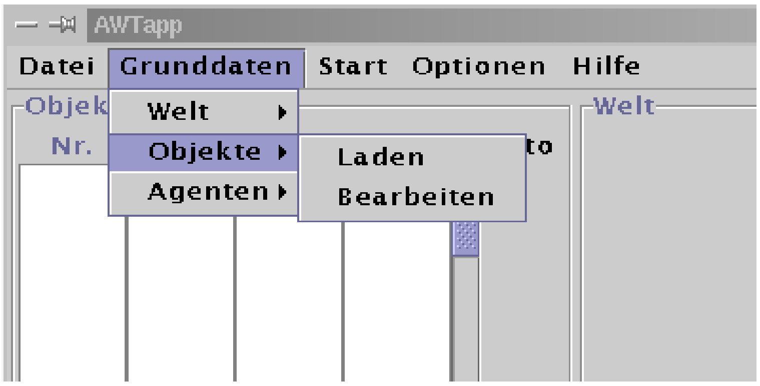


Abbildung 2.8: Menü: Grunddaten - Objekte

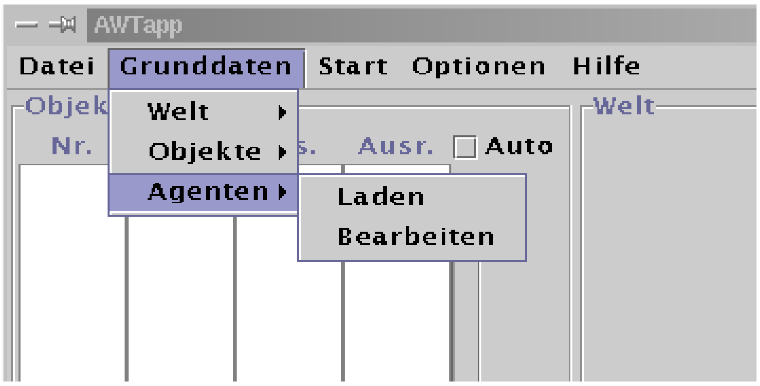


Abbildung 2.9: Menü: Grunddaten - Agenten

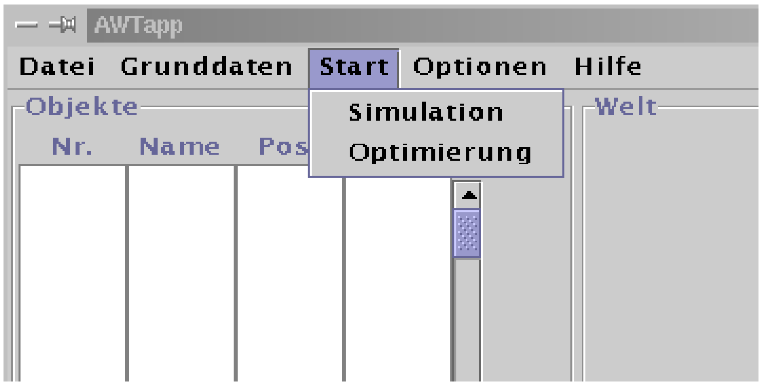


Abbildung 2.10: Menü: Start

Abbildung 2.11: Menü: Optionen

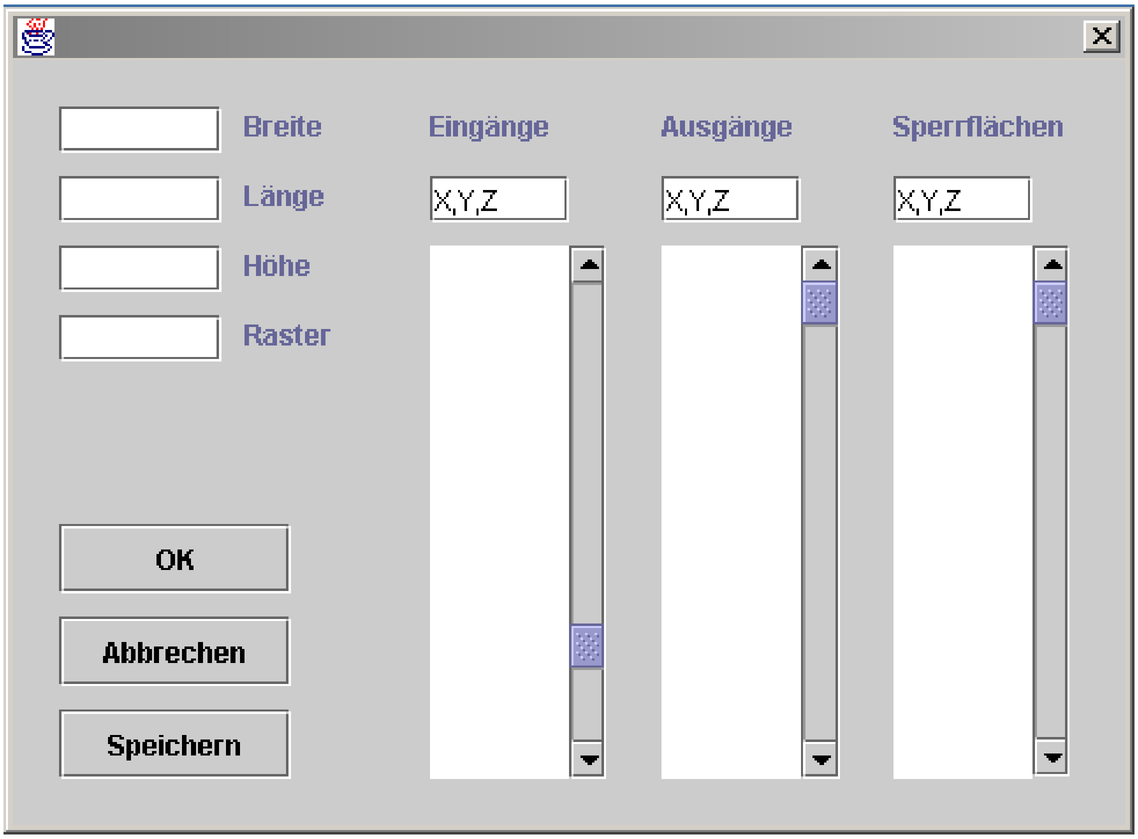


Abbildung 2.12: Eingabefenster für Daten der Welt

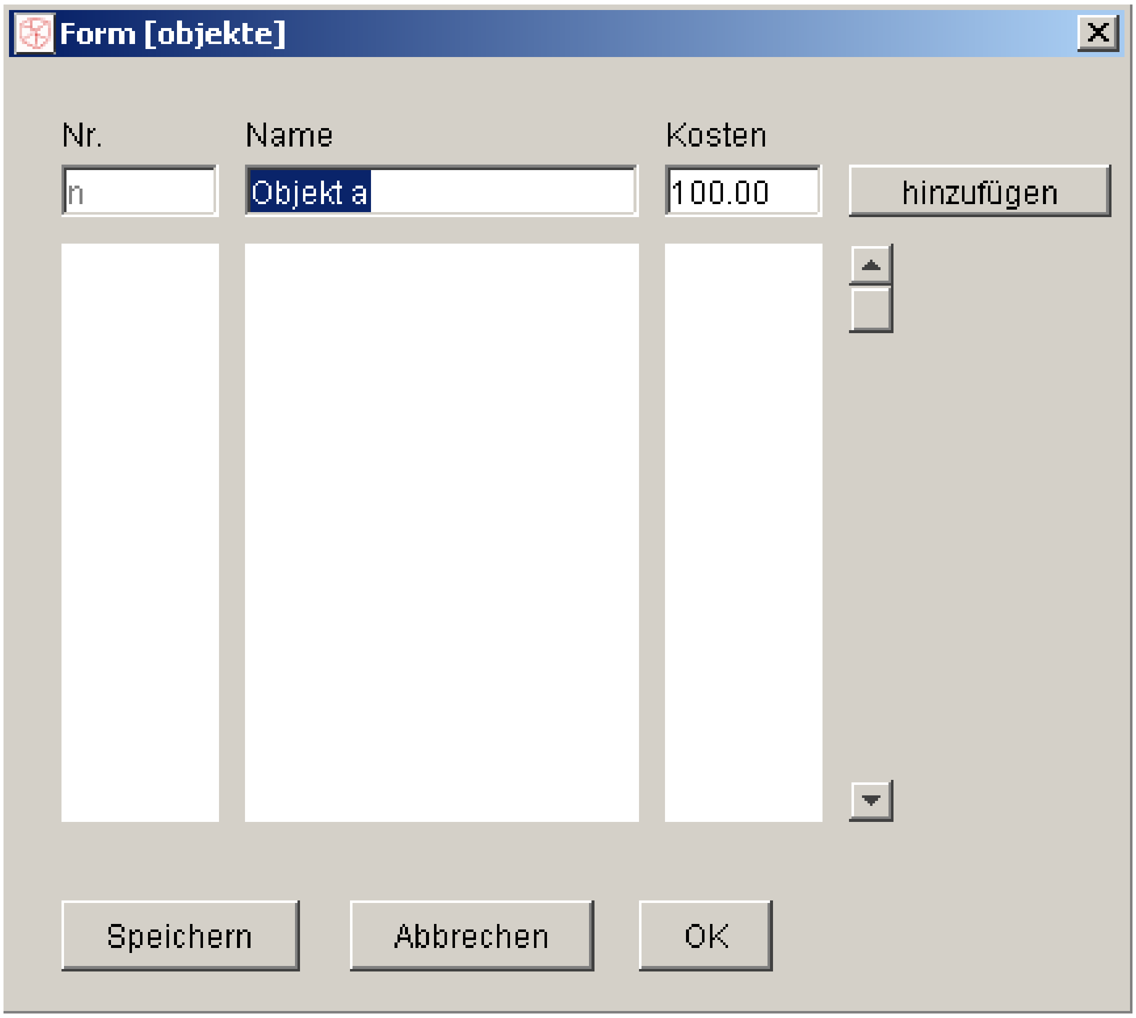


Abbildung 2.13: Eingabe der Objekte

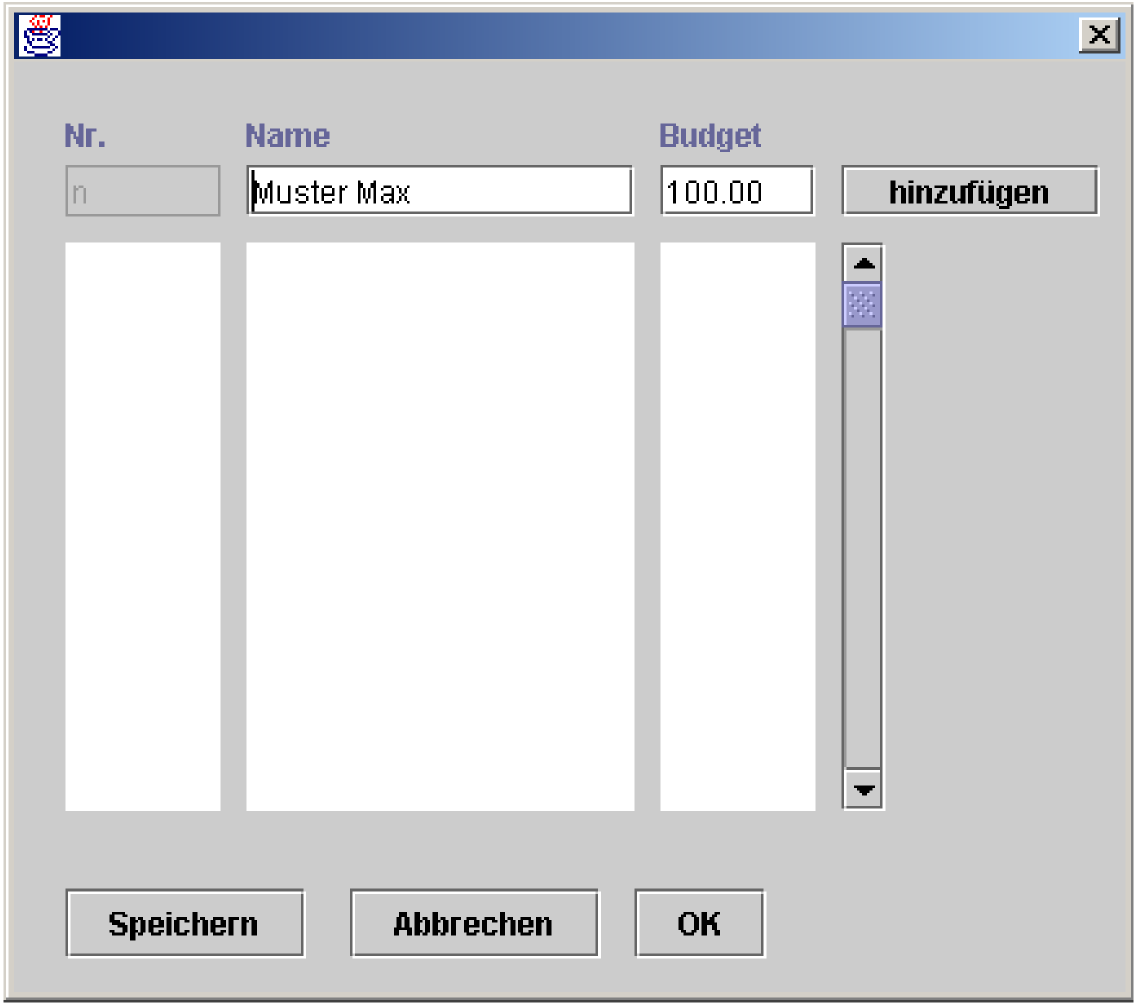


Abbildung 2.14: Eingabe der Agenten

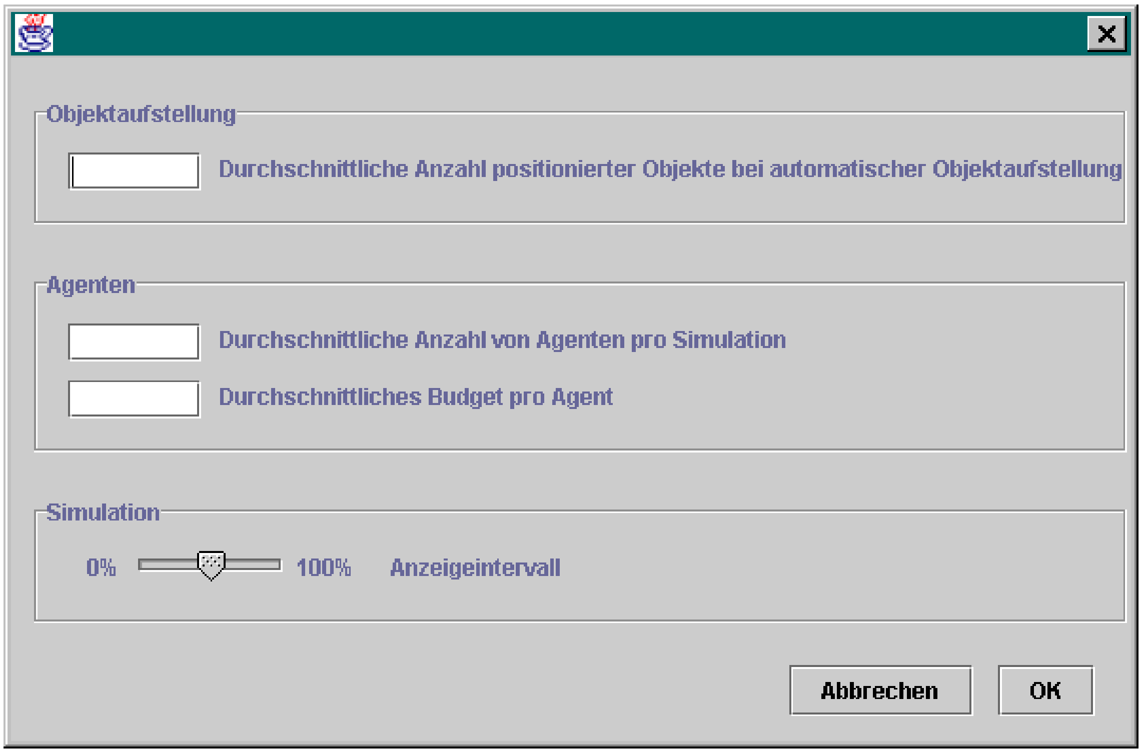


Abbildung 2.15: Einstellungen

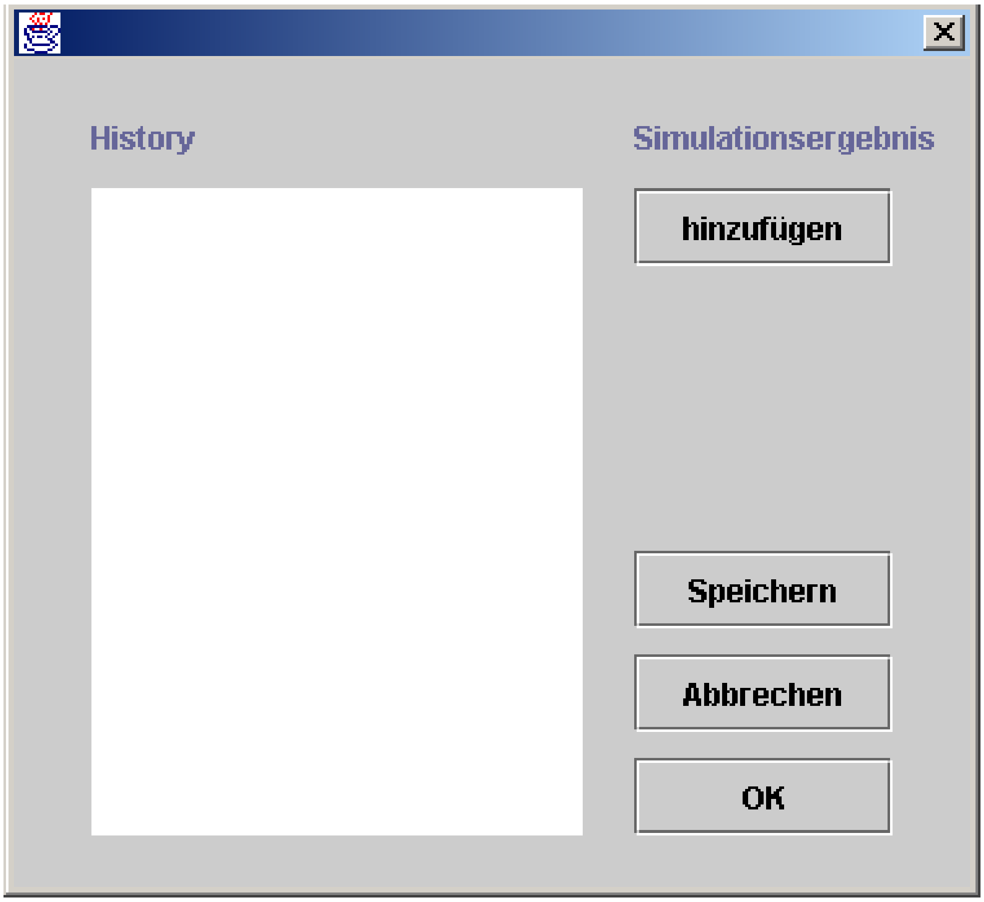


Abbildung 2.16: Fenster für Algorithmusoptimierung

Anhang A

Protokolle

A.1 Protokoll vom 15. März 2000

Leiter:	DI Mag. Roland Schwaiger
Protokollführer:	Florian Angulanza
Teilnehmer:	Manfred Bernhard Mayr
	Christian Osterrieder
	Markus Spitzlinger
Ort:	Salzburg
Zeit:	17^{30}

- Treffen wöchentlich, später weniger oft
- Was ist zu tun im Laufe des Praktikums:
  - Paper erstellen (in LaTeX)
  - Objektbeschreibung in UML (Unified Modelling Language)
  - Beherrschung von HTML
  - Programmieren in Java
- Beschreibung als LaTeX-file bis Mi 17^{30} als .TeX-Source
- Beschreibung der Grafik im Paper:
  Grafik besteht aus folgenden Elementen:
  - Objekten (Entities)
    - bewegen sich
    - Werte werden gesammelt
  - Agenten
    - bewegen sich anders als Objekte
    - kann nicht auf Objektfelder
  - Schwarzen Flecken
    - sind verbotene Zonen
  - Ein- und Ausgängen
- OOP ist ein Multiagentensystem

- Beschreibung des Diagramms im Paper:

  1. **Initialphase:**
     z.B.: System 24 Stunden laufen lassen / mit einer Auflösung von 1 Minute Daten aufzeichnen → Tagesergebnis $\mathcal{T}$ ergibt sich aus $24 * 60$ Minuten.
     Dies wird wiederholt, z.B. 30 mal → 1 Monat → 30 Experimente. Immer zwischendurch an den Objekten Änderungen machen → *Vergleichsdaten*.

Datum	Objekt	Value
02.03.2000	1	1000
02.03.2000	2	3000
...	...	...

  2. **Hauptphase:**
     Gleiche Laufzeit (z.B. 1 Tag), Optimierung aufgrund von Daten aus der Initialphase → *Genetischer Algorithmus* für Optimierung, dann Agenten wieder laufen lassen.

- 1. Aufgabe:
  - LaTeX-file aus ~rschwaig/out-dir/ppktII2000/gruppe2 holen und in eigenes Dokument kopieren
  - Dokument ergänzen
  - Fragen einbinden

- Nächster Termin: Mi, 22. März 2000

A.2 Protokoll vom 22. März 2000

Leiter:	DI Mag. Roland Schwaiger
Protokollführer:	Markus Spitzlinger
Teilnehmer:	Manfred Bernhard Mayr
	Christian Osterrieder
	Florian Angulanza
Ort:	Salzburg
Zeit:	17^{30}

- Was ist UML und wozu wird es verwendet?
  - UML ist die Abkürzung für Unified Modelling Language.
  - UML dient dazu die Entwicklung von Software zu dokumentieren und ist prinzipiell eine graphische Beschreibungssprache.

- UML in unserem Projekt:
  - Ein Tool um Dokumentationen in UML zu erstellen ist ARGO-UML (auf der Hompage von DI Mag. Roland Schwaiger zu finden (XML-Parser von IBM ist notwendig!).
  - Klassendiagramme und Benutzerfälle sollen mit UML-dargestellt werden.

- Wie werden die Agenten positioniert?
  - Ein Agent ist in diesem Fall eine Klasse.
  - Die Bewegung des Agenten wird zuerst ganz einfach mittels Zufallsgenerator realisiert (einfache Methode).
  - Die Anzahl der Agenten wird immer konstant gehalten, d.h. wenn ein Agent die Agentenwelt verläßt wird ein anderer nachgeschickt.

- Motivation warum ein GA verwendet wird:

- Wahrscheinlich ist dieses Problem ein Assigment-Problem.
- Ein Assigment-Problem ist jedoch NP-Vollständig!
- Herkömlicher Lösungsansatz:
  - Gegeben ist die Liste der Objekte, ihre Postion und ihrer Wertigkeit:
  - z.B.:

Objekt	Positon	Wert
1	1	1000
1	2	300
2	1	900
2	2	100
...	...	...

  - Nun sucht man einfach aus der Liste den Besten heraus (in unserem Fall Objekt 1 auf Postion 1) und positioniert ihn dort. Das Ergebnis ist in diesem Beispiel folgendes:

Objekt	Positon	
1	1	1000
2	2	100
...	...	...

Dies ergibt ein Gesamtergebnis von 1100.

Besser wäre jedoch folgendes:

Objekt	Positon	
1	2	300
2	1	900
...	...	...

Da das Gesamtergebnis hier 1200 ist!

- Wie wird überwacht, daß die Objekte die Agenten durch ihre Aufstellung nicht behindern, z.B. alle in einer Reihe wodurch es möglich wäre, daß kein Agent mehr zu einem Objekt kann?
  - Dies müßte der GA lösen, da die Objekte keine Punkte bekommen und somit als gute Lösung ausscheiden.
- Wie ermittelt man welchen Fitnisswert ein Objekt hat wenn es zwei Einträge in der Liste gibt? z.B.: Objekt 1 hat am Platz 1 zuerst 50 Punkte und beim nächsten Mal 100 Punkte bekommen.
  - Ermittelt man dies durch Zufall?
    - Dies ist sehr schlecht, da der GA zuerst einen Fitnisswert bekommt und beim nächsten Mal einen anderen. Welcher gilt?
  - Maximum?
  - Durchschnitt?

  Dies ist noch zu klären.
- Aufgabenstellung bis nächste Woche:
  - Welche Datenstrukur wird verwendet?
  - Welche Felder hat die Tabelle bzw. Datenbank?
  - Motivation GA ins Dokument einbinden.
  - Mit UML vertraut machen.
  - Dokument erweitern.

A.3 Protokoll vom 29. März 2000

Leiter: DI Mag. Roland Schwaiger
Protokollführer: Christian Osterrieder
Teilnehmer: Manfred Bernhard Mayr

	Markus Spitzlinger
	Florian Angulanza
Ort:	Salzburg
Zeit:	17^{30}

- Datenstruktur, welche Klassen werden vorerst benötigt?
  - Agentenklasse
  - Objektklasse
  - Klasse Tagesliste
  - Klasse Monatsliste

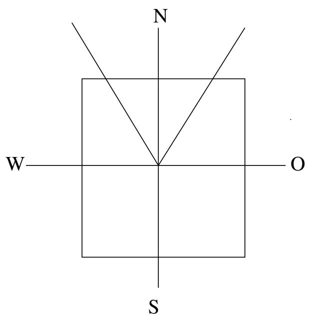
  Vorschlag für Tagesliste:

Objektbezeichner	Position	Wert
1	$(x, y)_1$	w_1
2	$(x, y)_2$	w_2
.	.	.
.	.	.
.	.	.
n	$(x, y)_n$	w_n

  Um eventuell später Daten weiterzuverarbeiten, sollten die einzelnen Zwischenwerte zum Zeitpunkt t gespeichert werden:

Objektbezeichner	Position	Wert			
1	$(x, y)_1$	w_t	w_{t+1}	...	w_{t+m}
2	$(x, y)_2$	w_t	w_{t+1}	...	w_{t+m}
.	.	.	w_{t+1}	...	w_{t+m}
.	.	.	w_{t+1}	...	w_{t+m}
.	.	.	w_{t+1}	...	w_{t+m}
n	$(x, y)_n$	w_t	w_{t+1}	...	w_{t+m}

- Objekte und Agenten haben eine Orientierung:

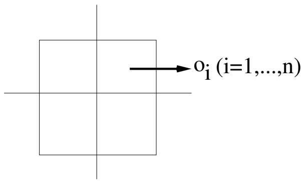


- weiters sollen folgende Daten mitprotokolliert werden:
  - Orientierung von Objekten
  - Orientierung von Agenten
  - von welchem Agenten kommt das Guthaben
  - Querverbindung: welcher Agent war an welchem Objekt
  - zu welchem Zeitpunkt kommt das Guthaben von einem Agenten

  d. h. eine Simulation sollte wiederholbar sein – Zufallsgenerator mit seed verwenden

- Klasse Monatsliste: Warum Orientierung an Stellplätzen und nicht an Objekten?

  Angenommen man hat mehr Objekte (n) als Stellplätze (m): $n > m$, Ziel ist es, aus einem Stellplatz das Maximum "herauszuholen" – Assignementproblem!

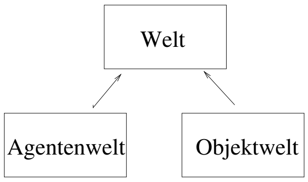


- Warum wird eine Monatsliste erstellt? Kann der GA nicht die Tageslisten verwenden?

  Falls ein Objekt mehrmals am gleichen Stellplatz war, wird z. B. das Maximum verwendet. Die Tagessicht wird dadurch eleminiert. Der GA benützt die Werte mehrerer Monatslisten nach folgendem Schema:
  Periode → Datensammlung → Optimierung → Periode → Datensammlung → Optimierung → usw.

- Datenstruktur der Welten:

  Agentenwelt und Objektwelt werden von einer "Welt" vererbt.



- Eine grafische Darstellung soll eingebaut werden:
  - wie bewegen sich die Agenten
  - wie ist die Welt aufgebaut
  - diese Darstellung sollte abschaltbar sein, da es die Rechenleistung mindert
  - die Schritte zwischen Darstellung und Berechnung sollten festgelegt werden können

- Aufgabenstellung bis zum nächsten Treffen:
  - Überlegung wie die Daten abgelegt werden (ASCII oder XML) – Java parsing!
  - mit UML vertraut machen (Powerpoint-Einführung)
  - Use-Cases entwickeln und zusätzliche Informationen erfragen

- Weitere Bemerkungen:
  - nächste Woche entfällt (5. April 2000)
  - GA steht noch nicht fest, wird bis zum nächsten Treffen erledigt.

A.4 Protokoll vom 12. April 2000

Leiter:	DI Mag. Roland Schwaiger
Protokollführer:	Manfred Mayr
Teilnehmer:	Markus Spitzlinger
	Christian Osterrieder
	Florian Angulanza
Ort:	Salzburg
Zeit:	17^{30}

- Besprechung des Klassendiagramms
  - Klasse `Welt`
    Wir sollen unsere Welt so konstruieren, so dass sie 3 Dimensionen hat.
    Die Welt kann mehrere Ein- und Ausgänge haben. Diese Tatsache muss im Klassendiagramm berücksichtigt werden. Das heißt Änderung des Typs bei den Attributen `eingang` und `ausgang`. Es wird festgelegt, dass es nur Ein- und Ausgänge am äußeren Rand der Welt geben kann. Diese Öffnugen werden jeweils über zwei Punkte definiert. Die Welt soll Außenwände haben, die auch graphisch dargestellt werden. Durchbrüche in diesen Wänden entsprechen Ein- oder Ausgängen. Ein Eingang kann aber muß nicht auch Ausgang sein und umgekehrt. Sperrflächen werden über X- und Y-Koordinaten bestimmt. Ist es vielleicht doch besser ein Metaobjekt einzuführen?
    Es soll ein Methode hinzugefügt werden, die es möglich macht Sperrflächen zu ändern. Diese Änderungen sollen auch intern dokumentiert werden.

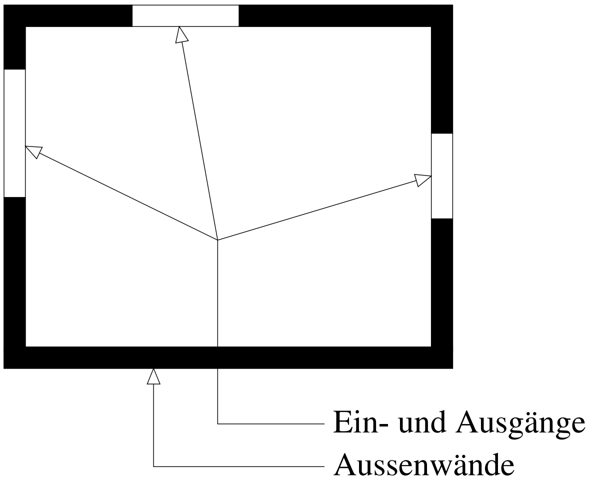


Abbildung A.1: Aussenwände und Öffnungen

  - Klasse `Agent`
    Es soll möglich sein den Agenten einen Namen zu geben.
    Dies stellt einen Beitrag zur Erhöhung der Flexibilität des Programmes dar.
  - Klasse `Objekt`
    Es soll ein besserer Name für das Attribut `kosten` gefunden werden. Es soll möglich sein die Kosten eines Objektes zu ändern.
  - Typ `Status`
    Die Ausrichtung des Objekts wird im Sourcecode mit den Bezeichnungen `N`, `S`, `O`, `W` definiert.
  - Klasse `ObjektGuthaben`
    Die Bezeichnung `Guthaben` ist schlecht gewählt, besser `Detaileintrag`. Es soll die Zeit mit protokolliert werden. Daher soll ein Attribut `Zeitschritt` in der Klasse `ObjektGuthaben` eingefügt werden.

Zeitschritt	Agent	Objekt
1	a_1	o_5
1	a_{10}	o_3
2	a_{12}	o_7
2	a_{17}	o_3
2	a_{33}	o_{45}
2	a_{52}	o_{17}
...	...	...

    Motivation:
    Da in unserer Tabelle nur ein Eintrag erfolgt, wenn ein Objekt das Guthaben eines Agenten reduziert,

kann ohne das Attribut `Zeitschritt` nicht festgestellt werden, wenn der Übergang vom Zeitschritt i auf Zeitschritt $i + 1$ erfolgte.

  – Klasse `SimulationErgebnis`
    Begründung warum es keine Verweise auf die Experimentenergebnisliste gibt: Es gibt in der Klasse `SimulationErgebnis` eine Methode `experiment_hinzufuegen`, die alle notwendigen Daten, die in der Experimentenergebnisliste existieren, in die Simulationsergebnisliste einfügt. Daher werden keine Verweise mehr auf die Experimentenergebnisliste mehr benötigt. Der Algorithmus kann dann direkt auf der Simulationsergebnisliste arbeiten. Die Folge ist Performancesteigerung während der Laufzeit.

- Genetischer Algorithmus

  Was haben wir?

  – Eine Liste von Objekten m
  – Eine Liste von Objektpositionen n
    Wie schaut die Beziehung zwischen m und n aus?
    Schwer zu sagen, da die Relation m zu n wie folgt sein kann: $m > n$, $m < n$ oder $m = n$.
  – Mutation (siehe Zusammenfassung)
  – Crossover (siehe Zusammenfassung)

- Aufgabenstellung bis zum nächsten Treffen

  – Prototypen von GUI entwicklen
    * Was muß man sehen?
    * Was für Fenster gibt es?
    * Wie wird die Welt dargestellt?
    * usw.
  – Ergänzung der Zusammenfassung

- Sonstige Bemerkungen

  – Vielleicht ist es möglich das Projekt zu publzieren.
  – Erweiterung im Dokument „Project Ideas“ werden wieder ins Netz gesetellt.
  – Nächstes Treffen: 12.Mai 2000, 18.00 - 18.30.

Anhang B

Glossar

Blockung: Zusammenfassen bestimmter Gene zu einer Einheit, deren Teile bei der weiteren Vererbung bei einer Inversion oder einem Crossover nicht mehr getrennt werden dürfen.

Crossover: Teilen zweier Chromosomen und Vererbung des über Kreuz zusammengesetzten Chromosoms. Hierbei muß man darauf achten, daß die Ergebnisse der Neukombination wieder gültige Chromosomen sind.

Entblockung: Aufhebung einer Blockung.

GA: Genetischer Algorithmus Algorithmus, der Strategien aus der Evolutionstheorie verwendet, um zu einem Problem eine möglichst gute Lösung zu finden.

Inversion: Umdrehen eines Teilstücks eines Chromosoms und Vererbung des so modifizierten Chromosoms an die nächste Population.

Mutation: Ein oder mehrere Gene eines Chromosoms werden bei der Vererbung zufällig abgeändert.

NP: Menge aller Sprachen, die von einer nichtdeterministischen Turingmaschine in polynomialer Laufzeit erkannt werden können (Determinismus). Hinter dem Kürzel NP verbirgt sich eines der schwierigsten und für die Praxis bedeutsamsten Probleme der Informatik, nämlich die Frage, ob gewisse Aufgabenstellungen schnelle Lösungsverfahren besitzen oder nicht.

Reproduktion: Ein Chromosom wird unverändert an die nächste Population vererbt.

Selektion: Die bzgl. der Bewertungsfunktion besten Chromosomen überleben.