if ((Port1 > input1) = true) [if ((Port1 < 8) = true) [set Port1 input1]]
set Port1 (7 + 5)
set Port1 (input1 ^ 2)

Evolutionäre Programmierung

set input1 (8 ^ 6)

Jugend forscht 2007/2008
Mathematik-Informatik

set input1 (9 - Port1)
set input1 (Port1 + 3)

von
Peter Schmitz

St. Michael-Gymnasium
Bad Münstereifel

set input1 (input1 / Port1)
if ((Port1 != input1) = true) [set input1 9]
if ((Port1 > input1) = true) [set Port1 (3 / 5)]
if ((Port1 != 4) = true) [if ((Port1 != 4) = true) [set input1 3]]

Inhaltsverzeichnis

1. Kurzfassung:

Evolutionäre Programmierung

Ausschlaggebend für meine Arbeit waren Gemeinsamkeiten der Fachgebiete Biologie, Sozialwissenschaften und Informatik, die mich zu folgenden Fragen führten: *Lassen sich Analogien zwischen der Evolutionstheorie und anderen Selbstorganisationsprozessen bilden? Können Lebewesen auf Informationsebene existieren? Halten sich Lebewesen aktiv in einem optimalen Mutationsfenster? Ist es möglich die Evolution technisch zu nutzen?*

Anfangs war es mein Plan gewesen, eine ungerichtete Evolution ohne feste Selektionsbedingungen zur Simulation von lebenden Datenstrukturen in einem Speicher zu verwenden. Nach einiger Beschäftigung mit der Thematik musste ich jedoch feststellen, dass die Simulation eines natürlichen Lebensprozesses noch nicht im Rahmen meiner Fähigkeiten lag.

Ich beschäftigte mich also intensiver mit genetischen Algorithmen, die eine gezielte Evolution mit festen Selektionsbedingungen verwenden. Diese sind nützliche Optimierungswerkzeuge für vielfältige Anwendungen wie z.b. Oberflächenanpassung, Routenplanung oder Kalkulation von passenden Formeln.

Mein vorläufiges Ziel war ein evolutionärer Algorithmus, der selbstständig Programme erzeugt, die an vorgebbare Bedingung angepasst sind. Zur Programmierung nutzte ich das Multiagentensystem „Netlogo“.

In einer Reihe von Vorversuchen testete ich die Komponenten des von mir entworfenen Algorithmus auf simpler Ebene und machte Experimente zur Mutation. Anschießend erstellte ich ein System, dass die Einhaltung der Syntax-Regeln der Programmiersprache garantiert, und setzte alle Komponenten in komplexer Form um, so dass eine Codeevolution stattfinden konnte.

Durch Analyse verschiedener Experimente gewann ich neue Erkenntnisse über den erforderlichen syntaktischen Aufbau evolutionärer Algorithmen. Auf dieser Basis entwickelte ich einen adaptiven Optimierungsalgorithmus für „Netlogo“-Simulationen. Statt wie bisher alle Parameterkonfigurationen durchzutesten nutzt dieser evolutionäre Methoden und findet selbst bei mehreren Milliarden Möglichkeiten schnell die optimale Einstellung.

2. Einleitung

2.1 Ideenbildung:

Es begann mit dem Science-Fiction-Roman „der Schwarm“ von Frank Schätzing, der von dem Konflikt der Menschheit mit einer uralten Einzeller-Rasse handelt, die eine kollektive Intelligenz besitzt. Eine Forscherin simuliert im Roman ein intelligentes Kollektiv aus Einzellern, weist dabei jedem Objekt verschiedenartige Speicher und Sensoren zu und entwirft eine Problemstellung, die das Kollektiv durch Selbstorganisation löst. Durch den Denkanstoß des Buches und meine spätere Analogienbildung zwischen Evolutionstheorie und Marktwirtschaft kam ich zu der Frage:

Ist der Prozess des Lebens von dem Medium der Chemie unabhängig und auf eine Informationsebene übertragbar?

Auch wenn ich später aus Gründen der Komplexität meine Ziele abändern musste, blieb diese Frage der Grundantrieb meiner Arbeit.

2.2 Synthetische Evolutionstheorie

Die heute anerkannte synthetische Theorie ist eine Verbindung von Darwins Ansatz der Evolutionstheorie und neueren Erkenntnissen der Genetik. Sie beruht auf folgenden Annahmen:

- Jedes Lebewesen ist durch sein Erbgut Teil eines Kombinationspools. Genotypisch ist die spezifische Kombination eines Lebewesens als Reihenfolge und Verwendung der 4 chem. Grundbausteinen der Erbsubstanz („Basen“) definiert.
- Eine Umgebung stellt Selektionsbedingungen, nach denen sich die Entwicklung der Population richtet.
- Gemäß den Bedingungen werden Kombinationen ausgewählt, abgeändert und begründen den besser angepassten Pool der nächsten Generation (siehe Abb.1)

<u>Erbgutveränderungen:</u>
Sowohl die Beschädigung des Erbgutes durch mutagene Stoffe und Strahlung, als auch der Austausch von Code mit Partnern bei der Rekombination werden als exogene Einflüsse klassifiziert. Anteilsmäßig geringe Mutation durch Übersetzungsfehler beim kopieren und Ablesen des Erbgutes sind endogener Natur.

Evolution ist laut Darwin kein aktiver Prozess: Es findet keine Entwicklung mit endlichem Ziel statt, sondern vielmehr eine unendliche Verbesserung durch Selbstorganisation, weshalb sie auch als urtümlichste Form des Lernens gilt.

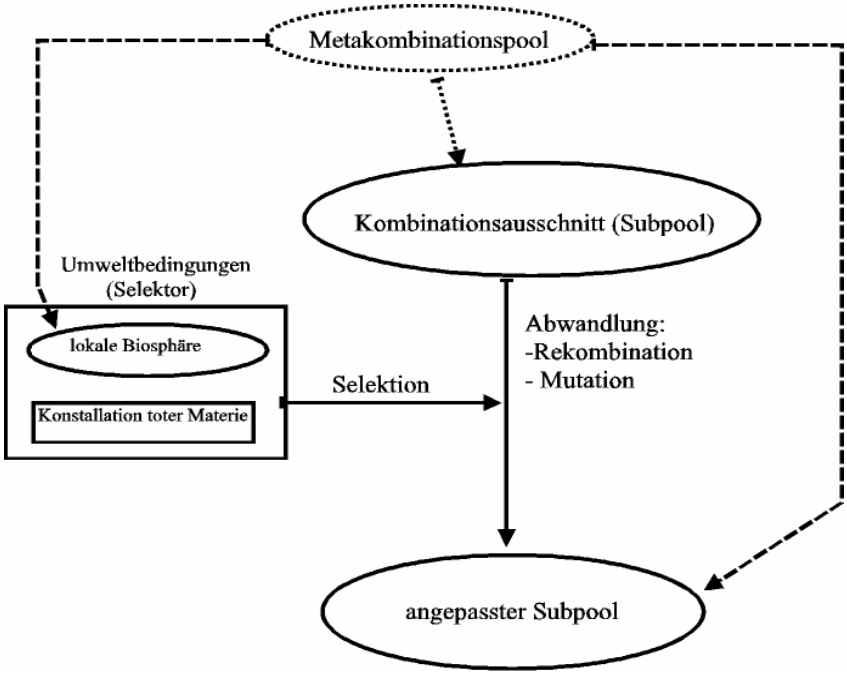

Abb. 1: Anpassung einer Gruppe an Selektionsbedingungen

3. Zielsetzung: Evolutionärer Algorithmus

Im Gegensatz zur natürlichen Evolution werden Aussonderungsbedingung (Selektion) und Evolutionsstrategie (Vermehrung, Mutation) fest vorgegeben. Durch einen Informationsinput, bestehend aus Selektionsbedingungen und zu kombinierenden Bausteine (Eigenschaften oder Code), erzeugt der Algorithmus eine nahezu perfekt passende Lösung.
Da jede mögliche Kombination eine einzigartige Zusammenstellung aus Grundbausteinen des Meta-Kombinationspools ist, entscheidet die Art der kombinierten Bestandteile über das Anwendungsgebiet des Algorithmus.

Praktischer Nutzen:
Evolutionäre Algorithmen werden schon seit den sechziger Jahren zur Optimierung von Flächen und Winkeln genutzt, wenn auch eher auf mechanische als auf virtuelle Art. Obwohl sich für manche Bereiche andere Lösungsmöglichkeiten als nützlicher erweisen liegt der Vorteil der E.A.s in ihrer allumfassenden Anwendungsmöglichkeit. Rechenberg arbeitete beispielsweise mit Winkeloptimierungen von Tragflächen und aquadynamischer Anpassung von Strukturen [1]

Ziel:
Das von mir entworfene Programm soll zu einer Reihe eingegebener Werte die am besten übereinstimmende und algebraisch kürzeste Funktion finden.

Meta-Kombinationspool/Ziel:
In der natürlichen Evolution ist das Auftreten eines Merkmals nicht absolut durch die Gene definiert, sondern kann eher als eine Funktion gesehen werden, die bei jedem Input aufgenommener Substanzen einen typischen Output an umgewandelten Stoffen liefert. Um eine größtmögliche Nähe zum natürlichen Prozess einzuhalten kombiniert mein genetischer Algorithmus keine „starren“ Eigenschaften, sondern Formeln und Programmcode.

Gruppe von Objekten/Erbgut:
Jede Kombination bekommt einen virtuellen „Körper“ zugewiesen, über den sie am Evolutionsprozess teilnimmt. Somit muss jedes Objekt einen Informationsspeicher besitzen. Das von mir entwickelte Programm nutzt Einzeller-artige Objekte als Basis der Evolution, da sie die einfachste Ebene des Lebens darstellen.

Syntaktisch sicheres System:
Ob man Code oder DNA-Basen kombiniert: Es existiert immer ein Regelwerk, dass die Erstellung einer Kombination im Rahmen der verwendeten Sprache sichert. Die molekulare Zusammensetzung der Basen und des Phosphatbandes einer DNA wird bei Übertragung der Evolution auf eine Informationsebene durch ein System ersetzt, dass nur im Rahmen der Programmiersprache kombiniert einen Code für die Plattform lesbaren Code erzeugt.

Abwandlung des Erbgutes:
Am Beispiel der natürlichen DNA-Mutation ist zu erkennen, wie vielfältig die Abläufe sind, die zu einer Änderungen des Codes führen. Die Mutationsfunktion von EAs muss kalibriert werden um eine optimale Evolution zu ermöglichen.

4. Informationen

4.1 Evolutionsstrategien (nach [1]):

Rechenberg versuchte seinen Algorithmus stufenweise der natürlichen Evolution anzunähern:
(In einer Klammer addierte Variablen werden gemeinsam/ bei einer Kommatrennung nur die Filialgeneration nach dem Selektionskriterium bewertet)

Strategie: (1 + 1)
Am simpelsten ist das sog. „Trial-and-Error-Verfahren“, bei dem eine zufällige Kombination geschaffen und abgeändert wird. Erfüllt die neue Kombination die Selektionsbedingungen besser als die vorige wird sie beibehalten, im anderen Fall wird die Änderung rückgängig gemacht.

Strategie: ($m + l$)
„m“ Eltern des Ausgangspools erzeugen „l“ mutierte Nachkommen. Die gesamte Population wird der Selektion unterworfen und die „m“ besten Individuen werden selektiert und zum Ausgangspool der nächsten Generation. Die genetische „Fitness“ des besten Individuums kann sich nur verbessern, d.h. die Kurve der Fitness über Generationen ist monoton steigend.

Strategie: (m/r , l)
Selbst bei relativ simplen Bakterien wird das Erbgut durch exogene und endogene Prozesse verändert. Bei eine Rekombination (biol. „Crossover“) werden „r“ Teile des Erbgutes ausgetauscht, wodurch evolutiv nützliche Eigenschaften schneller in der gesamten Population verbreitet sind als bei einer ($m + l$)-Strategie.

4.2 Das Multiagentensystem Netlogo

Zur Verifizierung meiner Annahmen durch Simulation vor der endgültigen Übertragung in eine eigenständig agierende Informationsstruktur eignen sich besonders Multiagentensysteme, deren Agenten unterschiedliche Aktionen ausführen können und sich wie Individuen verhalten. Zu Beginn meines Projekts hatte ich kaum Programmiererfahrung. Durch den Tipp meines Beratungslehrers Herrn Stein, stieß ich auf das speziell für den Programmiereinstieg konzipierte System „Netlogo“, dass auf der am M.I.T. entwickelten Sprache „Logo“ basiert.

Objekte:
Es gibt zwei Agententypen: „Patches“, die die Grundfläche der 2D-Simulation in einem Raster bedecken und „Turtles“, die sich zur weiteren Gruppenbildung in „Breeds“ einteilen lassen. Es besteht die Möglichkeit, Mitgliedern einer Agentengruppe Variablen zuzuordnen, die eine Sensorik in Bezug auf die Position und anderen Faktoren, aber auch eine Speicherung von Code als Erbgut erst ermöglichen.

Synchronisation
Jeder Befehl kann von einer Gruppe entweder parallel oder in einer vorgegebenen Reihenfolge der Agenten hintereinander ausgeführt werden, sodass ein einfaches Interagieren ermöglicht wird.

Visualisierung:
Beide Agententypen haben eine Farb-, Größen- und Label-Variable, was eine einfache Ansteuerung ermöglicht. Zusätzlich kann man die Form eines „Turtles“ durch die Variable „Shape“ ändern.

Speichertypen:
Variablen können in „Netlogo“ viele Speichertypen enthalten: Strings, Listen, Zahlen, Agenttypen, „Booleans“ usw… Einer der fundamentalen Bestandteile vieler Programmiersprachen, das Array, ist durch die Liste ersetzt, die sich durch eine Reihe von vorgefertigten Befehlen gut handhaben lässt.

5. Umsetzung

5.1 Testphase (siehe „Testversion.nlogo“)

Mein erster genetischer Algorithmus basierte im Prinzip auf einer (*m* + *l*)-Strategie, wobei ich die Selektion an den Anfang des Prozesses verschob um nur die „m“ besten Kombinationen zu vermehren und so die Effektivität zu steigern. Das Ziel meines Algorithmus ist das Finden der besten Lösung, der Populationsdurchschnitt spielt hierbei keine Rolle. Außerdem muss dieser Algorithmus eine monoton steigende Fitness, d.h. bei einer Entfernungs-Bewertung eine monoton fallende Kurve, erzeugen, weil die konstante Elternpopulation einen evoutiven Rückschritt verhindert.

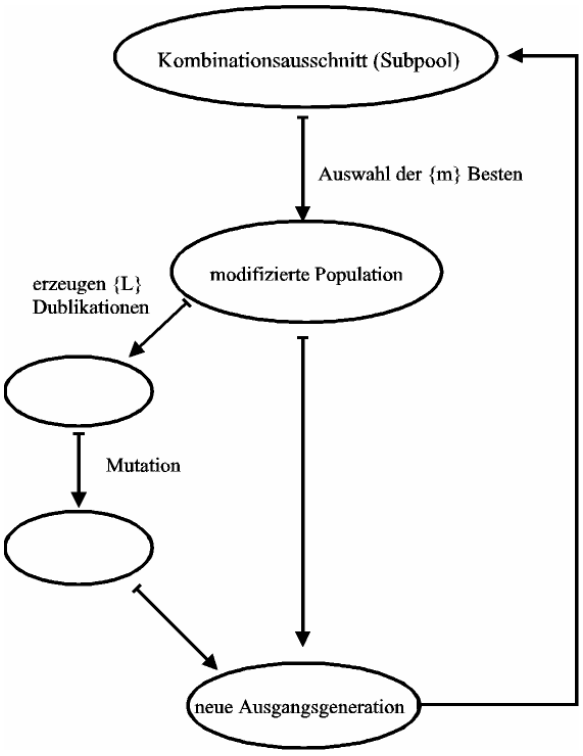

Abb. 2: Funktion meines Algorithmus

Zur Kalibrierung meines Algorithmus entwarf ich ein einfaches Programm, bei dem das Erbgut der Agenten durch eine Zahl symbolisiert und als Farbe angezeigt wird. Der vorhandene Subpool kann auf einen beliebigen Anteil des Meta-Kombinationspools (alle Farben auf einer Skala von 0 bis 139) beschränkt werden. Mit dem Regler „combo-limit“ lässt sich die Zusammensetzung des Subpools steuern.

Ein Selektor berechnet die Fitness aller {tAnzahl} Agenten über den Absolutwert der Differenz zwischen gewünschter Farbe und dem jeweiligen Erbgut {tDNA} und speichert ihn zusammen mit der Agenten-Kennnummer in eine Liste. Anschließend wird diese {Fitnessliste} in aufsteigender Form geordnet und die {m} besten Objekte zur Gründung der nachfolgenden Generation selektiert. {m} wird durch „Druck“ geregelt.

$$\frac{\{\text{tAnzahl}\}}{\{\text{m}\}} = \text{Selektionshärte}$$

Bei meinen ersten Versuchen hing die Duplikationszahl eines Elters von der Selektionshärte ab, sodass der Ausgangsbestand {tAnzahl} niemals überschritten wurde. Nach der Duplikation wird die Filialgeneration mutiert.

5.2 Vorversuche

Konstante Mutation:
Normalerweise wird die Mutationsschrittweite bei genetischen Algorithmen auf einen konstanten, empirisch gefundenen Optimalwert genormt.
Setzt man die Mutationsstufen meines Programms auf „constant“ treten modellbedingte Anomalien zwischen der Verschiebung des Mutationsfensters, der Zusammensetzung des Subpools und der Mutationsschrittweite auf:
Abhängig davon, ob beide Faktoren gerade sind, können nur bestimmte Zahlengruppen gefunden werden. Das Mutationsfenster verschiebt sich unkontrollierbar. Dabei ist die Unschärfe, die entsteht, wenn über das Ziel hinaus mutiert wird, gut zu erkennen.
Eine Bedingung wird nur durch eine Schrittweite von 1 zuverlässig erfüllt, was einem stupiden Abtasten aller Zahlen gleichkommt.

Anteilsmäßige Mutation
Danach war ich der Meinung, dass bei natürlicher Evolution die Mutation zwar immer eine konstante Rate am Erbgut beträgt, die Schrittweite jedoch mit der Länge des Erbgutstranges variiert. Mit der Mutationsregelung „dynamic“ kann diese Annahme simuliert werden.

Messungen mit verschiedenen Parametern zeigten jedoch, dass auch hier Anomalien entstehen, da mit der Größe des Erbgutes auch die Mutation und somit die Unschärfe zunimmt. Es wäre, im übertragenen Sinne, Organismen mit einem großen Erbgut nicht möglich sich durch feine Änderungen anzupassen, während ein kleines Erbgut in seiner Entwicklung stillstände.

5.3 Codekombinierender Algorithmus

Metakombinationspool:
Zur einfachen Handhabung habe ich einen minimalen Teil der Sprache Logo in die Kategorien Befehle, Bewegungen, Operationen, Variablen, Zahlen, „Booleans“, Operatoren, Konditionen und Logik-Symbole eingeteilt.

Objekteigenschaften:
Jeder Turtle bekam 5 Listen ({Genes}, {Pre-Structure}, {Structure}, {Pre-Matrix} ,{Matrix}) zugewiesen, die sein Genom repräsentieren und weitere zwei Sensorvariablen ({Port1}, {input1}), die zur Abfrage der Funktion seines Erbguts dienen.
Jeder Patch erhielt außerdem eine Variable {Radiation}, über die sich die Mutationsstärke an einem bestimmten Feldpunkt ändern lässt.

Syntaktisch sicheres System:
Im Vorfeld wird jeder Kombinationskategorie und jeder Erbgutliste ein Link in Form eines abzulesenden Strings zugewiesen und diese danach in Such- und Einsetz-Gruppen zusammengefasst.
Jeder Befehl wird nun nach diesem „Regelwerk“ ausgerichtet, sodass Bestandteile von Kategorien später nur dort eingesetzt werden können, wo sie keinen Syntaxfehler hervorrufen.

Codesynthese: *(s. Abb. 3)*
Jeder Agent der Population bekommt in mehreren Schritten ein einzigartiges Erbgut zugewiesen, dessen Bedeutung in absteigender Listenreihenfolge zunehmend präzisiert wird, d.h. bei jeder Stufe wird eine bestimmte Linkgruppe ausgetauscht.
An oberster Stelle, in den „Genen“, findet sich eine von der Prozedur „Codesynthese“ zufällig zusammengestellte Reihenfolge von Befehl- und Bewegungs-Links.
Neben Basiskomponenten des genetischen Algorithmus wie z.b. die Prozedur „Mutation“ habe ich einige Werkzeuge in Form von Reportern erstellt. Das Werkzeug „Analyse“ dient der Überprüfung von Listen auf eine Kriteriengruppe und liefert den passenden Pfad für einen „Codegenerator“, der den Link mit einem Bestandteil der jeweiligen Kombinationsgruppe ersetzt.
Bei jedem Syntheseschritt wird das „Memory“ geladen, analysiert, ersetzt und in der nächsten Stufe des Genoms zwischengespeichert. Im letzten Schritt wird die Liste „Pre-Matrix“, die keine Links mehr enthält, mit dem Werkzeug „Cutout-Brackets“ in einen für Turtles abspielbaren Code kompiliert, der in „Matrix“ gespeichert wird.

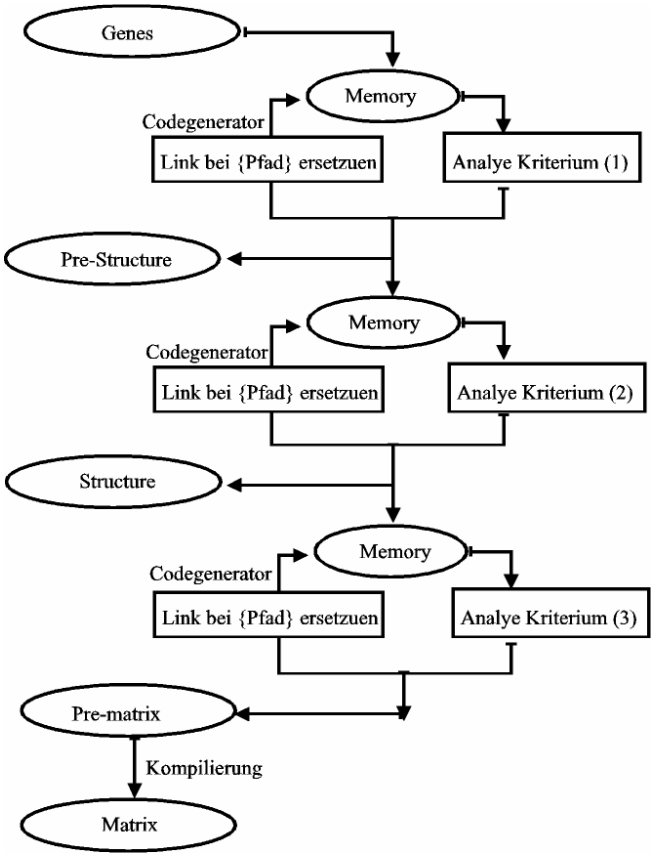

Abb. 3: Linkaustausch bei Codesynthese

Mutation:
Im Grunde ist eine Mutation eine Neupräzisierung eines zufällig gewählten Links in einer zufällig gewählten Liste. Die Aufteilung des Genoms ist sinnvoll, weil sich nur so genügend Ansatzpunkte für Mutationen finden lassen: Je öfter zwischengespeichert wird, desto feiner kann der Code mutieren, allerdings nimmt auch die Wahrscheinlichkeit auf Mutation eines Links mit zunehmender Auffächerung des Genoms ab.

<u>Maximal-/Mittelwert-Steuerung:</u>
Bei meinem Algorithmus musste ich vom Prinzip der definierten Mutationsschrittweite [1] abweichen, da die Linkstruktur keine Größe aufweist, die sich als Mutationseinheit definieren lässt, d.h. die Funktion des Codes lässt sich nicht durch die Summierung gleich großer Mutationen optimieren, wie es in der Natur der Fall ist, da jeder Link-Typ eine andere maximale Zeichenlänge in der Matrix besitzt.

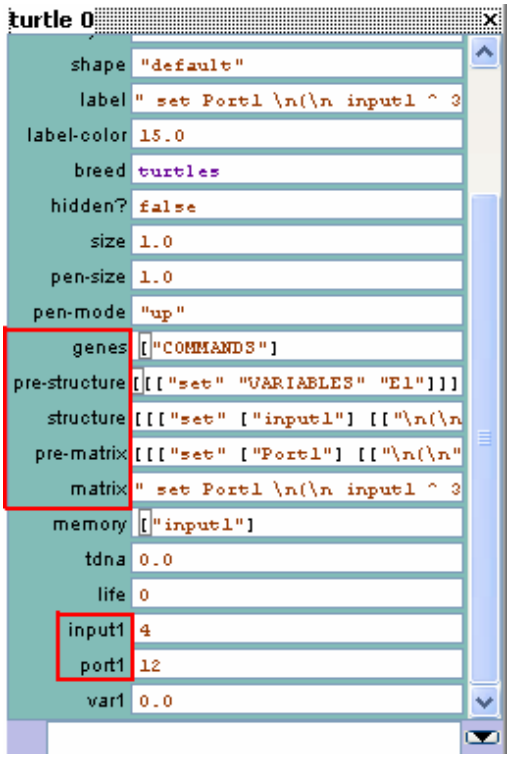

Abb. 4: Linkstruktur des Genoms

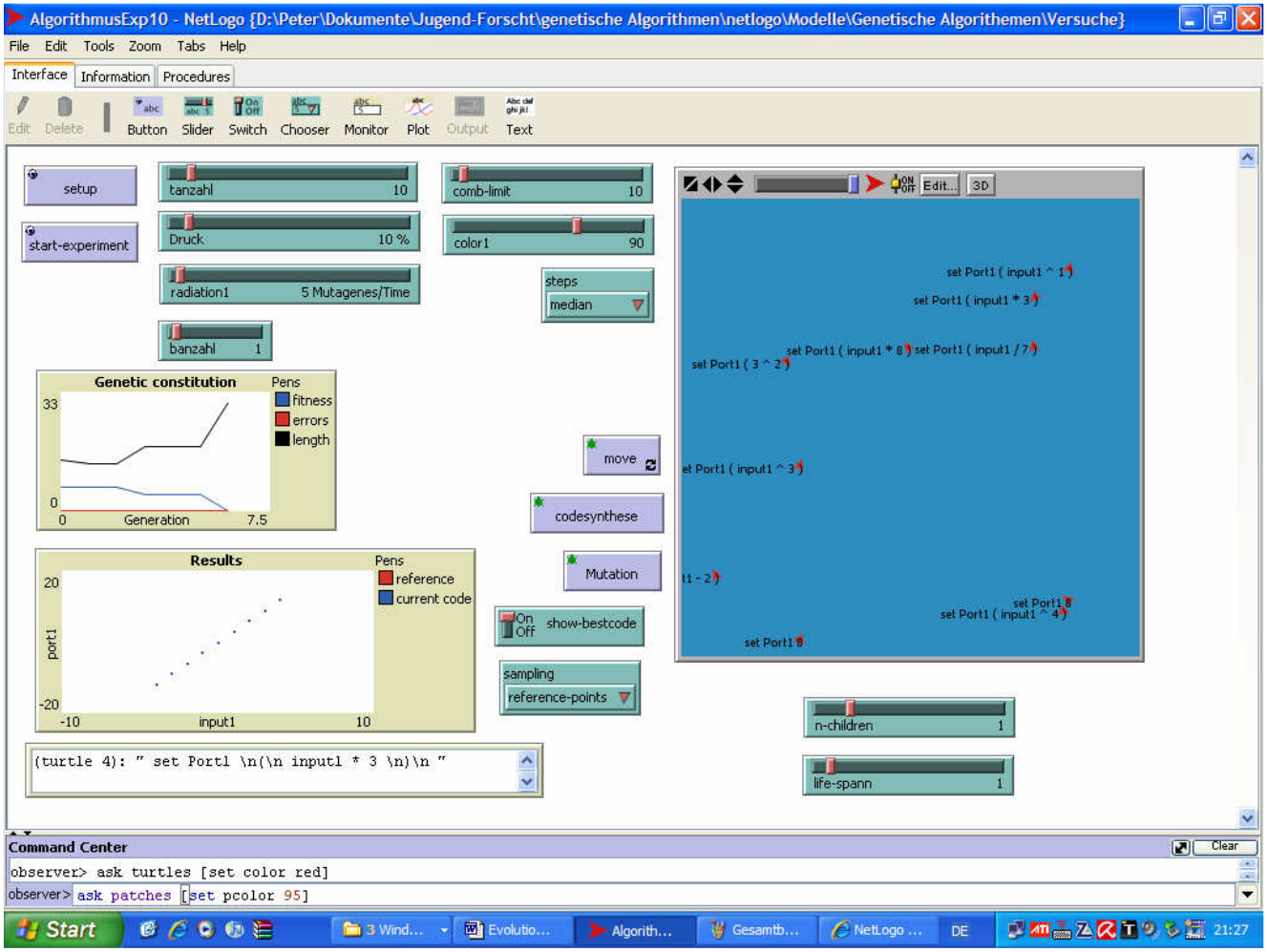

*Abb. 5:Oberfläche meines Programms: mit gefundener y= 3*x Funktion und Vorversuchreglern (rechts)*

Weil sowohl kleine als auch große Mutationen nötig sind, damit der Code das Optimum erreichen kann und da auch hier ein Zwang zur Mutation eine Unschärfe zur Folge hätte, bleibt nur die Bestimmung von *Anzahl und Art* der Mutationen durch Zufall.
Die „median"-Regelung unterscheidet sich von der „Max"- Regelung durch eine Steuerung der durchschnittlichen Mutationsanzahl. In einem Testlauf mit meinem einfachen Erprobungsprogramm zeigte sich, dass beide Ansätze leistungsfähiger als eine herkömmliche Schrittweiteneinstellung waren, da sie zuverlässig und sehr viel schneller eine passende Kombination fanden. Nach Abschluss der Versuchsreihe übertrug ich die Steuerung in meinen code-kombinierenden Algorithmus.

<u>Bildung mathematischen Formeln durch Anpassung an Koordinaten</u>
Da jedes Gen als ein Prozess mit In- und Output angesehen werden kann, liegt es nahe, diesen Prozess als algebraische Funktion darzustellen. Jede Kombination wird über die Turtelvariablen {input1} und {port1}, stellvertretend für x und f(x), angesteuert.

Selektionsbedingungen:
Die Gesamtfitness eines Agenten ist in 3 Kategorien mit absteigender Wichtigkeit unterteilt:

<u>Funktionswerte:</u>
Die eingegebenen kartesischen Koordinaten dienen als Referenzpunkte zur Berechnung der Fitness: Die Wertepaare der Algebra-Kombination werden mit den Referenzkoordinaten verglichen und ihre Abweichung gespeichert. Der Durchschnitt aller Abweichung einer Funktion wird berechnet und als erster Fitnesswert angegeben.

Errors:
Obwohl das System zur Sicherung der Syntax gewisse Spielregeln setzt, kann es trotzdem zu einer 0-Division oder zu Zahlen kommen, die Netlogo nicht verarbeiten kann. Aus diesem Grund wird die Anzahl der Fehler beim Ausführen des Codes als Beiwert hinzugefügt.

Länge der Formel:
Ohne eine Selektion nach der Formellänge könnte der Algorithmus passende aber unnötig komplexe Lösungen erzeugen. Die Länge der Matrix ist das letzte Kriterium mit der geringsten Bedeutung.

Selektion:
Wie auch bei meinen Vorversuchen wird die Gesamtfitness jedes Turtles in eine „Fittnessliste“ übertragen. Die anschließende Ordnung nach 3 Kriterien läuft über die Bildung einer Kommazahl durch Zusammenfügung der Fitnesskategorien und anschließender Ordnung. Kombinationen, die voraussichtlich zu große Zahlen bei der Ordnung erzeugen könnten, werden ausselektiert, d.h. der Turtle wird gelöscht.

Zu einer Koordinatenreihe gibt es viele perfekt zu den Referenzpunkten passende Funktionen, die jedoch in nicht überprüften Bereichen einen gänzlich anderen Verlauf darlegen, als es sich der Tester vorstellt. Für das Finden einer Kurve ist die eingegebene Informationsmenge, d.h. die Anzahl der Punkte, entscheidend. Beispielsweise kann der Algorithmus bei den Koordinaten (1 1), (0 0), (-1 -1) sowohl eine y=x als auch eine x^x-Funktion finden.
Bei der Visualisierung der aktuell besten Kombination kann man deshalb zwischen Referenzpunkt- und Rasteranzeige wählen.

Konstanten finden:
Über die Variable „Color1“ lässt sich die Größe eines einzelnen Referenzwertes bei einem Input von Null definieren. Es ist deutlich zu erkennen, wie sich der Wert der aktuell besten Formel der Referenzkoordinate annähert und sie schließlich erreicht.

Gerade Finden:
Gibt man genügend Referenzpunkte ein, so wird eine y=x- Gerade binnen zwei Generationen gefunden. Die hohe Geschwindigkeit lässt sich durch eine geringe Anzahl an Einsetzungsmöglichkeiten für Variablen erklären.

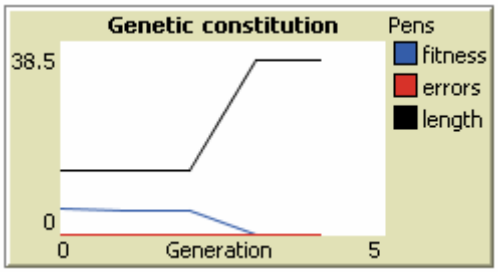

Abb. 5: Fitnesskurve bei Parabel

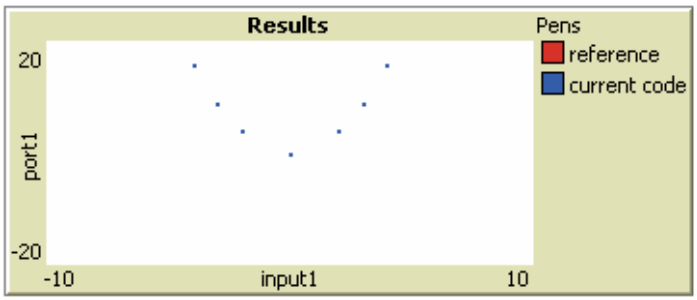

Abb. 6: mit Ref.-Punkten übereinstimmende y=x^2 Funktion

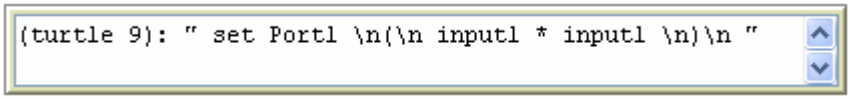

Abb. 7:Kompilierter Code der besten Kombination

6. Kritische Reflexion

6.1 Mutationsprobleme

Bei allen getesteten Funktionstypen bemerkte ich folgende Anomalie: Die Werte der aktuell besten Funktion näherten sich zwar an, kamen aber vor dem Optimum zum Stillstand. Nachdem ich einen Fehler in der Kommaordnung behoben hatte schien das Problem kurzfristig verschwunden zu sein, als ich später zur Optimierung mehr Koordinaten eingab tauchte der Effekt jedoch wieder auf.

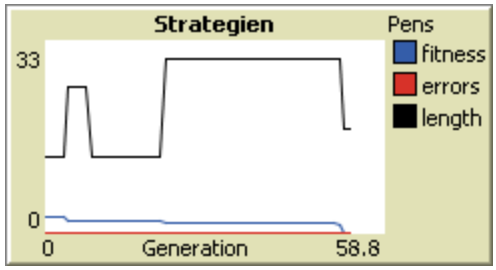

Abb. 7: Effekt einer zu harten Selektion

Beim Prüfen einer Funktion auf unterschiedlich viele Referenzwerte derselben Geraden stellte ich fest, dass die Abweichung von unpassenden Funktionen mit der Zahl der Referenzwerte zunimmt und somit die Evolution härter wird. Offenbar existiert bei der Selektion ein Fenster zwischen unzureichender Information und zu harter Aussonderung.

<u>Finden einer y=3*x Geraden</u>
9 Koordinaten: [-1 -3] [-2 -6] [-3 -9] [-4 -12] [0 0] [1 3] [2 6] [3 9] [4 12]
Abweichung der normalen y=x-Funktion: 4,4

10 Koordinaten: [-1 -3] [-2 -6] [-3 -9] [-4 -12] [0 0] [1 3] [2 6] [3 9] [4 12] [5 15]
Abweichung der normalen y=x-Funktion: 5

6.2 Bedeutung stummer Mutationen

Bei weiteren Beobachtungen stellte ich fest, dass auch bei der richtigen Informationsmenge die meisten neu geschaffenen Kombinationen ausselektiert werden, obwohl sie syntaktisch gesehen näher an der Optimallösung lägen als die zurückgebliebenen Funktionen, welche meistens einfacher Art waren. Das Problem liegt im syntax-sichernden System meines genetischen Algorithmus: Es sind keine „stummen“ Mutationen möglich, die im Hintergrund den Aufbau einer Formel, jedoch nicht deren Wertepaare verändern, weil jede Neudefinition eines Links Folgen hat.

<u>Neuer Algorithmus:</u>
Um die Selektion weiter zu entschärfen reduzierte ich meinen Algorithmus auf eine normale (m + l)- Strategie, ohne Vorselektion, sodass sich alle Objekte des Pools vermehren und die Filialgeneration anschließend mutieren kann. Durch eine schwächere Auslese ließen sich die Chancen für syntaktische Änderungen zwar steigern, blieben jedoch systembedingt niedrig.

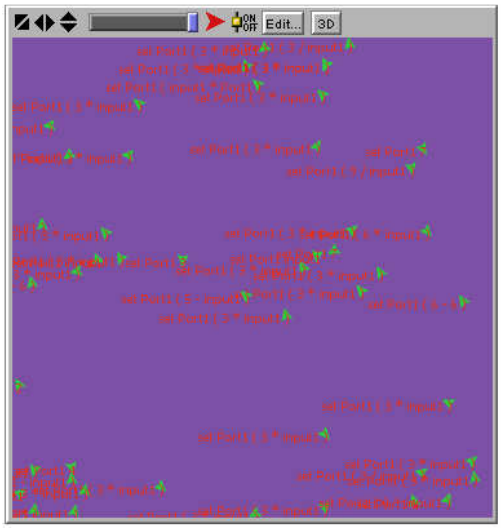

Abb. 8: Die parallele Kombinations-Leistung der Agenten nimmt mit ihrer Anzahl zu.

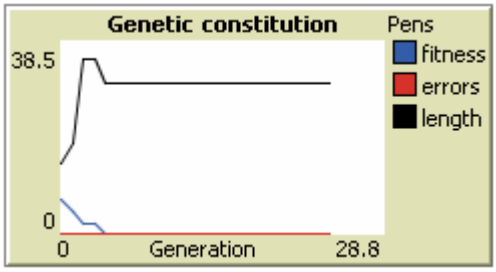

Abb. 9: mit abgemilderter Selektionshärte

Metakombinationspool:
Eine Freigabe und sinnvolle Nutzung der kompletten Programmiersprache ist fast unmöglich, weil mit jedem neuen Befehl die Anzahl möglicher Kombinationen drastisch erhöht und die Wahrscheinlichkeit auf eine besser geeignete Kombination schnell sehr klein wird. Aus diesem Grund findet mein aktueller genetischer Algorithmus nur simple Funktionen mit einer Operation und kann keine negativen Zahlen einsetzen.

Rechenkapazität:
Für einen höhere parallele Kombinationsleistung werden größere Populationen und damit mehr Zeit und Rechenleistung benötigt. Bereits bei verhältnismäßig wenigen Agenten wird der Prozessor stark ausgelastet. Der Algorithmus läuft zuverlässig ab, wenn auch sehr langsam.

6.3 Evolutionäres Optimierungsprogramm (siehe EA-Optimierung_Fur.nlogo)

Da auf dem Gebiet der Codeevolution noch viele Änderungen nötig waren suchte ich eine andere Anwendungsmöglichkeit. Ich entwickelte ein Optimierungsprogramm für Netlogo-Anwendungen.
Jede Simulation ist abhängig von ihren Eingangswerten und stellt oft nichtlineare Effekte dar, d.h. sie ist das Ergebnis eines Zusammenwirkens von verschiedenen Faktoren. Je nach Anzahl und Toleranzbereich der Variablen sind schnell viele Kombinationen möglich:

4 Variablen zu jeweils 100 Möglichkeiten = 100 Mrd. Kombinationen

Bisherige Optimierungsprogramme, wie „Behaviorspace“ von Netlogo rechnen alle Möglichkeiten durch, wohingegen ein evolutionär programmiertes Optimierungsprogramm nur die auf der Evolutionsleiter liegenden Kombinationen passiert, d.h. nur jene rechnet, die sich im Zuge der Mutation und der steigenden Fitness der Population ergeben.

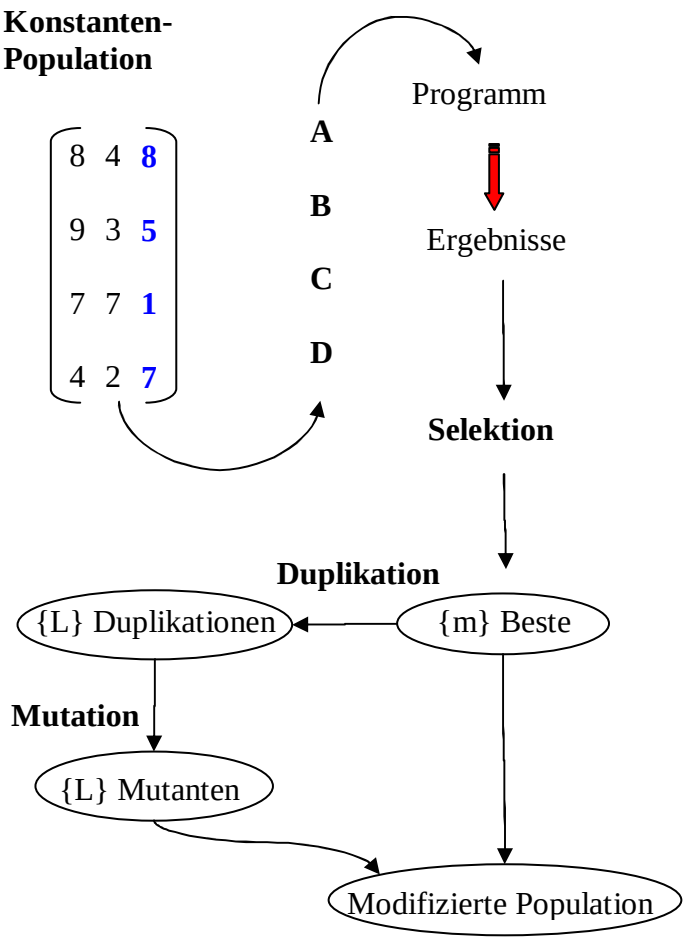

Abb. 10: Optimierungssystem

Hierbei wird eine Ausgangspopulation von Konstantengruppen erstellt. Jede Gruppe (Spalte) wird in das Programm eingespeist und die Resultate nach den Selektionskriterien aufgezeichnet. Es folgt das bekannte Schema meiner Evolutionsstrategie.
Der Vorteil des Algorithmus ist seine modellübergreifende Anwendbarkeit. Ich habe ihn weiter abstrahiert und auf das Kristallbildungsmodell von Netlogo angewendet. Die Datenaufnahme bezieht die 3 Größen der EAs: Kombinationspool, Selektionskriterien und Mutationssystem.

Nur die in der Liste {m_constants} vertretenen Variablen werden in einem definierten Toleranzbereich mit der empirisch bestätigten Medianmutation mutiert. Als Kriterium gab ich vor, einheitliche Strukturen zu bilden, d.h. es sollen möglichst viele gleichartige Partikel nach der durchlaufenen Zeit nebeneinander liegen.
Da das Kristallisationsprogramm zumindest teilweise vom Zufall abhängt, müssen mehrere Durchläufe gemacht werden, bis das Ergebnis einen repräsentativen Mittelwert zeigt.
Außerdem müssen alle Algorithmen, die zum Ablauf der Simulation notwendig sind, in das EA-Paket integriert werden.
Je nach Anzahl der Mutationsmöglichkeiten einer Variable müssen verschieden große Mutationsbasiseinheiten existieren, d.h. die Einheit muss in Relation zum Toleranzbereich stehen. Entweder werden variabelenspezifische Mutationsbasiseinheiten erstellt, oder die *maximale* Feinheit der Mutation wird grundsätzlich auf einen passenden Wert gesetzt.

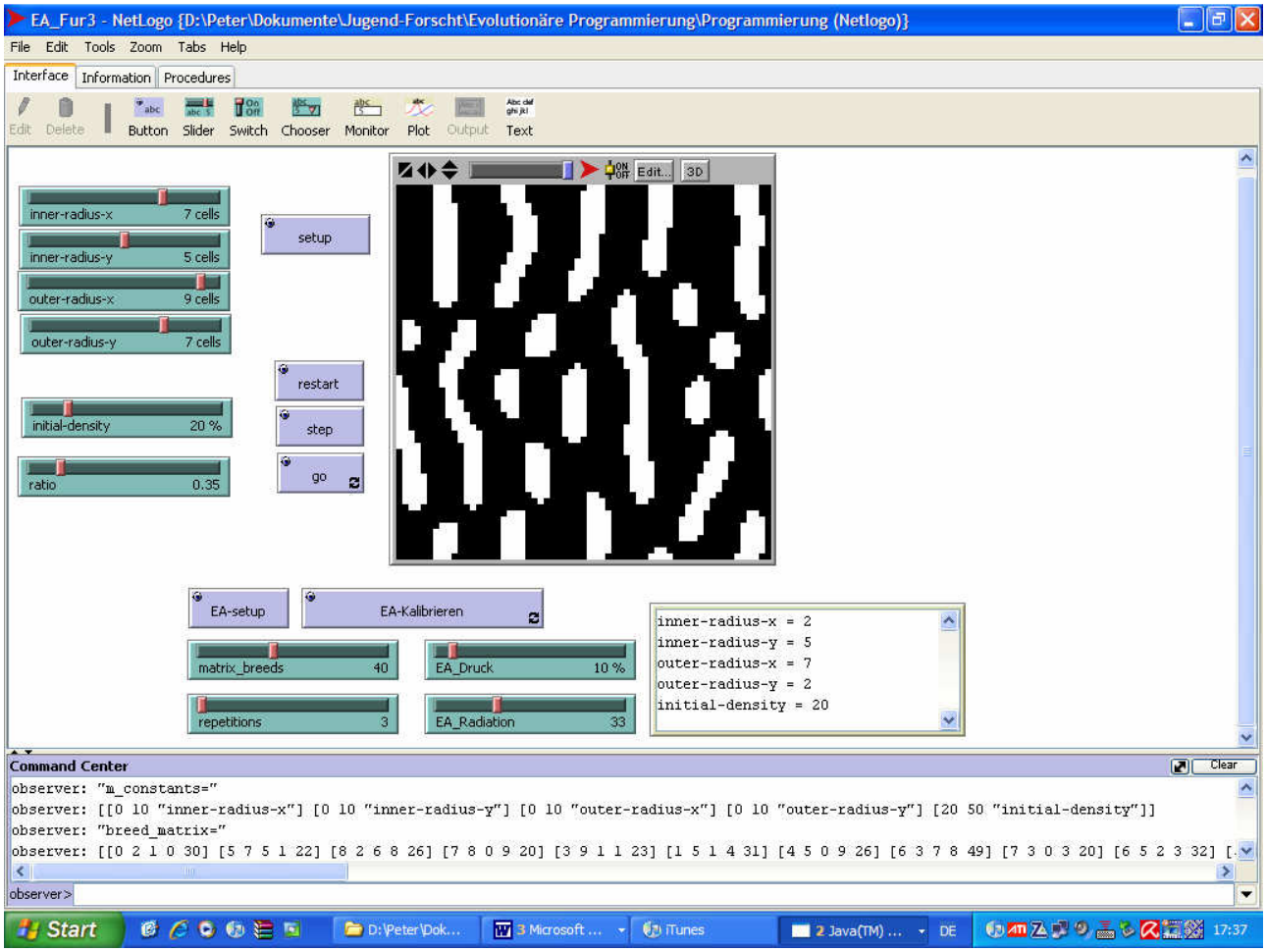

Abb. 11: Eine mittels evolutionärer Strategie gefundene Variabelenkonfiguration erzeugt eine starke Kristallisation. Unten: Adaptiver Optimierungsteil mit bester Einstellung und Konstantenpopulation

Nach dem Start der Simulation lässt sich eine Konstantenpopulation generieren und der EA starten. Relativ schnell entwickelt sich eine Konstantengruppe mit einem „initial density“-Wert von 20-22 heraus. Die anderen m-constants schwanken, auch wenn sie zeitweise gleich bleiben und so Evolutionsplateaus bilden, auf denen die Änderungen kaum die Fitness beeinflussen. Sie können durch scheinbar besser angepasste Einstellungen verdrängt werden, die jedoch bei weiterer Mutation komplett andere Werte aufweisen, wodurch das Plateau zusammenbricht und sich auf einer neuen Stufe aufbaut.

6.4 Fazit

Mein erstes Ziel war ein genetischer Algorithmus, der selbstständig Programme erzeugt, die an eine vorgegebene Bedingung angepasst sind. Dieses Ziel habe ich erreicht, auch wenn das Finden von mathematischen Formeln noch Probleme bereitet. Durch die Experimente konnte ich einige Fehler in meiner Programmierung erkennen, die mit dem Verständnis der Evolution selbst verbunden sind. Durch das Projekt habe ich einige Erkenntnisse gewonnen, die ich für Weiterentwicklungen nutzen kann. (Hier ist vor allem die Fähigkeit der drei EA-Grundbausteine, angepasste Lösungen zu erzeugen, von großer Bedeutung).

7. Ausblick

Verbesserung meines Algorithmus

Bei weiterer Beschäftigung mit einem evolutionären Algorithmus könnte ich das Syntaxsystem verbessern und möglicherweise eine Funktion zur Selbstkalibrierung einbauen, wie sie auch schon von Rechenberg angewandt wurde. Die Leistungsfähigkeit meines Algorithmus ließe sich außerdem durch eine Evolutionsstrategie mit zusätzlichem Chrossover steigern.

Multiselektive Evolution

Mein weiter entferntes Ziel bleibt nach wie vor die Simulation einer ungerichtete Evolution von lebenden Datenstrukturen in einer Informationsebene.
Grundkomponenten dieser erweiterten Evolutionssimulation wären asynchron interagierende Objekte, sowie eine Kopplung von Erbgut und Einflüssen, ausgedrückt durch ein Variabelenfeld, nach dem Schema des „Operon-Modells“ zur Genregulation bei Bakterien. Ziel ist ein System, das die Kombination durch eine Art Selbstorganisation erzeugt und modifiziert.

Anwendungen:
Eine gelenkte, multiselektive Evolution, könnte zur gesteuerten Ausbreitung von Software in einer Hardwarearchitektur genutzt werden. Im Allgemeinen ist eine künstliche Evolution vielfältig von Vorteil, da sie, richtig gesteuert, ein großes Planungspotenzial beinhaltet.

7.1 Hypothesenbildung

Hypothese der Mutationsanpassung:

Ich vermute, dass der vom Erbgut codierte stoffliche Aufbau eines Lebewesens direkt mit der Mutationsschrittweite und mit der Findung des Mutationsfensters zusammenhängt. Im Rahmen der Evolution würde sich eine Population an ihre mutagene Umgebung anpassen, indem sie den Aufbau ihrer Lebensformen verändert, sodass auch bei einer veränderten Strahlungsgröße das Mutationsfenster getroffen wird.
Dieser Ansatz lässt sich auch zur automatischen Kalibrierung der Mutation von genetischen Algorithmen verwenden, wenn man die Mutationseinheit als veränderliche angibt. Folglich hätte nur die optimale Einstellung einen Selektionsvorteil.

Hypothese der interdisziplinären Gültigkeit der Evolutionstheorie

Betrachtet man den Markt eines Landes, lassen sich menschliche Individuen oder Unternehmen als Wirtschaftssubjekte definieren. Eine Marktwirtschaft im Hinblick auf eine Nation ist ein offenes System: Waren-, Finanz- und Menschenströme „fließen“ in das System

hinein bzw. aus ihm heraus, da es in den globalen Markt als „Nationalsubjekt“ eingebunden ist.
Sowohl Wirtschafts- als auch Nationalsubjekte konkurrieren um eine Nachfrage, da ihre Existenz von einem Fließgleichgewicht, sei es nun finanzieller, materieller oder menschlicher Art, abhängt. Unternehmen können nur existieren, weil sie im Idealfall den Differenzbetrag zwischen gekauften Rohmaterialien und verkauften Produkten zur Selbsterhaltung oder Expansion nutzen. Nationen haben in diesem Zusammenhang ein spezifisches Import/Export Muster, dass aus ihrer Wirtschaftsstruktur resultiert.
Der Prozess „Globalisierung“ ist im Hinblick auf die Ökonomie, eine Konkurrenz von Unternehmensgruppen (Nationen) untereinander am Weltmarkt, wobei jede Gruppe für sich ebenfalls ein Marktsystem darstellt dessen Mitglieder untereinander und mit Subjekten des internationalen Marktes konkurrieren. Die spezifische Eigenschaft eines Wirtschaftssubjektes, der Prozess, den es ausführt, kann als seine Kombination angesehen werden.

Fragestellung:
Lässt sich die Evolutionstheorie abstrahieren und interdisziplinär anwenden?

Analogienbildung zwischen Marktwirtschaft und Evolution
Zwischen dem Modell der Marktwirtschaft und der biologischen Evolution gibt es viele Parallelen:
-Die Summe aller möglicher Lebewesen/ Wirtschaftssubjekte stellt einen Kombinationspool dar.
-Ein Lebewesen/Wirtschaftssubjekt ist eine einzigartige Kombination von Eigenschaften
-Umwelt/ Markt haben die Funktion eines Selektors und formen den lokalen Kombinationspool
-Lebewesen/Unternehmen/Nationen gründen ihre Existenz auf ein Fließgleichgewicht
-In Natur und Wirtschaft führt die Selektion zu Monopolen/Spezialisierung/Nischennutzung

Weitere Abstraktion
➔Nach weiterer Recherche konnte ich meinen Ansatz erweitern und ihn unter dem Begriff “Selbstorganisation“ zusammenfassen. Versteht man die Evolution als ungerichtete Entwicklung einer Summe von vernetzten Systemen, lassen sich grundlegenden Strukturen erkennen: Alle Objekte stehen miteinander über Regelkreise in Verbindung. Dabei existiert eine ständige Wechselwirkung durch Schwingungen eines Variablen-Feldes, das die Eigenschaften der Realität symbolisiert.
Die Schwingung eines Stoffes beispielsweise wirkt auf eine Zelle als Input. Was die Zelle daraus letztendlich erzeugt, wie ihr Stoffwechsel reagiert, wird durch das Führungsglied ihrer Regelkreise, ihrer DNA bestimmt. Dieses Führungsglied ist, wenn auch in anderer Form, für jede Selbstorganisation grundlegend, denn es bestimmt die spätere Strukturbildung durch Wechselwirkung. Man kann beispielsweise die Eigenschaften eines Atoms als Führungsglied seiner Schwingungsverarbeitung ansehen. Abschließend wird der ein „Energie-system“ zugewiesen, sodass die Schwingung eines Stoffes über die Energie die Schwingung eines anderen beeinflussen kann. Im Grunde ist dieses System in den Regeln der Führungsglieder vorhanden.

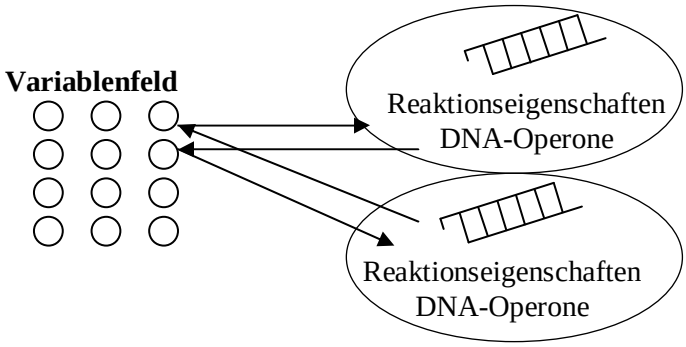

Abb.12: Wechselwirkung zur Selbstorganisation

8. Literatur- und Linkverzeichnis

Biologisches Grundwissen:
Genetik, Grüne Reihe, Materialien SII, Schroedel-Verlag, Braunschweig, 2004

Evolutionsstrategien:
[1] Ingo Rechenberg: Evolutionsstrategie '94. Stuttgart: Frommann-Holzboog 1994.

http://www.bionik.tu-berlin.de/
http://de.wikipedia.org/wiki/Evolutionsstrategie

Genetische Algorithmen:
http://de.wikipedia.org/wiki/Evolution%C3%A4rer_Algorithmus
http://www.evocomp.de/themen/evolutionsstrategien_vs_genetische_algorithmen/evolutionsstrategie_vs_genetischer_algorithmus.html
http://www.evocomp.de/themen/genetische_algorithmen/genalg.html#Codierung