TECHNISCHE UNIVERSITÄT DORTMUND
FAKULTÄT FÜR INFORMATIK

Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
an der Fakultät für Informatik
der Technischen Universität Dortmund

Genetische Programmierung einer algorithmischen Chemie

von
Christian W. G. Lasarczyk

Dortmund, 2007

Tag der mündlichen Prüfung
12. Dezember 2007

Dekan
Prof. Dr. Peter Buchholz

Gutachter
Prof. Dr. Wolfgang Banzhaf
Prof. Dr. Günter Rudolph

Andrea und meinen Eltern

Danksagung

Mein Dank gilt all jenen Menschen, die meine Arbeit und die dafür benötigte Zeit, bereichert haben. Zunächst bedanke ich mich bei Prof. Dr. Wolfgang Banzhaf für die Betreuung dieser Arbeit und seine Einladung an die Memorial University of Newfoundland (Kanada). Den Kollegen, der Leitung und der Verwaltung des Lehrstuhls für Algorithm Engineering (ehemals Systemanalyse) des Fachbereichs Informatik der Universität Dortmund danke ich für die großartige Arbeitsatmosphäre in den vergangen Jahren. Ich danke Prof. Dr. Thomas Bartz-Beielstein für die intensiven Diskussionen zur experimentellen Betrachtung von Algorithmen. Ihm, Dr. Udo Feldkamp und Mike Preuß danke ich für die Kommentierung meiner Arbeit. Neben den bereits namentlich genannten haben Karlheinz Schmitt und Boris Naujoks durch viele Diskussionen über die naturanalogen Verfahren mein Bild von diesen Verfahren und damit meine Arbeit geprägt.

Besonderer Dank gebührt darüber hinaus meinen Eltern für ihre Unterstützung und meiner Ehefrau Andrea Mannefeld, die ohne viel von mir zu haben immer für mich da war.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

Metaheuristiken aus dem Bereich der evolutionären Algorithmen suchen nach Lösungen unterschiedlicher Problemstellungen mittels Mechanismen, die der natürlichen Evolution entlehnt sind. Potentielle Problemlösungen werden dabei in Individuen repräsentiert und gehen in einem Prozess künstlicher Evolution durch Operationen wie Mutation oder Rekombination auseinander hervor. Eine Selektion guter Individuen, aus welchen mit den zuvor genannten Operationen Nachkommen abgeleitet werden, sorgt dabei für eine zielgerichtete Entwicklung.

In der genetischen Programmierung beschreiben die Individuen ein Modell, welches Elemente in Relation zueinander setzt. Üblicherweise handelt es sich bei diesem Modell um eine, als Computerprogramm interpretierbare, Struktur, die aus einer Menge von Eingaben die gewünschten Ausgaben erzeugt. Bekannte Eingabe-Ausgabe-Relationen dienen dabei als Grundlage für die Bewertung und Selektion der Individuen. Die Hoffnung ist, dass das Modell die Ein- und Ausgabe über Gesetzmäßigkeiten in Beziehung setzt, die auch jenseits der verwendeten Daten ihre Gültigkeit haben. Damit zählt die genetische Programmierung zu den Verfahren des maschinellen Lernens.

Der genetischen Programmierung liegt zumeist die Annahmen zugrunde, dass die Individuen eine evolvierte, wohldefinierte Struktur haben und ihre Ausführung deterministisch erfolgt. Diese Annahmen haben ihren Ursprung nicht beim methodischen Vorbild, der natürlichen Evolution, sondern sind ein bewusstes oder unbewusstes Erbe der Umgebung, in der die Evolution nachgebildet wird — der von-Neumann-Architektur.

John von Neumann hat mit der nach ihm benannten von-Neumann-Architektur weit mehr in der Informatik beeinflusst als das Gebiet der Rechnerarchitekturen. Angefangen bei den Programmiersprachen („Can programming be liberated from the von Neumann style?“, John Backus [6]) erreicht der Einfluss von von Neumann auch eine höhere Abstraktionsebene, das in der Praxis vorherrschende Verständnis von Algorithmen. Die Vorstellung von einer Kochrezept ähnlichen Beschreibung der Transformation einer Eingabe in eine Ausgabe durch die sequentielle Aneinanderreihung von Berechnungsschritten erschwert die Vereinbarkeit mit Aspekten jenseits der von-Neumann-Architektur, wie z.B. der räumlichen Struktur eines Algorithmus. Daher ist der Einfluss von von-Neumann auf die Evolution von Algorithmen durch die genetische Programmierung in letzter Instanz nicht verwunderlich, auch wenn sein Konzept wenig gemein mit den in der Natur evolvierten Systemen hat. Kapitel 1 führt zunächst in die genetische Programmierung und die unterschiedlichen Repräsentationen von Individuen ein.

In den letzten Jahren ist eine ganze Reihe von Konzepten und theoretischen

Modellen entstanden, die nur noch wenig Anleihen bei von Neumanns Rechnerarchitektur machen. Kapitel 3 beschreibt einige dieser Ansätze. Dabei sind jene von besonderem Interesse, deren (asynchron) parallel agierende Komponenten sich nichtdeterministisch verhalten und dezentral organisiert sind. Mit diesen Eigenschaften ähneln sie natürlichen Systemen stärker als klassische von-Neumann-Rechner.

Die Fähigkeit dieser Systeme, Berechnungen durchzuführen, entsteht erst durch die Interaktion ihrer Komponenten. Die Fähigkeit emergiert. Die Erforschung von Emergenz nimmt in der Artificial-Life-Forschung einen zentralen Stellenwert ein. Dieses Forschungsgebiet beschäftigt sich mit künstlichen Systemen, die Eigenschaften natürlicher Systeme aufweisen, wie etwa Fortpflanzung oder Reizreaktion. Die künstliche Chemie ist ein Teilgebiet und betrachtet Prozesse auf molekularer Ebene. Hier werden Systeme unter anderem so konstruiert, dass in ihnen das Ergebnis einer Berechnung emergiert. Kapitel 2 führt in die wichtigsten Elemente der künstlichen Chemie ein und stellt einige Systeme vor.

Aufgrund der Nähe der genetischen Programmierung zur von-Neumann-Architektur weiß man noch vergleichsweise wenig über die Evolution von Algorithmen, die in Form künstlicher Chemien kodiert sind, oder allgemeiner: über die Evolution von Algorithmen für Systeme jenseits der von-Neumann-Architektur.

Die vorliegende Arbeit nimmt sich dieser Fragestellung an und bedient sich hierfür der algorithmischen Chemie, einer künstlichen Chemie, die bei vereinfachter Betrachtungsweise aus einem veränderten Programmzeigerverhalten in der von-Neumann-Architektur resultiert. Reaktionen, eine Variante einfacher Instruktionen, werden hierbei in zufälliger Reihenfolge gezogen und ausgeführt. Sie interagieren miteinander, indem sie Produkte anderer Reaktionen verwenden und das Ergebnis ihrer Transformation, gespeichert in sogenannten Molekülen, anderen Reaktionen zur Verfügung stellen.

Der zweite Teil motiviert in Kapitel 4 zunächst die algorithmische Chemie und beschreibt sie anschließend im Detail. Die für ein zielgerichtetes Verhalten notwendigen Reaktionsschemata sollen später mittels genetischer Programmierung erlernt werden. Die Evolution einer algorithmischen Chemie wird mit den hierfür benötigten Operationen in Kapitel 5 dargestellt. Im Kapitel 6 werden Maße für einen systemübergreifenden Vergleich der evolvierten Lösungen und zur Bewertung der Ordnung evolvierter Chemien beschrieben.

Neben dem generellen Interesse an der Evolutionsfähigkeit solcher Systeme soll die Evolution algorithmischer Chemien mit einem etablierten Verfahren des maschinellen Lernens verglichen werden. Wie oben bereits angedeutet, wird die Ausführung einer algorithmischen Chemie durch einer Veränderung im Programmzeiger aus der Ausführung von Programmen auf der von-Neumann-Architektur abgeleitet. Dieses erlaubt die Evolution linearer Programme und algorithmischer Chemien weitestgehend identisch zu gestalten. Die bis einschließlich Kapitel 10 verwendeten Systeme unterscheiden sich dementsprechend nur im simulierten Verhalten des Programmzeigers. Die hohe Ähnlichkeit der Systeme erleichtert es, Abweichungen im Verhalten auf diesen Unterschied in der Umsetzung zurückzu-

führen.

Auch wenn für beide Systeme der gleiche Rahmen für die Evolution verwendet wird, so gilt es dennoch für beide Varianten eine adäquate Parametrisierung des Algorithmus zu finden. Da trotz des geringen Unterschieds das Verhalten beider Systeme grundverschieden ist, darf per se nicht von einer identischen Parametrisierung ausgegangen werden. Für einen fairen Vergleich hat die Parameterfindung weitgehend unabhängig voneinander mit vergleichbarem Aufwand zu erfolgen. Der dritte Teil der vorliegenden Arbeit widmet sich dieser Aufgabe.

Zur Bestimmung der Parameter findet die sequentielle Parameteroptimierung Verwendung. Sie wird hier um ein Verfahren zur Erhöhung der Wahrscheinlichkeit für die Bestimmung des besten Designpunkts aus allen experimentell betrachteten Designpunkten erweitert. Sowohl das Verfahren als auch die Erweiterung werden in Kapitel 7 vorgestellt.

Für die betrachteten Problemstellungen werden anschließend in Kapitel 8 angemessene Algorithmendesigns ermittelt. Für die Bewertung der Evolution algorithmischer Chemien und dem Vergleich mit der linearen genetischen Programmierung werden drei Problemstellungen herangezogen, die unterschiedliche Aufgabenfelder abdecken. Es handelt sich zudem um typische Problemstellungen in der genetischen Programmierung, welches neben dem hier vorgenommenen Vergleich auch den Vergleich mit anderen Systemen aus der Literatur ermöglicht.

Im vierten Teil werden die beiden Systeme hierauf aufbauend, d.h. mit der entsprechenden Parametrisierung, empirisch untersucht. Im Kapitel 9 wird zunächst der Evolutionsverlauf betrachtet. Neben der Verteilung und dem Verlauf der Individuengüte während der Evolution werden auch die für die algorithmische Chemie entwickelten Ordnungsmaße im Zeitverlauf betrachtet. Die Struktur der Individuen und damit der evolvierten Problemlösungen sind dann im Kapitel 10 das Objekt der weiteren empirischen Betrachtung. Den Schwerpunkt bildet hier die Tendenz zum Bloat und damit zur Anhäufung von Informationen, welche keine direkten Beitrag zur Fitness des Individuums leisten. Es zeigt sich, dass die in einer algorithmischen Chemie kodierten Lösungen deutlich kompakter sind.

Die sowohl bei der Parameteroptimierung als auch bei der empirischen Analyse gewonnenen Erkenntnisse über die Evolution einer algorithmischer Chemien finden im fünften und letzten Teil Eingang in weiterführende Betrachtungen.

Der Rekombinationsoperator wurde im Hinblick auf die Minimierung des Systemunterschieds zunächst aus der Rekombination linearer Individuen und den Unterschieden im Programmzeigerverhalten abgeleitet. Bei den Analysen des Parameterraums im Anschluss an die Optimierung zeigt sich, in Form einer geringen Rekombinationsrate, dass die Evolution hier hauptsächlich von der Mutationsoperation profitiert. Auch wenn die Rekombination als Suchoperation in der genetischen Programmierung nicht unumstritten ist, soll ihr hier eine zentralere Bedeutung zukommen. Inspiriert von homologen Rekombinationoperationen findet daher im Kapitel 11 die Entwicklung eines neuen Rekombinationsoperators statt. Die erneute Parameteroptimierung und empirische Betrachtung zeigt seine Eignung für die Evolution einer algorithmischen Chemie. Die Performance der Evolution ist nun

deutlich besser und weist zum Teil keinen signifikanten Unterschied zur Evolution linearer Programme im Vergleichssystem auf.

Durch die zufällige Auswahl und Ausführung von Instruktionen ist die Ausführungsdauer nicht determiniert. Die Anzahl der ausgeführten Instruktionen hat aber Einfluss auf den Evolutionserfolg und zudem zeigt sich, dass die Höhe und Größe der Individuen durch die Anzahl ausgeführter Reaktionen beeinflusst wird. Der Zusammenhang wird in der Wahrscheinlichkeit für die erfolgreiche Assemblierung der in der Chemie kodierten Lösung und ihrer Höhe vermutet. Eine Obergrenze für diese Wahrscheinlichkeit wird in Kapitel 12 hergeleitet.

Die Arbeit schließt mit einer Zusammenfassung und dem Ausblick im Kapitel 13.

Teil I.

Grundlagen

1. Genetische Programmierung

Die Evolution ausführbarer Strukturen als Teilgebiet der *evolutionären Algorithmen* (EA) geht zurück auf Cramer [36]. Besser bekannt ist dieses Teilgebiet jedoch unter dem Begriff genetische Programmierung (GP), welcher später von Koza [62] eingeführt wurde.

Eiben und Smith [40] grenzen die GP von den übrigen EAs durch das Anwendungsgebiet ab. Während andere Verfahren aus dem Bereich evolutionärer Algorithmen zumeist nach einer Eingabe suchen, die bei einem bekannten Modell eine Ausgabe mit maximalen Nutzen erzeugt (Optimierung), handelt es sich bei der GP um ein Verfahren des *maschinellen Lernens*. Bei diesem wird für ein bekanntes Eingabe-Ausgabe-Verhalten ein Modell mit der besten Übereinstimmung gesucht (Modellbildung)[1].

Genetische Programmierung

Genetische Programmierung bezeichnet die Erzeugung von ausführbaren Strukturen mit Mitteln der Evolution (Banzhaf u. a. [17]). Bei diesen Strukturen kann es sich um Computerprogramme unterschiedlicher Repräsentation handeln oder um andere Strukturen, wie z. B. neuronale Netze. Die Evolution arbeitet auf einer Population von Programmen, Individuen genannt, welche mit nichtdeterministischen Operationen erzeugt werden.

Abbildung 1.1 zeigt exemplarisch die Phasen, die ein Individuum und seine Nachkommen zyklisch durchlaufen. Jedes Individuum repräsentiert dabei eine potenzielle Lösung für ein definiertes Problem. Während die Individuen der ersten Generation zufällig erzeugt werden, gehen spätere Individuen aus den selektierten Individuen vorangegangener Generationen hervor. Sie werden durch Suchoperatoren erzeugt. Bei EAs kommen zumeist die Rekombination und die Mutation als Suchoperatoren zur Anwendung, wobei die erste Operation mehrere selektierte Individuen zu einem neuen Individuum kombiniert und die Mutation ein Individuum zufällig verändert. Die so erzeugten Individuen unterziehen sich einer Bewertung, bei der ihnen eine Fitness zugewiesen wird. Auf der Grundlage dieser Fitness findet später die Selektion statt. In der GP müssen der Genotyp des Individuums und die dadurch repräsentierte Lösung nicht identisch sein. Dann bedarf es einer Abbildung des Genotyps auf einen ausführ- und bewertbaren Phänotyp.

Von den Anfängen der GP bis heute wurde eine Vielzahl an Möglichkeiten vorgestellt, eine ausführbare Struktur im Genom eines Individuums zu repräsentieren. Mit diesen Repräsentationen gehen immer auch eigene Such- und Abbildungsoperationen einher. In diesem Kapitel werden einige Ansätze zur Evolution ausführbarer Strukturen und die zugehörigen Operationen vorgestellt. Für andere Aspekte der

[1] Als dritten Problemtyp nennen Eiben und Smith noch die Simulation, bei der bei gegebenem Modell die zu einer Eingabe gehörende Ausgabe gesucht ist.

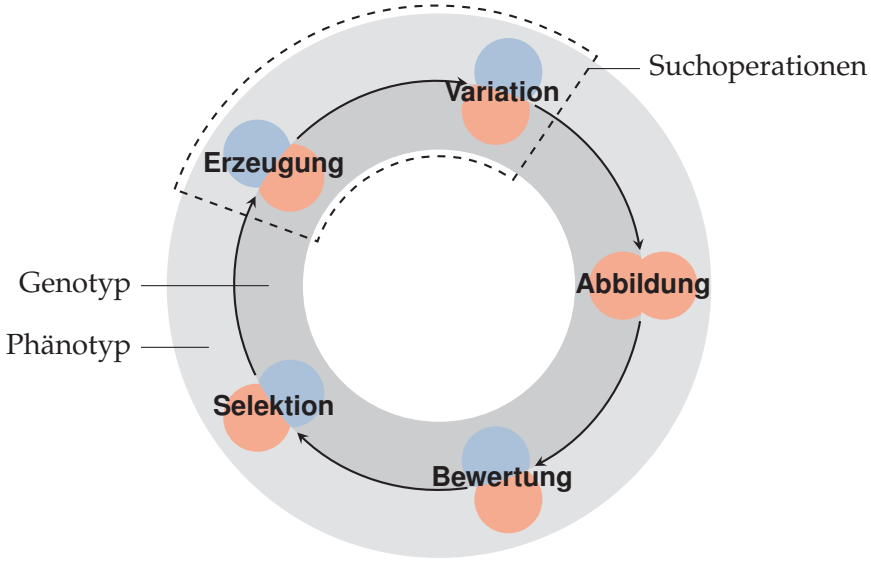


Abbildung 1.1.: Der vereinfachte Zyklus eines GP-Laufs, bestehend aus der Generierung, Bewertung und Selektion von Individuen. Die Individuen werden mittels Suchoperatoren erzeugt, die hierfür auf den selektierten Individuen arbeiten. Soweit sich Genotyp (innerer Ring) und Phänotyp (äußerer Ring) eines GP-Individuums unterscheiden, arbeiten die einzelnen Phasen auf jeweils einer dieser beiden Sichtweisen (rot markiert).

GP, wie z. B. die Fitnessberechnung und die Selektion, wird an dieser Stelle auf Koza [62, 63] und Banzhaf u. a. [17] verwiesen.

Die Einteilung der Repräsentationen erfolgt in drei Kategorien. Erstens werden in Kap. 1.1 die Ansätze nach Banzhaf [10] anhand der Frage voneinander abgegrenzt, ob Phänotyp und Genotyp eines Individuums unterschieden werden. Zunächst werden Ansätze vorgestellt, bei denen diese Unterscheidung nicht stattfindet, d. h. der Lösungs- und der Suchraum sind identisch. Zweitens motiviert und beschreibt Kapitel 1.2 Verfahren, bei denen der Genotyp vor der Bewertung zunächst auf den zugehörigen Phänotyp abgebildet wird. Als Drittes werden in Kap. 1.3 die Verfahren abgegrenzt, bei denen die Abbildung des Genotyps auf den Phänotyp nichtdeterministisch erfolgt.

1.1. Phänotyp-Evolution

Bei den in diesem Abschnitt behandelten Repräsentationen wird nicht zwischen Genotyp und Phänotyp unterschieden. Die Suchoperatoren arbeiten somit auf derselben Darstellung des Daten- und Kontrollflusses, wie die für die Ausführung und Bewertung zuständige Operation. Eine Abbildung findet nicht explizit statt, dennoch sind die Strukturen rein virtuell und ergeben sich erst durch Interpretation eines Speicherbereichs.

1.1.1. Baum-GP

Kozas [62] baumbasierte Repräsentation war die Erste, die zu einer nennenswerten Popularität gelangte. Die Struktur der Individuen entspricht bei ihm einem Syntaxbaum, dessen Knoten aus Elementen der *Funktionsmenge* und dessen Blätter aus Elementen der *Terminalmenge* stammen. Während in den Knoten die Informationsverarbeitung stattfindet, werden diese Informationen in den Blättern an das evolvierte Programm übergeben. Die Terminalmenge kann neben den Eingaben an das System aber auch konstante Werte oder Funktionen ohne Argumente enthalten. Die Funktionsmenge wird zumeist problemspezifisch gewählt und muss in der Lage sein, die gesuchte Lösung auszudrücken. Mögliche Elemente der Funktionsmenge sind Operatoren, Funktionen, Zuweisungen oder Kontrollflussanweisungen.

Mit Ausnahme von typisierten Ansätzen arbeiten GP-Verfahren zumeist mit syntaktisch abgeschlossenen Funktions- und Terminalmengen. Dieses bedeutet, dass alle Funktionen jedes Terminal und jeden Rückgabewert anderer Funktionen verarbeiten kann. Um z. B. in einem System mit reellwertigen Terminalen auch boolesche Operationen zuzulassen, muss jede boolesche Operation wissen, welche (reellwertigen) Werte sie als true und welche sie als false interpretiert. Enthält die Funktionsmenge die Division, so muss für die Abgeschlossenheit entweder sichergestellt werden, dass der Divisor nicht null wird, oder die „Division durch Null“ muss erkannt und abgefangen werden.

Die Darstellung der Individuen als Baum hat zwei Vorteile. Zum einen kann ein so evolviertes Programm direkt vom Benutzer als symbolischer Ausdruck interpretiert werden, zum anderen ermöglicht diese Struktur zusammen mit der syntaktischen Abgeschlossenheit eine einfache Realisierung der Suchoperatoren. Die Rekombination zweier Individuen zur Erzeugung von Nachkommen erfolgt durch den Austausch von Unterbäumen (Abb. 1.2). Hierfür wird in zwei selektierten Individuen zunächst jeweils ein Knoten gleichverteilt gewählt. Diese beiden Knoten stellen die Wurzeln von Unterbäumen dar, welche im nächsten Schritt ausgetauscht werden. Hierdurch entstehen zwei Nachkommen. Besitzen die ausgetauschten Unterbäume eine unterschiedliche Größe, dann können sich auch die Nachkommen in ihrer Größe von ihren Vorfahren unterscheiden. Aufgrund endlicher Speichergrößen wird dem so möglichen Wachstum der Individuen eine künstliche Grenze gesetzt. Bei der Mutation eines Individuums wird zunächst wieder einer seiner Knoten zufällig gewählt. Der an ihm beginnende Unterbaum wird gelöscht und an seiner Stelle ein zufällig erzeugter, neuer Baum eingefügt.

Automatically Defined Functions

Koza beschreibt noch einige weitere Suchoperatoren und Erweiterungen, von denen hier nur die *automatische Definition von Funktionen* (ADFs) [63] vorgestellt wird. Bei diesem Verfahren, wie auch bei anderen Modularisierungstechniken, werden bereits evolvierte Strukturen abgekapselt und als Modul aufgerufen. Im weiteren Evolutionsverlauf werden diese Module als Einheit betrachtet. Hierdurch sind sie bei der Rekombination geschützt und als Ganzes vererbbar. Module, welche

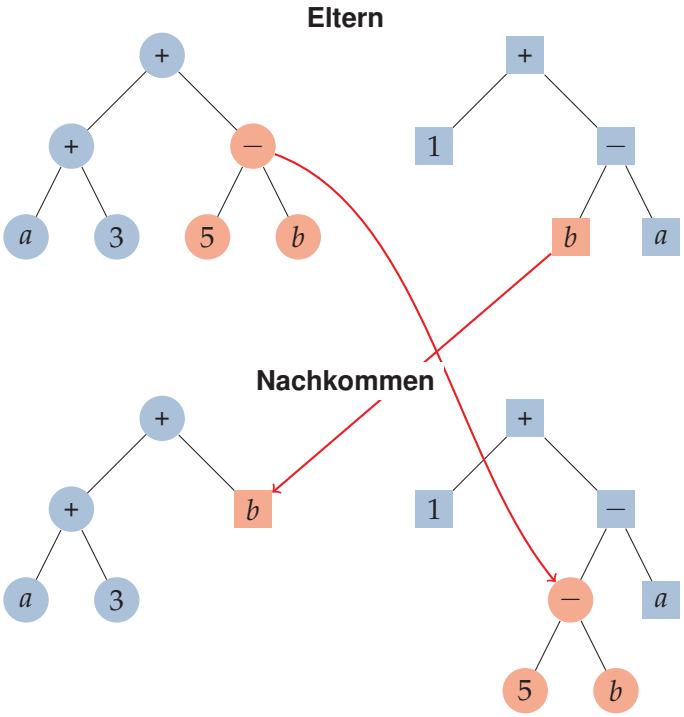


Abbildung 1.2.: Rekombination beim baumbasierten GP. Die Wurzel der markierten Unterbäume wird gleichverteilt in den beiden Bäumen gewählt. Zur Rekombination werden die beiden markierten Teilbäume zwischen den Eltern ausgetauscht, hierbei entstehen zwei Nachkommen.

sich im Laufe der Evolution bewähren und somit in der Population verbreiten, werden als *Building Blocks* interpretiert.

Für die Verwendung von ADFs spaltet sich der Baum eines Individuums nach der Wurzel in mehrere Zweige, die in unterschiedlicher Form der Evolution unterliegen:

1. Die *funktionsdefinierenden Zweige* enthalten die ADFs. Zu jeder ADF gehört eine Argumentenliste und ein Unterbaum mit dem eigentlichen Funktionskörper. Die Terminalmenge des Unterbaums wird durch die Elemente der Argumentenliste erweitert.

2. Die ADFs werden Teil der Funktionsmenge und können so im *ergebnisproduzierenden Zweig* des Individuums Verwendung finden.

Die Suchoperatoren müssen sicherstellen, dass die benötigten ADFs existieren, auch die ADFs können weiter der Evolution unterliegen.

1.1.2. Parallel Algorithm Discovery and Orchestration

Abweichend von der ursprünglichen Darstellung der Programme als Baum, verwenden Teller und Veloso [113] als Erste einen Graphen in ihrem System zur

Parallel Algorithm Discovery and Orchestration (PADO). Programme sind bei PADO gerichtete Graphen, bei denen jeder Knoten mit einer beliebigen Anzahl anderer Knoten verbunden sein kann, wobei jede Verbindung einen möglichen Kontrollfluss darstellt. Für den Datenaustausch zwischen den Knoten wird ein gemeinsamer Stapel (Stack) benutzt, d. h. die Operationen entnehmen dem Stapel ihre Argumente, führen die Berechnung durch und schreiben ihre Ergebnisse zurück auf den Stapel. Damit unterscheiden sich Daten- und Kontrollfluss unter Umständen. Zusätzlich existiert noch ein indexierter Speicher, in dem spezielle Operationen Daten vom Stapel speichern und aus dem andere Operationen Daten auf dem Stapel ablegen können. Die Knoten bestehen bei PADO aus zwei Teilen, neben der auszuführenden Operation wird in den Knoten die Entscheidung über den weiteren Verlauf des Kontrollflusses getroffen. Auch bei dieser Entscheidung können Informationen vom Stapel verwendet werden.

Zwei markierte Knoten stellen den Programmstart und das Ende dar, andere Knotentypen stehen für den Aufruf von weiteren Programmen. Diese Programme werden entweder im Individuum evolviert, dann heißen sie „Miniprogramm“, oder stammen aus einer Bibliothek aufrufbarer, extern evolvierter Programme. Die „Miniprogramme“ ähneln den ADFs, sie können sich rekursiv aufrufen oder auf Programme der Bibliothek zurückgreifen. Bei PADO werden Programme ausschließlich durch das Überschreiten einer festgelegten Ausführungsdauer terminiert. Sollte ein Programm vor Ablauf dieser Zeit seinen als Ende markierten Knoten erreichen, so wird das Ergebnis, ein bestimmter Wert aus dem indexierten Speicher, protokolliert und das Programm erneut aufgerufen. Aus allen protokollierten Werten errechnet sich am Ende der Laufzeit das Endergebnis, wobei die später protokollierten Ergebnisse ein höhere Gewichtung erhalten.

Bei der Rekombination wird jedem der beiden Eltern ein Teilgraph entnommen und dieser in den jeweiligen anderen Elter eingefügt, wo er mit dessen Restgraphen neu verbunden werden muss. Bei der Mutation wird ein Teilgraph vollständig entfernt und durch einen neu erzeugten Graphen ersetzt. Darüber hinaus unterliegen bei PADO Teile der Rekombinations- und Mutationsoperation selbst wieder der Evolution.

Auf den Programmen der externen Bibliothek arbeitet die Evolution indirekt. Die Bibliothek speichert Teillösungen der einzelnen Individuen und macht sie der Allgemeinheit zugänglich. Eine Bewertung findet über die Häufigkeit statt, mit der die Miniprogramme in den Individuen Verwendung finden. Selten verwendete Teillösungen werden durch neu gewählte ersetzt.

1.1.3. Multiple Interacting Programs

*Multiple Interacting Program*s (MIPs) sieht Angeline [5] als Generalisierung *neuronaler Netze* (NN). NN können beliebige, berechenbare Funktionen abbilden, wenn mehrere Schichten zugelassen werden. Sie benötigen hierzu aber unter Umständen sehr viele Neuronen. Angeline passt die in den Neuronen stattfindenden Berechnungen evolutionär an, um die Anzahl benötigter Neuronen zu verringern und die

Generalisierungsleistung der NN zu erhöhen.

Bei MIPs werden im neuronalen Netz drei Knotentypen unterschieden. Die Eingangsneuronen enthalten die Terminale des Systems, wie z. B. die Eingabe oder auch zufällig erzeugte Werte. Die Zwischen- und Ausgangsneuronen enthalten jeweils eigene Programme, die sich im Laufe der Evolution entwickeln. Die Repräsentation der Programme geschieht in den von Koza verwendeten Syntaxbäumen. Die Berechnungen der Ausgangsneuronen werden als Ausgaben des Systems interpretiert. Die Neuronen sind in vorwärts gerichteten oder rekurrenten Netzen miteinander verbunden. Die Programme greifen über die Kanten auf die Ergebnisse der Programme verbundener Neuronen zurück und interagieren so miteinander. Die Verbindungen der Neuronen bestimmen die Terminale, die in den Programmen der Neuronen verwendet werden.

Als Suchoperatoren definiert Angeline zehn Mutationsoperationen. Er teilt diese, nach dem Ausmaß der verursachten Strukturänderung, in drei Gruppen auf. In der untersten und mittleren Gruppe finden nur Änderungen innerhalb der Programme statt, welche die Struktur des Netzes nicht ändern. In der untersten Gruppe sind dieses einzelne Instruktionen in den Programmbäumen, in der mittleren Gruppe werden komplette Teilbäume geändert. Die Mutationsoperationen in der obersten Gruppe verändern auch die Struktur des Netzes, indem sie Programme austauschen, hinzufügen oder entfernen.

1.1.4. Parallel Distributed Genetic Programming

Parallel Distributed Genetic Programming (PDGP) von Poli [90] basiert auf der Darstellung der Individuen als Graph. Die Knoten des Graphen repräsentieren Funktionen und Terminale, die Kanten beschreiben den Kontroll- und Datenfluss. Den Vorteil gegenüber der Baumdarstellung sieht der Autor in der Mehrfachverwendbarkeit von berechneten Ergebnissen und der damit kompakteren Speicherung der Programme.

Die Knoten des Graphen werden in einem zweidimensionalen Gitter fester Größe angeordnet. Die beiden Dimensionen des Gitters werden im Folgenden als horizontale und vertikale Dimension bezeichnet. Jedem Punkt des Gitters ist ein Knoten mit einer Funktion oder einem Terminal zugeordnet. Ist der Knoten Bestandteil eines Datenflusses, der in einem der Ausgabeknoten endet, so wird er als aktiver Knoten bezeichnet. Die Verbindungen sind in der vertikalen Dimension gerichtet und nur zwischen Knoten benachbarter horizontaler Ebenen zulässig[2]. Dadurch ist die Verbindung in der vertikalen Richtung bereits vorgegeben und es genügt, einen horizontalen Offset anzugeben. Reicht dieser über die Grenzen des Gitters hinaus, so wird der Offset umgebrochen und an der anderen Seite des Gitters fortgesetzt.

Generiert und verändert werden die Individuen durch Operatoren, welche die syntaktische Korrektheit sicherstellen. Rekombination geschieht durch den Aus-

[2]Durch die in der Funktionsmenge enthaltenen Identitätsfunktion können Ergebnisse aber auch unverändert durch eine Ebene geleitet werden.

tausch von Teilgraphen mit anderen Individuen oder durch die Injektion eines Teilgraphen durch ein anderes Individuum. Ähnlich der Rekombination bei der Baumdarstellung wird gleichverteilt ein zufälliger Knoten gewählt. Alle Knoten, die an der Berechnung seiner Ausgabe teilhaben, gehören zum selektierten Teilgraphen. Dieser Teilgraph wird nun entweder an einer zufälligen Position in ein anderes Individuum eingefügt oder mit dem Teilgraphen eines anderen Individuums getauscht. Als Mutationsoperation sieht Poli das Einfügen eines zufälligen Teilgraphen und das Ändern der Knotenverbindungen vor.

Die Interpretation der Programme startet bei den auf der obersten Ebene liegenden Ausgabeknoten, diese rufen rekursiv die Auswertungsfunktionen der tiefer liegenden Knoten auf. Auch wenn die Interpretation sequenziell erfolgt, so betrachtet Poli [90] jeden Knoten als Prozessor und jede Verbindung als Kommunikationskanal. Die Auswertung gleicht dann dem parallelen Durchreichen von Anfragen und dem Rückübertragen von Ergebnissen. Diese Betrachtung des Interpreters als paralleler Datenflussrechner gibt der PDGP ihren Namen.

1.2. Genotyp-Evolution

Die oben beschriebenen GP-Varianten arbeiten auf den Phänotypen und damit direkt auf der Struktur des Daten- und/oder Kontrollflusses. Der Suchraum besteht aus der Menge aller syntaktisch korrekten Programme, die mit Hilfe der gewählten Repräsentation erzeugt werden können. Es liegt in der Verantwortung der Suchoperatoren, die syntaktische Korrektheit als harte Restriktion zu beachten.

Genotyp-Phänotyp-Abbildung

Motiviert durch morphogenetische Prozesse in der Natur, schlägt Banzhaf [10] die Restriktionsbehandlung über eine Genotyp-Phänotyp-Abbildung vor. *Genotyp und Phänotyp sind getrennte Sichten auf ein Individuum, die Suchoperatoren arbeiten auf dem Genotyp, welcher zur Auswertung auf den Phänotyp im restringierten Lösungsraum abgebildet wird.*

Wenn die Abbildungsfunktion sicherstellt, dass jeder Genotyp einen gültigen Phänotyp erzeugt, bedarf es keiner Beschränkungen im Suchraum. Des Weiteren haben Abbildungsfunktionen häufig die Eigenschaft, dass sie bzgl. einer Operation benachbarte Genotypen auf den gleichen Phänotyp abbilden, wodurch ihre Individuen trotz unterschiedlicher Genome die gleiche Güte erhalten. Die Operation, welche die benachbarten Genotypen ineinander überführt, ist mit Blick auf die Selektion neutral und so in der Lage, eine Gendrift zu erzeugen. Diese Drift ermöglicht wiederum einen Anstieg der Diversität, wodurch die Chance erhöht wird, ein lokales Optimum im Suchraum zu verlassen. Dieses Szenario ist in Abb. 1.3 dargestellt. Eine harte Restriktionsbehandlung, beispielsweise durch Zuweisung einer „tödlichen“ Fitness, hätte unter Umständen den Pfad aus dem lokalen Optimum verbaut, der nun über den Anstieg der Diversität gefundenen wurde. Kirschner und Gerhart [61] sehen in der Neutralität einen der Mechanismen, die zur Evolutionsfähigkeit eines Phylums beitragen, indem sie Auswirkungen mutativer Änderungen auf den Phänotypen abmildern. In der Natur sind dem-

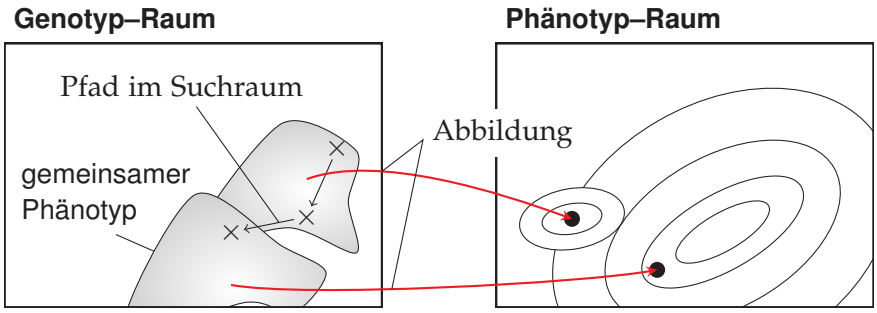


Abbildung 1.3.: Bei der Genotypsuche arbeiten die Suchoperatoren auf dem Genom des Individuums, welcher zur Bewertung der Güte auf einen ausführbaren Phänotyp abgebildet wird. Zumeist beschreiben mehrere, durch Suchoperator auseinander hervorgehende Genotypen, einen Phänotypen. Diese Eigenschaft führt zu Neutralität und als Folge zu Gendrift, der die Diversität der Genotypen steigert und den Fortschritt erleichtert. Zu sehen ist ein Suchpfad, der über einen neutralen Schritt zu einer Verbesserung führt.

Codon	Symbol
000	a
001	$+$
010	$/$
011	a
...	...

Tabelle 1.1.: Ausschnitt aus einer möglichen Abbildungsfunktion von Bitsequenzen auf Elemente der Terminal- und Funktionsmenge.

nach die meisten Mutationen (quasi-)neutral. Die Motivation in Kap. 4.1 wird auf die Evolutionsfähigkeit und ihre Mechanismen zurückkommen. Im Folgenden werden einige GP-Varianten beschrieben, die eine Genotyp-Phänotyp-Abbildung verwenden.

1.2.1. Lineare Genetische Programmierung

Im ersten Ansatz zur Genotyp-Phänotyp-Abbildung findet ein Bitstring-Genotyp Verwendung (Banzhaf [10]). Bei der Abbildung auf den Phänotyp werden aufeinander folgende Bits als codierende Region (Codons[3]) interpretiert, die Informationen über die zu verwendenden Operationen und Terminale enthalten. Die Zuordnung geschieht über eine vorgegebene Tabelle, wie sie in Tab. 1.1 exemplarisch abgebildet ist. Aus der Sequenz von Operationen und Terminalen wird ein Programmbaum erzeugt, welcher anschließend ausgewertet wird. Da keine Restriktionen im Suchraum existieren, können beliebige, Bitstrings verarbeitende Suchoperatoren verwendet werden.

In einer späteren Arbeit verwenden Keller und Banzhaf [58] eine veränderte Abbildungsfunktion. Die Menge syntaktisch korrekter Programme muss sich in

[3] In der Biologie bezeichnet der Begriff „Codon“ ein Basentriplett der mRNA.

diesem Ansatz durch eine kontextfreie Grammatik beschreiben lassen, was z. B. für ANSI-C Programme möglich ist. Beim Überprüfen einer Symbolsequenz auf ihre syntaktische Korrektheit bzgl. einer solchen Grammatik lässt sich zu jedem Zeitpunkt effizient die Menge gültiger Symbole bestimmen. Diese Eigenschaft nutzen die Autoren, um ungültige Sequenzen zu reparieren. Trifft ihre Abbildungsfunktion auf ein Codon, das ein ungültiges Symbol codiert, so wird aus der Menge gültiger Symbole jenes gewählt, dessen Codon die geringste Hamming-Distanz zum aktuellen Codon hat. Mit weiteren Mechanismen versuchen sie sicherzustellen, dass die Symbolsequenz am Ende der Abbildung vollständig ist. Unvollständige Abbildungen, wie z. B. „$3 \cdot a+$“, korrigieren sie durch Entfernen der hinteren Symbole, bis die Korrektheit hergestellt ist.

Der Abbildungsfunktion kommt für die Evolutionsfähigkeit besondere Bedeutung zu. Dem wollen Keller und Banzhaf [59] durch Evolution der Abbildungsfunktion (dem genetischen Code) Rechnung tragen. Anstelle der global definierten Abbildungsfunktion codieren die Individuen ihre eigene Abbildungsfunktion, welche einer mutativen Änderung und Selektion unterliegt. Unterschiedliche Abbildungen bedeuten eine unterschiedliche genetische Repräsentation identischer Phänotypen und damit eine unterschiedliche Fitnesslandschaft. Sie vermuten, dass sich jene Abbildungen durchsetzen, die für eine Individuum und seine Nachkommen eine geeignete Fitnesslandschaft erzeugen. Die Güte einer Abbildungsfunktion bewerten sie durch den Anteil der Genotypen, die auf den perfekten Phänotyp abgebildet werden. Sie zeigen, dass sich im Laufe der Evolution, zusammen mit Individuen hoher Fitness, auch Abbildungsfunktionen hoher Güte durchsetzen.

Compiling GP

Mit dem *Compiling GP System* (CGPS) arbeiten Nordin und Banzhaf [82, 84] nah an der von-Neumann-Architektur, die heutigen Rechnern zugrunde liegt. Ihr System verwendet eine Teilmenge des Maschinencodes des verwendeten Prozessors. Die Ausführung der Individuen bedarf daher nicht der Interpretation, sondern erfolgt direkt. Die verwendete Teilmenge der Instruktionen enthält vorwiegend arithmetische und logische Operationen sowie spezielle Instruktionen für variable Kontrollflüsse. Neben der Evolution von Unterfunktionen, ähnlich der ADFs, erlaubt ihr System auch den rekursiven Aufruf dieser Funktionen, wobei die maximale Rekursionstiefe beschränkt ist.

Die Codierung eines Individuums erfolgt als Binärstring in einem zusammenhängenden Speicherbereich. Nur ein Teil des Speicherbereichs unterliegt evolutionären Veränderungen. Andere Teile sind statisch und enthalten für die Ausführung benötigte Instruktionen, wie den Funktionsrahmen der Hauptfunktion und ggf. den der evolvierten Unterfunktionen. Die Rekombination erfolgt durch den Austausch von Blöcken vollständiger Instruktionen aus den nicht statischen Bereichen der Individuen. Falls vorhanden, wird zusätzlich eine Unterfunktion zwischen den Rekombinationspartnern ausgetauscht. Mutationen in einer Instruktion geschehen durch Wahl einer anderen Operation oder durch Wahl anderer Quell- bzw. Zielregister. Abbildung 1.4 zeigt das Beispiel eines linearen Genotyps und dem aus

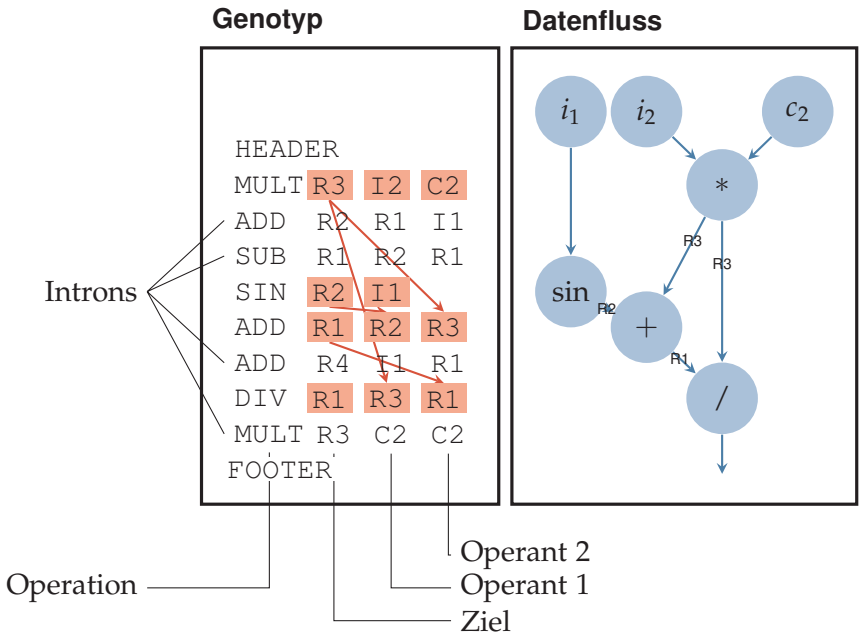


Abbildung 1.4.: Genotyp und der in ihm codierte Datenfluss, als Eigenschaft des Phänotyps, bei einem linearen GP-System.

ihm resultierenden Datenfluss als Eigenschaft des Phänotyps. Aufgrund des linearen Charakters der Genotypen ist die Arbeitsweise dieses Systems und ähnlicher Systeme als *lineare genetische Programmierung* (LGP) bekannt.

Linear-Tree-GP und Linear-Graph-GP

In den Varianten *Linear-Tree-GP* [56] und *Linear-Graph-GP* [57] tragen Kantschik und Banzhaf der Tatsache Rechnung, dass von Menschen erzeugten Programmen meist mehrere mögliche Datenflüsse generieren und entweder die Eingabe oder ein Zwischenergebnis zur Laufzeit über den Datenfluss bestimmt.

Der Datenfluss wird hierzu in erster Instanz in einem Baum bzw. Graphen vorstrukturiert. Jeder Knoten besteht aus einem linearen Programm, welches in einer Verzweigungsfunktion endet. Diese Funktion arbeitet auf denselben Variablen wie das lineare Programm und entscheidet über den Knoten, dessen lineares Programm anschließend zur Auswertung kommt. Sie erweitern die Knoten des Baumes bzw. des azyklischen, gerichteten Graphen mit dem Ziel, diese Strukturen mit den Vorteilen linearer Ansätze, wie z. B. erhöhter Neutralität, zu verbinden. Die Auswertung des Individuums terminiert, wenn das lineare Programm eines Knotens ohne Nachfolger abgearbeitet wurde. Sie benutzen zwei Rekombinationsoperatoren. Der erste Operator tauscht entsprechend der reinen graph- bzw. baumbasierten Ansätze Teilbäume/-graphen zwischen den Individuen aus. Der zweite Operator ist an das Vorgehen bei linearen Repräsentationen angelehnt. Er wählt zufällig je einen Knoten in beiden Eltern und tauscht Programmsegmente zufälliger Position

und Länge zwischen den linearen Programmen beider Knoten aus. Die Mutation besitzt die Möglichkeit, das lineare Programm einer Knoten auszutauschen, die Verzweigungsfunktion zu ändern oder Knoten neu zu verbinden.

1.2.2. Enzyme Genetic Programming

Unter der Bezeichnung *Enzyme GP* fassen Lones und Tyrrell [70] zwei Modelle zusammen, das „Activity Modell“ und das „Functionality Modell“. Eine Repräsentation als Baum dient ihrer Meinung nach vor allem der menschlichen Interpretation und weniger der Evolution. Sie sprechen dieser Struktur die Evolutionsfähigkeit ab und machen dieses an der schlechten Rekombinierbarkeit der Lösungen fest, wodurch die Rekombination zur ungeeigneten Operation für die systematische Betrachtung des Suchraums wird.

Das Vorbild dieser Repräsentation ist die Evolution metabolischer Netzwerke. Ihr Genotyp fester Länge beim „Activity Modell“ besteht aus einer konstanten Menge funktionaler Komponenten, die Instanzen der verfügbaren Elemente aus der Funktions- und Terminalmenge darstellen. Jede dieser Komponenten ist mit jeder anderen Komponente zu einem vollständigen Graphen verbunden. Das Genom der Individuen codiert die Stärke, mit denen die Verbindungen ausgeprägt sind. Eine Komponente mit ihren Verbindungen bezeichnen sie in Anlehnung an das biologische Vorbild als *Enzym*.

Bei der deterministischen Abbildung des Genotyps auf den Phänotyp, hier als Expression bezeichnet, folgen sie, ausgehend vom Ausgabeknoten, den stärksten Kanten. Sollte eine Verbindung rückläufig sein und somit zu einem bereits exprimierten Enzym führen, so wird die stärkste Verbindung verfolgt, die zu einem noch nicht erreichten Enzym führt. Das fundamentale Prinzip besteht nach Aussage der Autoren darin, dass die Struktur des Phänotyps aus den Verbindungspräferenzen der Komponenten abgeleitet wird. Rekombination erfolgt durch Austausch, Mutation durch Veränderung der Verbindungsstärken.

Das später eingeführte „Functionality Modell“ [69] soll den Nachteil fest definierter Komponentenmengen sowie die mit der vollständigen Verknüpfung einhergehende Redundanz beseitigen. Bei variabler Komponentenmenge lässt sich ein funktionaler Kontext nicht mehr fest mit einzelnen Komponenten verknüpfen. Um die Komponenten/Enzyme bei der Rekombination wieder in den richtigen Kontext einzuordnen, erhalten diese ein Profil. Das Profil wird beschrieben durch einen Punkt in einem Raum, dessen Dimension der Größe der Funktionsmenge entspricht. Die Koordinaten des Punktes werden durch die im Enzym enthaltene Funktion sowie von den Ausprägungen der an ihm anliegenden Enzyme bestimmt. Aufgabe der Schnittstelle ist es, das funktionale Profil des Enzyms wiederzugeben. Für die Eingänge der Enzyme wird das Profil der gewünschten Belegung per Evolution bestimmt. Ein Matching findet über die Distanz dieser Punkte im Raum statt.

Bei der Expression im Functionality Modell werden die explizit evolvierten Bindungswünsche der Enzyme durch die Distanz zwischen den evolvierten Ein-

gangprofilen und den errechneten Ausgangsprofilen ersetzt. Das Enzym mit der geringsten Distanz darf sich an einen Eingang binden. Die Rekombinationsoperation rekombiniert die Individuen, die ein ähnliches funktionales Profil, gemessen an den Ausgabeknoten, aufweisen. Dabei werden zusammenhängende Blöcke von Enzymen in das lineare Genom des anderen Individuums induziert, eine weitere Operation entfernt unabhängig hiervon einen anderen Block von Enzymen. Die Mutation arbeitet auf den von den Enzymen geforderten Eingangsprofilen.

1.2.3. Grammatical Evolution

Die *Grammatical Evolution* (GE) von O'Neill und Ryan [86, 87] erlaubt es, Programme in einer beliebigen kontextfreien Sprache zu evolvieren. Ihr System muss zu diesem Zweck die Grammatik der gewählten Sprache in Backus-Naur-Form (BNF) kennen.

Die BNF der Grammatik definiert für jedes Nichtterminalsymbol eine Menge alternativer Produktionsregeln. Bei der Erzeugung eines Phänotyps wird mit dem Startsymbol beginnend, iterativ versucht, alle Nichtterminalsymbole zu substituieren, bis ausschließlich Terminalsymbole im Phänotyp enthalten sind. Jedes Nichtterminalsymbol wird durch eine seiner in der Grammatik definierten Alternativen ersetzt. Die Wahl der Alternative ist im Genotyp, einem Binärstring variabler Länge, deterministisch festgelegt.

Zur Wahl einer Alternativen werden 8 aufeinanderfolgende Bits zu einem ganzzahligen Codon zusammengefasst. Diese $2^8 = 256$ unterschiedlichen Codons können weit mehr als die vorhandenen Produktionsregel adressieren. Der Wert des Codons wird daher ganzzahlig durch die Anzahl der Produktionregeln für das aktuelle Nichtterminalsymbol dividiert, der Divisionsrest bestimmt die zu wählende Alternative. Damit haben Änderungen des Genotyps, die zum selben Divisionsrest führen, keinen Einfluss auf den Phänotypen. Durch diese Form der Auswertung ist jedes Codon gültig. Eine vergleichbare Redundanz, so O'Neill und Ryan, lässt sich in der Natur beobachten. Dort codieren von den 64 möglichen Codons auf Nukleotidbasis 61 die 20 unterschiedlichen Aminosäuren. Dies wird als degenerierter Code bezeichnet. Die Bedeutung der Codons ist kontextabhängig. Der Kontext ist durch das Nichtterminal gegeben, für dessen Alternativenwahl es herangezogen wird.

Damit der Phänotyp vollständig ausgeprägt werden kann, wird der Genotyp unter Umständen mehrfach, zyklisch herangezogen[4]. Das biologische Vorbild der Autoren ist hier die Gen überlappende Codierung von Eigenschaften bei einigen Organismen. Sollten nach einer vorgegebenen maximalen Anzahl von zyklischen Genotypdurchläufen noch Nichtterminalsymbole vorhanden sein, so wird das Individuum mit dem schlechtesten Fitnesswert belegt. Ansonsten wird der so erzeugte Phänotyp der Fitnessfunktion zugeführt.

Durch die oben beschriebene Repräsentation lässt sich jedem Genotyp ein Fitnesswert zuordnen, wenn auch nicht immer über die Abbildung auf einen gültigen

[4]Dieses Vorgehen wird von den Autoren „Wrapping" genannt.

Tabelle 1.2.: Transkription und Translation bei der grammatischen Evolution und in biologischen Systemen (angelehnt an [86, Abb. 1]).

Grammatical Evolution	Biologisches System
Binärstring	DNA
Transkription	
Ganzzahlstring	RNA
Translation	
Produktionsregel	Aminosäure
Progammfunktionalität	Protein
ausführbares Programm	phänotypischer Effekt

Phänotyp. Im Raum des Genotyps sind beliebige Bitstrings erlaubt und Standardoperatoren für die Rekombination und Mutation können genutzt werden. Als zusätzliche Operation definieren O'Neill und Ryan die Codon-Duplikation, bei der Kopien einzelner Codons an das Chromosom angehängt werden.

Neben der Verwendung eines degenerierten Codes und der überlappenden Codierung von Eigenschaften sehen O'Neill und Ryan die GE auch bei der Transkription und Translation biologisch motiviert. Tabelle 1.2 stellt die einzelnen Schritte der GE den Analogien in biologischen Systemen gegenüber.

1.2.4. Induktion stochastischer, kontextfreier Grammatiken

Die Genotypen im System von Kammeyer und Belew [55] beschreiben stochastische, kontextfreie Grammatiken. Hier unterliegen die Grammatiken der Evolution. Dies unterscheidet ihr System von der Grammatical Evolution, bei der die Grammatik Teil der Systemkonfiguration ist.

Das Genom ist ein String aus Terminal- und Nichtterminalsymbolen sowie einem Sonderzeichen, welches den String in codierende Regionen unterteilt. Diese Regionen codieren nacheinander die linke und rechte Seite einer Produktionsregel sowie die Wahrscheinlichkeit ihrer Anwendung. Sollte eine codierende Region nicht passend interpretiert werden können, so wird sie ignoriert und mit der Betrachtung der nächsten Region fortgefahren. Dieses ist zum Beispiel dann der Fall, wenn eine Region als linker Teil einer Produktionsregel interpretiert werden soll, aber kein Nichtterminalsymbol enthält.

Die Trainingsmenge enthält Wörter und Wahrscheinlichkeiten, mit denen die Grammatik diese erzeugen soll. Die Fitness der Individuen basiert auf den Unterschieden zwischen den in der Trainingsmenge vorgegebenen Wahrscheinlichkeiten für die Erzeugung einzelner Wörter und den von der evolvierten Grammatiken erzeugten Wahrscheinlichkeiten. Vor der Auswertung werden die Individuen noch mittels einer lokalen Suche optimiert, bei der die evolvierten Wahrscheinlichkeiten der Produktionsregeln mithilfe einer Heuristik angepasst werden. Es handelt sich folglich um ein hybrides Verfahren.

1.2.5. Cartesian Genetic Programming

Die Bezeichnung *Cartesian Genetic Programming* (CGP), eine GP-Variante von Miller [76, 78], begründet sich in der kartesischen Anordnung indizierter Knoten eines gerichteten Graphen. Die Knoten sind in einer festen Anzahl von Zeilen und Spalten angeordnet und in ihrer Funktion und Verknüpfung konfigurierbar.

Die Konfiguration der Knoten geschieht im Genotyp der Individuen und besteht aus Funktionszuweisungen und Verbindungsdefinitionen für jeden Knoten. Die Funktion wird dabei aus der definierten Funktionsmenge gewählt. Jeder Eingang der Knoten wird mit dem Ausgang eines Knotens aus einer der vorangegangenen Spalten verbunden. Die maximale Arität in der Funktionsmenge gibt die Anzahl der Eingänge jedes Knotens vor, die Anzahl der interpretierten Eingänge hängt von der zugeordneten Funktion ab. Dadurch, dass sowohl die Anzahl der konfigurierten Eingänge, als auch die Anzahl der Knoten konstant ist, besitzt das Genom eine feste Länge. Die Ein- und Ausgabe des Systems geschieht über Knoten, die sich auf beiden Seiten des Arrays in einer separaten Spalte befinden und deren Elemente adressierbar sind. Der Adresse der Ausgabeknoten wird im Genom codiert.

Der Genotyp speichert die Knotenkonfigurationen in einer festen Reihenfolge sowie die Indices der Ausgabeknoten. Bei der Rekombination können diese Werte zwischen den Individuen getauscht werden. Während der Genotyp eine feste Länge besitzt, hat der Phänotyp eine variable Länge, die von der Menge nicht exprimierter Gene abhängt. Ein Gen gilt dann als exprimiert, wenn der Knoten, den es konfiguriert auf einem Pfad zwischen den Eingaben und den Ausgaben liegt. Bei der Initialisierung und der Mutation sorgen eine Menge von Bedingungen dafür, dass nur gültige Programme erzeugt werden.

1.3. Nichtdeterministische Genotyp-Phänotyp-Abbildung

Dieser Abschnitt beschreibt Repräsentationen in der GP, bei denen die Abbildung des Genotyps auf den Phänotyp nichtdeterministisch erfolgt. Ein Teil von ihnen ist dabei verwandt mit den *„estimation-of-distribution" Algorithmen* (EDA). Während die oben vorgestellten GP-Systeme angesammeltes Wissen in einer Population von Individuen speichern, verwalten EDAs ihr Wissen in einer Wahrscheinlichkeitsverteilung. Die Verteilung wird im Laufe der Evolution hier zugunsten der Erzeugung bevorzugter Programmvarianten adaptiert.

1.3.1. Probabilistic Incremental Program Evolution

Eingeführt wurde die *Probabilistic Incremental Program Evolution* (PIPE) von Salustowicz und Schmidhuber [101]. PIPE verwaltet eine Verteilung darstellbarer Programme in einem adaptiven, probabilistischen Prototypbaum. Für jeden Knoten des Baumes ist eine diskrete Wahrscheinlichkeitsverteilung über die Menge der vorhandenen Funktionen und Terminale definiert. Die Funktionen können unterschiedlicher Arität sein. Die Menge der Terminale, die bei Salustowicz und

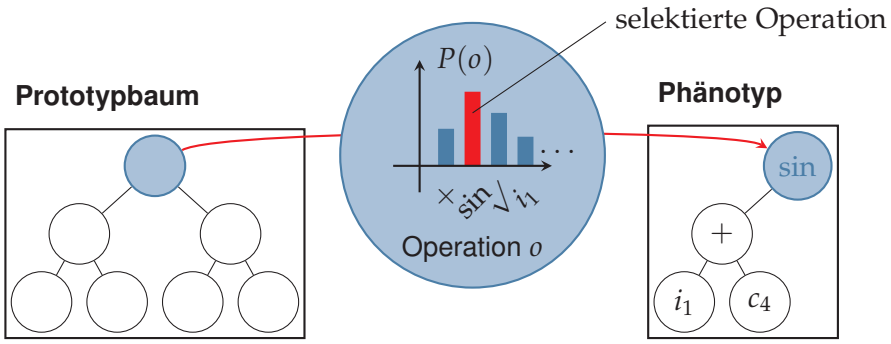


Abbildung 1.5.: Generierung eines Nachkommens auf der Grundlage eines Prototypbaums bei PIPE [101]. In der Mitte ist groß die Wahrscheinlichkeitsverteilung in der Wurzel des Prototypbaums dargestellt. Bei der Erzeugung des Individuums wurde auf der Grundlage dieser Verteilung zufällig die Sinusfunktion für seine Wurzel gewählt. Da diese Funktion die Arität 1 hat, wurde nur ein weiterer Unterbaum ausgebildet.

Schmidhuber als Funktionen der Arität null behandelt werden, setzt sich zusammen aus den Eingabevariablen und einer Konstanten, die zusätzlich in jedem Knoten evolviert wird.

Die Generierung neuer Individuen geschieht durch Sampling auf der Grundlage des im Prototypbaum gespeicherten Verteilungsmodells (Abb. 1.5). Jeder Knoten in den Phänotypen der Individuen wird auf der Grundlage der Verteilung im entsprechenden Knoten des Prototypbaums erzeugt. Die Ausbildung des Individuums beginnt mit der Wurzel. Dafür wird zunächst eine Funktion entsprechend der Wahrscheinlichkeitsverteilung in der Wurzel des Prototypbaums gezogen. Hat diese Funktion die Arität null (Terminal), so ist die Ausbildung des Baums beendet, bei Funktionen höherer Arität ist es nötig, eine entsprechende Anzahl von Teilbäumen auszubilden.

Die so erzeugten Individuen werden anschließend auf der Trainingsmenge ausgewertet. Die besten Individuen werden selektiert und zur Anpassung des Prototypbaums herangezogen. Dabei wird die Verteilung in den Knoten des Prototypbaums so verändert, dass sich die Wahrscheinlichkeit erhöht, die selektierten Individuen zu erzeugen. Die Stärke der Anpassung bzgl. eines Individuums hängt von der Relation seiner Fitness zur Fitness des allzeit Besten, dem Elitisten, ab. Mit einer definierten Häufigkeit erfolgt die Anpassung nur auf Grundlage des Elitisten. Um neben diesem Lernprozess den Suchraum weiter zu erforschen, werden in den Knoten des Prototypbaums die Wahrscheinlichkeiten für die einzelnen Instruktionen zusätzlich mutiert.

Die Größe des Prototypbaums, der initial nur die Wurzel enthält, wird dynamisch angepasst. Eine Erweiterung des Baums findet dann statt, wenn eine Funktion gewählt wurde, bei der nicht alle Eingänge bedient werden können, da die entsprechenden Unterbäume nicht existieren. Eine Beschneidung des Baumes in einem Knoten durch Entfernen seiner Unterbäume wird dann vorgenommen, wenn Teil-

bäume mit hoher Wahrscheinlichkeit nicht mehr benötigt werden, da Funktionen mit entsprechender Arität nur mit geringer Wahrscheinlichkeit gewählt werden.

1.3.2. Extended Compact Genetic Programming

Sastry und Goldberg [102] halten bei der Evolution von Programmen für essenziell, dass die Suchoperatoren das Zusammenwirken der Instruktionen wahrnehmen und berücksichtigen. Damit kritisieren sie die diesbezüglich relativ blinden Operatoren in der GP, wie z. B. die Rekombination durch Austausch beliebiger Teilbäume.

Mit *Extended Compact Genetic Programming* (eCGP) kombinieren Sastry und Goldberg PIPE aus dem vorherigen Abschnitt mit einem „Extended Compact"-Genetischen Algorithmus (eCGA) [51]. Der eCGA ist eine EDA-Variante, bei der Gene zu Partitionen zusammengefasst werden können, wobei ihre einzelnen Wahrscheinlichkeitsverteilungen durch eine gemeinsame Randverteilung ersetzt werden. Während Salustowicz und Schmidhuber bei PIPE nur einfache Zusammenhänge zwischen den Positionen im Baum und den zu wählenden Instruktionen erkennen, sollen mit eCGP multivariate Zusammenhänge genutzt werden.

Zu diesem Zweck passt eCGP nicht nur die Parameter, sondern auch die Struktur des Modells an. Die Strukturanpassung geschieht in jeder Generation auf der Grundlage der selektierten Individuen. Ziel der Strukturanpassung ist es, die Größe des Modells und die Komplexität der durch das Modell beschriebenen Population zu minimieren. Eine „gierige" Suchheuristik betrachtet dabei in jedem Schritt alle möglichen Modellvarianten, die aus der Kombination zweier Teilbäume entstehen und selektiert das beste Modell als Grundlage für den nächsten Minimierungsschritt. Dieses wird solange wiederholt, bis keine Verbesserung mehr erzielt werden kann. Der probabilistische Prototypbaum wird dabei in einzelne nicht überlappende Teilbäume aufgetrennt. Für diese Teilbäume wird dann die Randwahrscheinlichkeit der unterschiedlichen Ausprägungen[5] beschrieben. Die einzelnen Wahrscheinlichkeiten für die Zustände der internen Knoten entfallen. Die Wahrscheinlichkeit einer Teilbaumausprägung gibt somit am Ende der Evolution tendenziell die Bedeutung der gemeinsamen Verwendung der enthaltenen Instruktionen und Terminale wieder.

1.3.3. Probabilistische Grammatiken

Auch das Verfahren von Bosman und de Jong [24] arbeitet nach den EDA-Prinzipien. Während PIPE und eCGP die Verteilung der Programme in einem probabilistischen Prototypbaum modellieren, adaptieren Bosman und de Jong eine stochastische Grammatik, die zur Produktion der Individuen verwendet wird und verändern so im Laufe der Evolution den Fokus ihrer Suche. Die Produktionsregeln der Grammatik beschreiben Zusammenhänge, die im Gegensatz zu den beiden anderen Ansätzen unabhängig von einer Position im Genom sind. An den GP Standardverfahren kritisieren die Autoren, dass die Bedeutung positioneller Zusammenhänge

[5] Zur Vereinfachung lassen die Autoren in diesen Teilbäumen nur eine Instruktion zu.

zumeist unbekannt ist und somit Operationen wie die Rekombination keine Annahmen über die Art der auszutauschenden Teillösungen machen können[6].

Die Wahrscheinlichkeit für die Wahl einer Produktionsregel bei mehreren Alternativen ist in ihrem Ansatz von der Tiefe im Ableitungsbaum abhängig, in der die Regel Anwendung finden soll. Auf der Grundlage selektierter Individuen werden diese Wahrscheinlichkeiten angepasst.

Ratle und Sebag [97] wählen ein in weiten Teilen mit dem vorherigen Ansatz vergleichbares Vorgehen. Der Hauptunterschied liegt in der Verwendung einer festen Menge von Produktionsregeln. Dieses erlaubt es den Autoren, Vorwissen aus der Problemdomäne in das System zu integrieren. In ihrem Fall handelt es sich dabei um Wissen über die Umformung und Verknüpfung physikalischer Einheiten.

[6] Ansätze, die Teillösungen kapseln, wie z. B. die in Kap. 1.1.1 eingeführten ADFs, nehmen sie von ihrer Kritik aus.

2. Künstliche Chemie

Dieser Kapitel führt in den Themenkomplex der *künstlichen Chemie* (KC) ein und beschreibt die Merkmale ([39, 106]) in denen sie sich unterscheiden lassen. Das Kapitel schließt zum besseren Verständnis mit einigen Beispielen ab. Kapitel 3.1 enthält weitere Beschreibungen von KCs, die mit dem Ziel der Datenverarbeitung entworfen wurden.

Die Erforschung *künstlichen Lebens*[1] (ALife) beschäftigt sich mit nicht natürlichen Systemen, die Eigenschaften lebendiger Systeme[2] aufweisen. Untersucht werden dabei Prozesse auf molekularer, sozialer und evolutionärer Ebene, mit dem Ziel, die Komponenten, die zur Entstehung des Lebendigen führen, und ihre Wechselwirkungen zu erfassen. Den Wechselwirkungen kommt mit dem Konzept der Emergenz zentrale Bedeutung in der ALife-Forschung zu. *Bei Emergenz entstehen die globale Eigenschaften eines Systems aus der lokalen Interaktion vorhandener Komponenten.* Das heißt das System als Ganzes lässt sich nicht allein über die Eigenschaften der Komponenten beschreiben, wesentlich ist die Organisation der im Zusammenspiel der Komponenten bereitgestellten Funktionalität. Die Annahme, dass die Bedeutung und die Funktionsweise der *Organisationsprinzipien* disziplin-übergreifend ist, führt zu einer interdisziplinären Ausrichtung der ALife-Forschung.

Emergenz

Die künstliche Chemie ist eine Teildisziplin dieser ALife-Forschung und beschäftigt sich mit den Bedingungen auf molekularer Ebene, unter denen Leben entstehen kann (vergleiche Abb. 2.1). Von der Evolutionstheorie, die mit den Mitteln von Variation und Selektion versucht, Artenvielfalt und die Anpassung an unterschiedliche Lebensräume zu erklären, grenzt sie sich durch die Frage nach dem Ursprung der variierten und selektierten Einheiten ab. Die Abgrenzung zur sozialen Ebene kann über die Existenz des Lebendigen erfolgen.

Dittrich u. a. [39] listen die Fähigkeit zur *Reproduktion*, *Selbstorganisation* und *-erhaltung* als wesentlich für Entstehung komplexer Systeme. Banzhaf [12] identifiziert des Weiteren Selbstassemblierung und Selbsterzeugung als Bestandteil der Selbstorganisation und differenziert sie wie folgt: Bei der *Selbstassemblierung* entsteht das Ganze direkt aus seinen Teilen und ihrer Interaktion. Lassen sich bei diesem Prozess mehrere Stadien bzw. Systemzustände identifizieren, die zu immer komplexeren Einheiten führen, so handelt es sich um *Selbsterzeugung*. Die *Selbstorganisation*, als letzte Stufe, baut auf den beiden vorherigen auf und ermöglicht schließlich erst die Prozesse der Selbsterhaltung und Weiterentwicklung. Die KC-Forschung untersucht mittels abstrakter Modelle die Bedingungen und

[1] Auch im deutschsprachigen Raum als Artificial Life oder kurz ALife bekannt.

[2] Aus Sicht der Biologie sind dieses Wachstum, Fortpflanzung, Reizreaktion, Stoffwechsel und Bewegung, gelegentlich wird auch Zellbildung hinzugezählt.

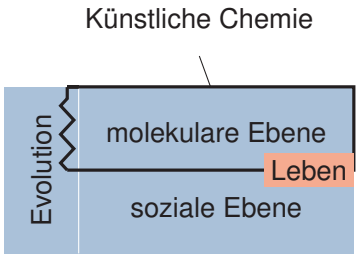


Abbildung 2.1.: Die Artificial Life Forschung untersucht Prozesse auf molekularer, sozialer und evolutionärer Ebene, mit dem Ziel, die für die Entstehung des Lebendigen wichtigen Organisationsprinzipien zu ermitteln. Die Evolution wird hier als orthogonal zu den beiden anderen Ebenen betrachtet, da Elemente dieser Ebenen selbst der Evolution unterliegen. Die künstlichen Chemie, als Teilgebiet von Artificial Life, beschäftigt sich überwiegend mit der molekularen Ebene, tangiert aber auch die Ebene der Evolution.

Mechanismen, mit denen die oben genannten Fähigkeiten gebildet werden können.

2.1. Gemeinsame Komponenten

In den meisten KC-Modellen lassen sich drei Komponenten identifizieren. Dieses sind die strukturgebende Menge von *Molekülen*, die *Regelmenge*, die die Interaktion der Moleküle beschreibt und damit die Funktion des Systems definiert, und der für die Anwendung der Regeln zuständige Algorithmus, der die *Dynamik* des Systems bestimmt. Die folgenden Abschnitte beschreiben die Komponenten und ihre möglichen Ausprägungen.

2.1.1. Moleküle

Die Menge S definiert die im System möglichen Moleküle. Bei einer endlichen Menge kann dieses durch vollständige Aufzählung geschehen ($S = \{s_1, \ldots, s_n\}$), bei einer unendlich großen Menge durch die Beschreibung ihrer Elemente (z. B. $\mathbb{N}^+$, jede Zeichenkette mit den Zeichen ‘a’ und ‘b’). Die Moleküle sind je nach Modell unterschiedlicher Natur, mögliche *Strukturen* sind λ-Ausdrücke [44], Binärstrings [8], Programme [98, 114], Vektoren [14], Beweise [27, 28] etc.

Reaktor

Zum Zeitpunkt der Ausführung befinden sich die Moleküle im *Reaktor* P, auch *Population* oder *Suppe* genannt. Bei der *expliziten Modellierung* der Moleküle im Reaktor entspricht die einfachste Form der Beschreibung einer Multimenge mit Elementen aus S. Diese Form der Modellierung kann z. B. dann erfolgen, wenn die Element im Reaktor keiner räumlichen Anordnung unterliegen und der Inhalt des Reaktors als gut gerührt angesehen wird. Bei der expliziten Modellierung der Moleküle ist der Speicheraufwand proportional zur Populationsgröße ($O(|P|)$). In einigen Systemen ist es hinreichend, die Moleküle *implizit* zu modellieren, z. B.

durch Verwaltung eines Konzentrationsvektors. Der Speicheraufwand liegt dann in der Größenordnung der Anzahl unterschiedlicher Moleküle ($O(|S|)$).

2.1.2. Reaktionen

Die *Regelmenge R* beschreibt die Interaktionen der Moleküle. Häufig wird die Notation der Regeln an die chemische Notation angelehnt, ein Beispiel bei explizit modellierten Molekülen wäre

$$\underbrace{s_{i_1} + \cdots + s_{i_n}}_{\text{Edukte}} \rightarrow \underbrace{s_{o_1} + \cdots + s_{o_m}}_{\text{Produkte}} \quad . \tag{2.1}$$

Die linke Seite beschreibt die für die Reaktion notwendigen Moleküle, auch *Edukte* oder Reaktanten genannt, das Zeichen „+“ trennt die einzelnen Moleküle voneinander. Nur wenn alle Elemente für die Reaktion vorhanden sind, gelangt die Regel zur Anwendung. Die rechte Seite listet die *Produkte* der Reaktion. Die Anwendung einer Regel wird bildlich als *Kollision* ihrer Edukte bezeichnet.

Bei der impliziten Modellierung der Moleküle beschreiben die Regeln eine Konzentrationsänderung im Reaktor

$$a_1 s_1 + \cdots + a_n s_n \rightarrow b_1 s_1 + \cdots + b_n s_n \quad . \tag{2.2}$$

Die Regel beschreibt, welche Anteile (a_i) der Edukte in welche Teile (b_i) der Produkte überführt werden[3]. In Anlehnung an das chemische Vorbild werden a_i und b_i als stöchiometrische Faktoren bezeichnet, sie unterliegen dabei aber bei den meisten Modellen nicht den Randbedingungen der Stöchiometrie, wie z. B. der Massenerhaltung.

Die explizite Auflistung der Regeln erzeugt u. U. eine sehr große Regelmenge oder ist unmöglich. Regeln können daher vereinfacht auch implizit formuliert werde. So bestimmen Banâtre u. a. [7] z. B. das Maximum einer Multimenge S über die Regel

$$s_1 + s_2 \rightarrow s_2, \quad \text{wenn } s_1 \leq s_2 \text{ mit } s_1, s_2 \in S\,.$$

Während solche einfachen Regeln für viele abstrakte Modelle ausreichen, berücksichtigen die Regeln komplexerer KC-Systeme wesentlich mehr Bedingungen, z. B. eine räumliche Anordnung [13] oder, bei der Simulation einer natürlichen Chemie, Informationen über die benötigte Mindestenergie.

Zeit wird in künstlichen Chemien in Form von *Generationen* gemessen. Eine Generation entspricht dabei der Ausführung einer festen Anzahl Reaktionen. Um die Entwicklungsgeschwindigkeit von der Reaktorgröße zu entkoppeln, wird die Anzahl der pro Generation ausgeführten Reaktionen gleich der Reaktorgröße gesetzt.

[3] Dabei müssen weder alle Moleküle für die Reaktion vorhanden sein ($a_i = 0$), noch müssen sie produziert werden ($b_i = 0$).

```
gegeben : Population P, Regelmenge R und Terminierungskriterium
repeat
    Moleküle ← subset(P);
    T ← anwendbar(R, Moleküle);
    /* T ⊆ R, ∀t ∈ T: Edukte(t) ⊆ Moleküle                    */
    if T ≠ ∅ then
        Regel ← wähleZufällig(T);
        P ← P ∪ Produkte(Regel) \ Edukte(Regel);
    end
until terminate();
```

Algorithmus 2.1 : Algorithmus zur Ausführung einer künstlichen Chemie bei explizit modellierten Molekülen.

2.1.3. Dynamik

Entsprechend den unterschiedlichen Darstellungsformen von Molekülen im System existieren verschiedene Algorithmen zur Beschreibung der Dynamik im Reaktor.

Explizite Molekülmodellierung

Bei Systemen, in denen die Moleküle explizit dargestellt werden, wird die Dynamik häufig mithilfe *zufälliger Molekülkollision* modelliert. Der Ablauf ist in Alg. 2.1 skizziert. Dem Reaktor werden dafür zunächst einige Moleküle entnommen. Im Fall einer unstrukturierten Multimenge geschieht dieses zufällig. Die gezogenen Moleküle legen die Teilmenge der Regeln fest, die Anwendung finden können. Dieses sind jene Regeln, bei denen für jedes Edukt ein entsprechendes Molekül gezogen wurde. Bei mehreren anwendbaren Regeln wird aus ihnen die anzuwendende Regel zufällig bestimmt. Können die gezogenen Moleküle den Eduktbedarf keiner Regel decken, so wird eine neue Teilmenge gezogen. Es handelt sich dann um eine so genannte *elastische Kollision*. Nach der Anwendung werden die Edukte aus dem Reaktor entfernt und ihre Produkte hinzugefügt. Ist die Menge der Produkte größer als die der Edukte, so bedarf es ggf. einem Abfluss von Molekülen, um ihre Anzahl konstant zu halten, prinzipiell ist ebenso ein permanenter Zufluss denkbar. Diese Prozedur wiederholt sich, bis ein vorgegebenes Terminierungskriterium erreicht wurde. Der Nachteil der Simulation mittels Molekülkollision ist der hohe Zeitaufwand, um Reaktionen hervorzurufen, die von Molekülen in geringer Konzentration abhängen.

Implizite Molekülmodellierung

Differenzialgleichungen eignen sich bei einer impliziten Darstellung der Moleküle zur Simulation der Dynamik. Jede Reaktion beschreibt Konzentrationsänderungen der an ihr beteiligten Moleküle. Es sei $[s_i]$ die Konzentration des Moleküls s_i, a_i^r und b_i^r sind die Anteile des Moleküls s_i in den Edukten und Produkten der Reaktion r

(Gl. 2.2). Dann beschreibt die Differenzialgleichung

$$\frac{d[s_i]}{dt} = (b_i^r - a_i^r) \prod_{j=1}^{n} [s_j]^{a_j^r}$$

die durch eine Regel $r \in R$ verursachte Konzentrationsänderung. Um die Konzentrationsänderung im Reaktor zu ermitteln, gilt es, die durch alle definierten Regeln verursachten Änderungen zu summieren. Der Nachteil dieses Ansatzes ist seine beliebige Genauigkeit. So bewirken auch noch Moleküle eine Änderung, deren Konzentration so gering ist, dass sie nur mit einer vernachlässigbar geringen Wahrscheinlichkeit eine Reaktion verursachen würden. Einfache Differenzialgleichungssysteme bieten der Vorteil, dass sie analytisch behandelt werden können, um so aus ihnen mögliche Fixpunkte oder Zyklen abzuleiten.

Konstruktive Systeme und Metadynamik

Einige Systeme weisen, neben der im vorherigen Abschnitt beschriebenen Dynamik, eine sogenannte *Metadynamik* auf, bei der sich die Menge der anwendbaren Regeln zur Laufzeit ändert. Hierzu bedarf es eines *konstruktiven Systems*, welches entweder explizit Regeln ändert, hinzufügt oder entfernt, oder die Menge der anwendbaren Regeln implizit ändert, indem Molekülarten verbraucht werden oder zuvor nicht existierende Molekülarten als Produkt einer Reaktion hinzukommen.

2.2. Beispiele

Skusa u. a. [106] teilen die Anwendungsfelder der künstlicher Chemien in Informationsverarbeitung, Optimierung und die Modellbildung, z. B. für biologische, soziale oder allgemein parallele Prozesse, ein. Im Folgenden werden einige Systeme aus dem Bereich der KCs beschrieben, die nicht der Informationsverarbeitung dienen. Beispiele für informationsverarbeitende KCs befinden sich im Kapitel 3.1.

Die ersten beiden Beispiele sind biologisch motiviert und modellieren die Wechselwirkung zwischen den in einem System vorhandenen Komponenten und deren Funktionalität, die Komponenten verändern kann. Das letzte Beispiel behandelt das System Tierra, welches sich sowohl als künstliches Sozialsystem als auch als künstliche Chemie interpretieren lässt und damit die interdisziplinäre Bedeutung der beobachteten Organisationsformen verdeutlicht.

2.2.1. Fontanas Algorithmische Chemie

Fontana [44] vermutet, dass die Fähigkeit zur Adaption von Systemen durch eine Wechselwirkung vorhandener Systemkomponenten mit der von ihnen bereitgestellten Funktionalität zustande kommt. Sind diese Interaktionen konstruktiv, so können die entstehenden Neuerungen zu lebendigen Systemen führen. Selbstorganisation, sowohl bzgl. der Komponenten als auch der mit ihnen assoziierten Funktionalität, ist in diesen Systemen von besonderer Bedeutung, da jede Neuerung

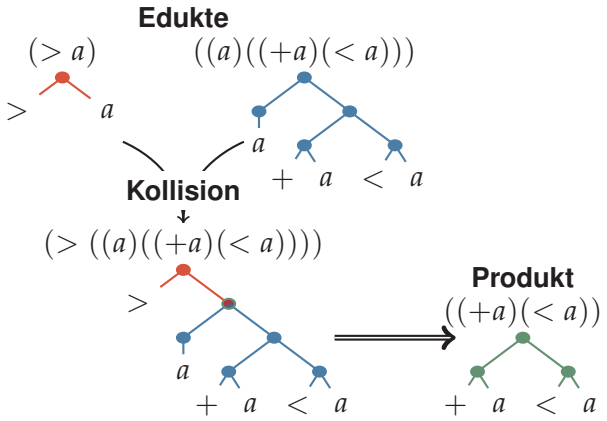


Abbildung 2.2.: In Fontanas Modell eines adaptiven Systems beeinflussen sich die Zusammensetzung der Systemkomponenten und die bereitgestellte Funktionalität wechselseitig. Er verwendet hierfür ein LISP ähnliches System Funktionen ändernder Funktionen. Die Abbildung stellt eine Kollision (Komposition) in diesem System dar. Zur Anwendung kommt dabei nur eine der sechs definierten Operationen (siehe [44], Anhang B). Es handelt sich um die „inversehead"-Operation ('>'), welche den letzten Ausdruck einer übergebenen Liste zurückgibt.

mit neuen Relationen einhergeht und zu starken Änderungen der Systeme führen kann. Die Tatsache, dass die Systeme weder in der Erzeugung neuer Komponenten noch in deren Interaktion beschränkt sind, bezeichnet Fontana mit „there are no 'absolut phenomena' in biology".

Er greift zur Modellierung seiner innovativen und adaptiven Systeme auf Alonzo Churchs λ-Kalkül in Form eines LISP System zurück. In einem solchen System lassen sich kombinatorische Variationen von Strukturen in Form von Funktionen darstellen, die eine wechselseitige Modifikation erlauben. Des Weiteren ermöglichen sie mit einer endlichen Beschreibung unendlich viele Strukturen.

Die Moleküle in seinem als algorithmische Chemie bezeichneten System sind Funktionen ändernde Funktionen der Arität eins. Die Menge $\mathcal{F}$ entspricht der Gesamtheit aller syntaktisch korrekten Funktionen im System. Die Definition der Reaktion setzt sich zusammen aus der eigentlichen Interaktion und zusätzlichen Kollisionsregeln. Interaktion ist dabei allgemein definiert über eine binäre Funktion $\phi(s,t) : \mathcal{F} \times \mathcal{F} \rightarrow \mathcal{F}$. Fontana beschreibt die Komposition $f(g(x))$ zweier Funktionen $f(x)$ und $g(x)$ als mögliche Interaktion, die in einer neuen Funktion $h(x)$ resultiert ($h = \phi(f,g)$). Abbildung 2.2 stellt eine solche Komposition zweier Funktionen dar. Die Kollisionsregeln Φ schaffen die Rahmenbedingungen für die Interaktionen und verhindern so unerwünschte Ereignisse wie die Kollisionen mit `NIL`, unendliche Rekursionen oder die Entstehung der Identitätsfunktion. Die Kollision $\Phi(f,g)$ der Funktionen f und g erzeugt $h = \phi(f,g)$, falls die Kollisionsregeln die Funktion h akzeptieren, ansonsten gilt die Kollision als elastisch.

Für die Dynamik werden zwei unterschiedliche Varianten definiert. Die erste

Variante entspricht der iterativen Anwendung der Abbildungsfunktion

$$M : \mathcal{P} \mapsto \mathcal{P}, \quad \mathcal{A}_{i+1} = \Phi[\mathcal{A}_i] \,.$$

Dabei ist $\mathcal{P}$ die Potenzmenge aller möglichen Funktionsobjekte $\mathcal{F}$ in dem System. Der Zustand des Systems in Iteration i ist definiert über die aktuell im System existierende Teilmenge $\mathcal{A}_i \in \mathcal{P}$. Die Funktion

$$\Phi[\mathcal{A}] = \{j : j = \phi(i,k),\ (i,k) \in \mathcal{A} \times \mathcal{A}\}$$

beschreibt die Durchführung alle paarweisen Kollisionen in einem Iterationsschritt.

Als Turinggas bezeichnet Fontana die zweite Form der von ihm definierten Dynamik. Das Turinggas wird als Multimenge modelliert, wodurch sich Funktionen in ihrer Anzahl und damit Konzentration unterscheiden können. Die Dynamik lässt sich als stochastischer Prozess beschreiben und ähnelt der in Alg. 2.1 beschriebenen Dynamik. Zwei Funktionen f und g werden zufällig gewählt, die Erste auf das Zweite angewandt und anschließend werden alle drei Funktionen Bestandteil der Multimenge,

$$f + g \rightarrow f + g + (f(g)).$$

Die Änderung in der Häufigkeit einer Funktion hängt damit von der Häufigkeit der Funktionen ab, die zu seiner Erzeugung benötigt werden. Um die Anzahl der Funktionen konstant zuhalten, wird nach jeder produktiven Reaktion eine Funktion zufällig entfernt.

Unabhängig von der eigentlichen Funktionsmenge lassen sich Eigenschaften natürlicher Systeme in diesem System mathematisch definieren. Auf diese Weise beschreibt Fontana autokatalytische Prozesse, Saatmengen oder parasitäre Teilmengen und die Beziehungen zwischen solchen Systemen. Für die Definition einiger dieser Eigenschaften greift er auf den sogenannten Interaktionsgraph ([44, Kap. 4]) zurück.

Ein Beispiel für eine solche Definition, die nicht des Interaktionsgraphen bedarf, ist die Autokatalyse. Ein Zustand $\mathcal{A}$ gilt als autokatalytisch, wenn gilt,

$$\forall j \in \mathcal{A},\ \exists\, i,k \in \mathcal{A}, \text{so dass } j = \Phi(i,k) \,,$$

d. h. alle Systemkomponenten können aus Komponenten des Systems erzeugt werden.

In seinen Experimenten konnte Fontana die Entstehung vieler Organisationsformen beobachten. So entwickelten sich z. B. gegenseitig stabilisierende Interaktionsmuster und Hierarchien, die auf selbsterhaltenden, interagierenden Teilmengen basieren. Ein explizites Verbot von Funktionen, die eine identische Abbildung erzeugen, führte nicht nur zu mehr Diversität, sondern auch zu einer verstärkten Kooperation auf Mengenebene und einer höheren Robustheit gegenüber externen Störungen in Form zufälliger Funktionen.

2.2.2. Matrixmultiplikationschemie

In der Genetik sind RNA Stränge zum einen Informationsträger, zum anderen können sie als Enzym (Ribozyme) chemische Reaktionen katalysieren und somit die Verarbeitung von RNA Strängen beeinflussen. Inspiriert von dieser Doppelfunktion beschreibt Banzhaf [8] ein System, dessen Elemente diese Doppelfunktion nachahmen und mit ihr die Fähigkeit zur Replikation zeigen.

Bei den Molekülen dieses Systems handelt es sich um Binärstrings fester, quadratischer Länge N (mit $\sqrt{N} \in \mathbb{N}^+$), die als Genotyp $\vec{s} = (s_1, \ldots, s_N)$ eines Individuums betrachtet werden. Ein Genotyp kann reversibel auf eine zweidimensionale Matrix, dem Phänotyp, abgebildet werden ($\vec{s} \rightarrow \mathcal{P}_{\vec{s}}$), wodurch dieser anschließend als Operator verwendet werden kann. Dieser Prozess wird in Anlehnung an das oben beschriebenen biologische Vorbild „Faltung“ genannt. Ein solcher Phänotyp kann nun mit einem anderen Molekül $\vec{s}'$ reagieren

$$\vec{s}'' = f(\mathcal{P}_{\vec{s}}, \vec{s}')$$

und erzeugt dabei eine neues Molekül $\vec{s}''$. Es handelt es sich bei der Funktion f um eine Kombination von Matrixmultiplikation und Schwellenwertfunktion. Neben dem Reaktionsprodukt werden auch $\vec{s}$ und $\vec{s}'$ wieder freigegeben. In Anlehnung an die Reaktionsgleichung (2.1) gilt daher

$$\vec{s} + \vec{s}' \rightarrow \vec{s} + \vec{s}' + \vec{s}'' \quad .$$

In jedem Zeitschritt werden dem Reaktor zwei seiner M Binärstrings entnommen, einer wird in einen Operator „gefaltet“ und mittels f auf den anderen String angewandt. Anschließend gelangen die beiden Strings zusammen mit dem Reaktionsprodukt wieder in den Reaktor. Um eine Konkurrenzsituation zwischen den Molekülarten zu erzeugen, wird das „Volumen“ des Reaktors künstlich konstant gehalten und hierfür nach jeder Reaktion ein überzähliges Element zufällig entnommen. Um auf Dauer im Reaktor bestehen zu können, müssen Binärstrings sich entweder selber reproduzieren oder durch andere reproduziert werden[4]. Dies führt zur Selektion von vergleichsweise schnell reproduzierbaren Elementen.

Die Simulation und die zusätzliche Modellierung mittels eines Differenzialgleichungssystems zeigen, dass eine solche KC ausgehend von der identischen Konzentration aller Molekülarten[5] in einen Attraktor gelangen kann, in welchem eine Teilmenge der Binärstrings koexistiert. In späteren Untersuchungen [11] wird das System um die Fähigkeit erweitert, Binärstrings variabler und nicht quadratischer Länge miteinander reagieren zu lassen. Die Dynamik dieses System tendiert zu komplexeren Molekülen, gemessen an der Länge der Strings. Zudem wird die durch ein begrenztes Volumen bedingte Selektion durch eine weitere gerichtete Selektion ergänzt ([9]). Die Binärstringmoleküle erhalten hierfür eine Semantik, sodass

[4] Dieses schließt längere Ketten zyklischer Abhängigkeit mit ein.

[5] Vom Reaktor ausgenommen wurde der aus Nullen bestehende, Destruktor genannte, String, der sich bei der Reaktion mit einem beliebigen String reproduziert.

ihnen eine Güte, als Maß für die Eignung zur Lösung eines Optimierungsproblems, zugeordnet werden kann. In regelmäßigen Abständen werden Moleküle schlechter Güte durch die variierte Kopie eines zufällig gewählten, anderen Moleküls aus der Chemie ersetzt.

2.2.3. Autonome Programme

Im Bereich ALife sind im Laufe der Jahre einige Systeme entstanden, deren Gemeinsamkeit darin besteht, dass sie sich des Bilds parallel agierender Prozesse bedienen, deren Programme um einen gemeinsamen Speicherbereich konkurrieren[6] [96, 98]. Exemplarisch für diese Klasse von Systemen wird im Folgenden Rays Tierra System [98] und eine Erweiterung von Thearling [114] beschrieben.

Das Tierra System besteht aus einem zentral verwalteten Programmspeicher, indem sich Programme, auch „Kreaturen“ genannt, befinden, denen jeweils ein eigener Prozessor exklusiv zugeordnet wird. Das Speichermanagement erlaubt es den Programmen den Code anderer Programme ausführen, nicht aber zu überschreiben. Die Nutzung fremden Programmcodes wird als Interaktion betrachtet. Während die Analogie der Entwicklung in diesem System zur sozialen Evolution schon aufgrund der Begriffswahl („Kreaturen“) augenscheinlich ist, kann die Entwicklung auch als chemische Evolution betrachtet werden. Dann handelt es sich bei den Programmen um die Moleküle, die sich einen Reaktor, in Form des Speichers, teilen. Jede Interaktion, die wie oben beschrieben erfolgt, stellt dann eine Reaktion dar, die unter Umständen in einer veränderten Funktionalität resultiert.

Den Anfang bildet ein einzelnes, von Hand erstelltes, Individuum, welches die Fähigkeit besitzt, sich zu replizieren. Hierfür reserviert es Speicher für den Nachkommen und kopiert sich in diesen. Anschließend agieren beide Programme unabhängig voneinander. Diversifikation entsteht über Fehler (Mutationen) beim Kopiervorgang. Wurde die Funktion nicht zerstört, reproduzieren sich die Individuen fortlaufend. Ist der Speicher gefüllt, so entfernt ein „Reaper“ genannter Prozess Individuen auf der Grundlage von Alter und den in der Vergangenheit erzeugten Fehlern.

Ray beobachtet eine Entwicklung, die zu Individuen kleiner Programmlänge führt, die somit schneller reproduziert werden können. Die Längenreduktion geschieht häufig dadurch, dass diese Individuen keine eigene Routinen für die Reproduktion enthalten, sondern diese von anderen Individuen nutzen. Als Sozialsystem betrachtet weisen sie somit ein parasitäres Verhalten auf.

Thearling [114] erweitert das System um die Fähigkeit zur parallelen Berechnung, indem Programmen mehrere CPUs mit eigenem Registersatz und Befehlszeiger

[6] Dieses Bild geht zurück auf das Spiel „Core War“ von Dewdney [38] in dem die Spieler mit ihren Programmen versuchen, durch Manipulation des gemeinsamen Speicherbereichs (der Core), gegnerische Programme zum Ausführen einer ungültigen Instruktion zu bewegen. Der für die Programmierung verwendete „Redcode“ ist Assembler ähnlich und enthält zehn verschiedenen Instruktionen, unter anderem Instruktionen für die Aufteilung und parallele Ausführen von Programmen.

zugewiesen werden können, die weiterhin auf einem gemeinsamen Programmcode agieren. Dieses ist zum einen biologisch motivert durch den Wunsch, mehrzellige Systeme simulieren zu können. Zum anderen will Thearling die Metapher des Mehrzellers für die Modellierung paralleler Berechnungen nutzen.

Um vom Zustand des Einzellers in einen mehrzelligen Zustand zu wechseln, wird eine zusätzliche „split“-Operation eingeführt. Beim Aufruf dieser Operation wird ein weiterer Prozessor dem Programm zugeordnet. Die Programmzeiger beider Prozessoren zeigen anschließend auf die Instruktion nach der „split“-Operation. Ein Registerwert, der sich in beiden Einheiten unterscheidet, ermöglicht im Folgenden unterschiedliche Kontrollflüsse. Ein neuer, von Hand kreierter Vorfahre sorgt dafür, dass vor Erzeugen der Nachkommenkopie das Elter-Programm auf zwei Prozessoren aufgeteilt wird und jeder Prozessor eine Hälfte des Elters kopiert, wodurch die Reproduktion in der Hälfte der Zeit erfolgen kann.

Das System weist in seiner Entwicklung nun zwei Phasen auf. In der ersten Phase findet, ähnlich dem Ursprungssystem, eine Optimierung statt, die zu einer Verkürzung der Programmlänge und zu Parasitismus führt. In der zweiten Phase wird die Kopieerzeugung weiter parallelisiert, bis zur Ausnutzung der maximal 16 möglichen Prozessoren pro Individuum. Es entstehen Individuen, die ihre Performance durch intensive Parallelisierung steigern.

3. Alternative Informationsverarbeitung

Dieses Kapitel beschreibt zunächst Systeme, in denen die Selbstorganisation ihrer Komponenten einen entscheidenden Faktor für die Informationsverarbeitung auf globaler Ebene darstellt. In einem zweiten Abschnitt werden Beispiele aus einem jungen Gebiet der Informatik dem „Spatial Computing“ behandelt. Bei diesen wird versucht, einen Teil des Aufwands für die Informationsverarbeitung von der Zeit in den Raum zu verlagern.

3.1. Selbstorganisierende, informationsverarbeitende Systeme

Die folgenden Abschnitte stellen künstliche und natürliche Chemien vor, die konstruiert wurden, um Informationen zu verarbeiten. Die Fähigkeit ihrer Komponenten zur Selbstorganisation, die sie direkt oder indirekt zur parallelen, teils asynchronen Verarbeitung nutzen, war für ihre Wahl ausschlaggebend.

3.1.1. Künstliche Chemie

Das Bild der Chemie mit ihren vielen zeitgleich stattfindenden Reaktionen dient der ersten Gruppe von Systemen als Metapher für parallele Algorithmen und großen dezentralen Datenstrukturen ([27, 114, 120]).

Metabolische Robotersteuerung

Bakterien reagieren sehr schnell auf Änderungen der Umwelt, indem sie ihren Stoffwechsel verändern. Die daran beteiligten biologischen Zellen nehmen Informationen in Form von Stimuli wahr und verarbeiten diese. Die Signalverarbeitung in Bakterien ist robust, die Funktionalität in ihnen hierarchisch in unterschiedliche Einheiten gegliedert. Diese Einheiten verarbeiten nicht nur Informationen anderer Einheiten weiter (metabolisches Netzwerk), sondern beeinflussen sich auch wechselseitig (regulatorisches Netzwerk). Die Fortbewegung ist eine Aufgabe, die, gespeist von externen Stimuli, von Bakterien gelöst wird. Dieses motiviert Ziegler u. a. [120], Zellen als Modell für ein robustes und verteilt agierendes Kontrollsystem heranzuziehen. Dabei verwenden sie eine künstliche Chemie, um die Steuerung eines Roboters zu realisieren.

In ihrer Implementierung betrachten sie das Bakterium als funktionale Einheit, die durch einen Reaktor dargestellt ist. Die Moleküle sind implizit als Konzentrationsvektor gegeben. Eingaben erhält das System über 8 Distanz- und 8 Lichtsensoren,

deren Zustand die Konzentration einer Molekülart ändert. Von der Konzentration zweier anderer Moleküle wiederum ist die Ansteuerung der beiden Motoren abhängig, die, mit Rädern verbunden, den Roboter fortbewegen. Das Reaktionssystem wurde so konstruiert, dass der Roboter das Licht sucht und den Kontakt mit anderen Objekten meidet. Zusätzliche Versuche zeigen, dass die so konstruierte Steuerung robust gegen Störungen ist, die bei diesen Versuchen in Form von kleinen zufälligen Konzentrationsänderungen eingebracht wurden.

Sortieralgorithmus

Banzhaf u. a. [14] betrachten die Sortierung einer Sequenz, die Ermittlung der Parität eines Bitstrings und die Berechnung von Primzahlen mithilfe künstlicher Chemien. Strategien zur Lösung dieser Probleme setzen sie so um, dass als Folge lokaler Interaktionen die gewünschte Berechnung stattfindet und das Ergebnis bei makroskopischer Betrachtung ermittelt werden kann.

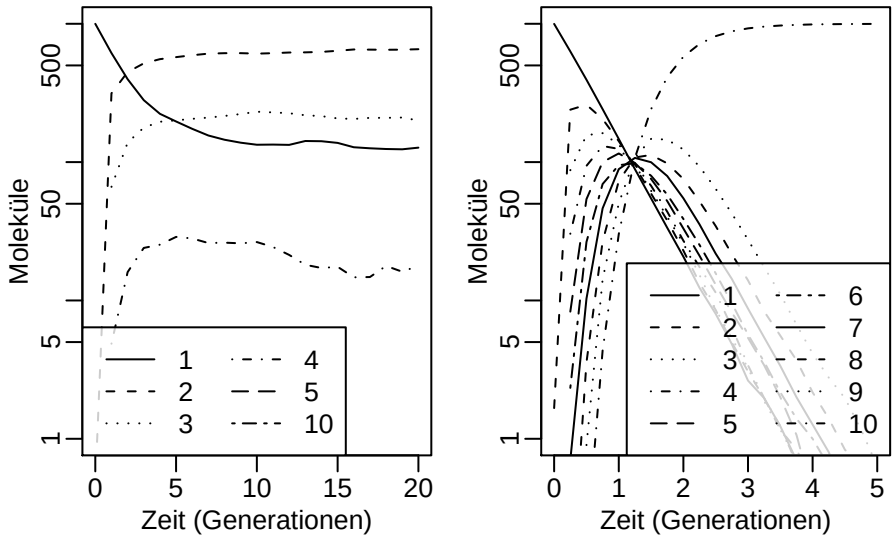
Im Folgenden wird noch einmal der Sortieralgorithmus nachvollzogen, da die Experimente die Bedeutung der Wahl eines angemessenen Reaktionsschemas für die Erreichung eines Ziels verdeutlichen. Die Moleküle des Systems bestehen zunächst aus den zu sortierenden Werten, die mehrfach im Reaktor abgelegt werden. Die Sortierung muss aus der Interaktion dieser Werte entstehen. Die Reaktionen sind konstruktiv, sodass sich im Laufe der Zeit auch Zwischenergebnisse in Form von Wertsequenzen im Reaktor befinden.

In einem ersten, „naiven“ Ansatz wird versucht, schrittweise geordnete Sequenzen größerer Länge zu erzeugen. Zu diesem Zweck sind nur solche Reaktionen nicht elastisch, bei denen ein Wert n_j mit einer Sequenz $N_i(k)$ der Länge k reagiert, dessen größter Wert n_i kleiner n_j ist. Es existieren folgende Reaktionen:

$$
\begin{array}{ll}
N_i(k) + n_j \rightarrow N_i(k) + N_j(k+1) & \text{wenn } n_i < n_j, \\
N_i(k) + n_j \rightarrow N_i(k) + N_i(k) & \text{wenn } n_i = n_j, \\
N_i(k) + n_j \rightarrow N_i(k) + n_j & \text{(elastisch) sonst.}
\end{array}
$$

Die Dynamik des Systems führt dazu, dass Sequenzen entstehen, die eine sortierte Teilmenge der Elemente erhalten, wobei die einzelnen Elemente n_j konsumiert werden. Hierdurch gelangt das System sehr schnell in einem quasi stabilen Zustand, in dem ausschließlich elastische Reaktionen möglich sind (vgl. Abb. 3.1a). Eine sortierte Sequenz mit allen Elementen entsteht zumeist nicht.

Ein zweiter Ansatz basiert auf einer einzelnen Regel, die der Bildung und Kopie eines DNA-Doppelstrangs nachempfunden ist. Hierfür werden zunächst die Elemente zweier Sequenzen aneinander ausgerichtet und in beiden Sequenzen die Elemente ergänzt, die nur in der anderen Sequenz enthalten sind. Der Doppelstrang wird wieder aufgetrennt und die beiden einzelnen Sequenzen wandern in den Reaktor. Folgendes Beispiel verdeutlicht exemplarisch den Übergang



(a) Naiver Ansatz **(b)** Doppelstrang-Ansatz

Abbildung 3.1.: Dynamik zweier künstlicher Chemien mit unterschiedlichen Regelmengen. In jedem Reaktor befinden sich 10 verschiedene Molekülarten, die zu geordneten Sequenzen reagieren können. Dargestellt ist der Anteil von Molekülen mit den jeweiligen Sequenzlängen, die gesuchte Lösung hat die Länge 10. Die Experimente wurden denen von Banzhaf u. a. [14] nachempfunden, gemittelt wurde über 30 Wiederholungen. Im naiven Ansatz lassen sich lediglich sortierte Sequenzen bis zur Länge 4 beobachten.

$N_c(3) + N_d(3) \rightarrow N_d(4) + N_d(4)$:

$$\begin{matrix} N_c(3) = \{n_a, n_b, n_c\} \\ N_d(3) = \{n_a, n_c, n_d\} \end{matrix} \rightarrow \begin{pmatrix} \{n_a, n_b, n_c, -\} \\ \{n_a, -, n_c, n_d\} \end{pmatrix} \rightarrow \begin{pmatrix} \{n_a, n_b, n_c, n_d\} \\ \{n_a, n_b, n_c, n_d\} \end{pmatrix}$$

In Abb. 3.1b lässt sich anhand der Längen, der entstehenden Sequenzen, die Dynamik des Systems verfolgen. Bereits nach kurzer Zeit haben alle Sequenzen die maximale Länge erreicht, enthalten also alle Werte in sortierter Reihenfolge. Der Vergleich beider Systeme zeigt die Bedeutung angemessener Reaktionsschemata.

3.1.2. Molekulares Rechnen

Die Zellbiologie schreibt den einzelnen Zellen die Fähigkeit zu, Informationen zu verarbeiten. So erfüllen proteinbasierte Schaltkreise die Funktion eines Nervensystems zur Steuerung von Einzellern. Die funktionelle Verknüpfung tausender Proteine in komplexeren Organismen gleicht einem neuronalen Netz, welches im Laufe der Evolution trainiert wurde. Bray [26] gibt einen Überblick über die informationsverarbeitenden Fähigkeiten von Proteinen in lebenden Zellen.

Die gesteuerte Durchführung von Berechnungen auf molekularer Ebene hat ihren Durchbruch mit dem Experiment von Adleman [2] erfahren und sich in den

folgenden Jahren als eigenständige Forschungsdisziplin etabliert. Seither sind eine Vielzahl Techniken entstanden, die zur gezielten Manipulation und Auswertung von Molekülen herangezogen werden können. Garzon und Deaton [47] beschreiben einige dieser Techniken und geben einen Überblick über das Forschungsgebiet. DNA ist dabei das am häufigsten zugrunde liegende Makromolekül, da es sich leicht duplizieren lässt und als Einzelstrang selbstständig sein Komplement wählt. Aber auch RNA findet in einigen Ansätzen Verwendung.

Der Vorteil von Berechnungen auf molekularer Ebene liegt in dem selbstorganisierten und parallelen Ablauf. Dieses ermöglicht massive Parallelität ohne zentrale Koordination. Die meisten Ansätze bestehen daher in einem Brute-Force-Vorgehen, bei dem parallel in einer Vielzahl potenzieller Lösungen nach der richtigen Lösung gesucht wird. Adleman zählt des Weiteren die Energieeffizienz zu den Vorteilen des molekularen Rechnens. Offensichtlicher Nachteil dieser Ansätze sind nach Ruben und Landweber [99] eine fehlende Wiederverwendbarkeit und eine hohe Spezialisierung der eingesetzten Makromoleküle, eine hohe Fehlerrate durch falsche Bindungen und die fehlende Automatisierung die zu einem hohen Zeitaufwand für die Experimente führt[1].

Stellvertretend für Systeme, die Berechnung auf molekularer Ebene durchführen, wird im Folgenden das Experiment von Adleman beschrieben, bei dem DNA Sequenzen als Informationsträger fungieren.

Adlemans Experiment

Anhand der Berechnung eines gerichteten Hamiltonpfads demonstriert Adleman [2] die Möglichkeit, Berechnung mittels DNA-Molekül-Interaktionen durchzuführen. Das Hamiltonpfad-Problem besteht in der Prüfung auf Existenz eines Pfades in einem gerichteten Graphen, der in einem markierten Knoten $v_{\text{in}} \in N$ startet und in einem zweiten, festgelegten Knoten $v_{\text{out}} \in N$ endet, so dass jeder Knoten der Knotenmenge N exakt einmal durchlaufen wird. Das Hamiltonpfad-Problem ist NP-vollständig.

Zur Durchführung der Berechnung auf molekularer Ebene setzt Adleman den in Alg. 3.1 skizzierten Algorithmus in Standardverfahren aus dem Bereich der DNA-Analyse um. Zunächst assoziiert er mit jedem Knoten i des Graphen ein zufällig[2] erzeugtes Oligonukleotid[3] O_i mit einer Länge von 20 Basen. Für jede Kante $i \rightarrow j$ wird ein ebenfalls 20 Basen langes Oligonukleotid $O_{i \rightarrow j}$ erzeugt, welches mit 10 Basen vom 3'-Ende von O_i, gefolgt von 10 Basen vom 5'-Ende von

[1] Adelman benötigte für die einfache Instanz des Hamiltonpfad Problems sieben Labortage. Allerdings dürft der zeitliche Aufwand nahezu unabhängig von der Größe der gewählten Probleminstanz sein.

[2] Adleman arbeitet auf einer relativ kleinen Probleminstanz mit 7 Knoten und 13 Kanten. Für größere Probleme sieht er in dieser Zuordnung zufälliger Oligonukleotide zu Knoten die Probleme der Fehlhybridisierung und der zu schwachen Bindung. Hier sind im Laufe der Jahre eigenständige Tools (vgl. Feldkamp u. a. [43]) entstanden, die sich auf die Erzeugung angemessener Sequenzen spezialisiert haben.

[3] Bei einem Oligonukleotid handelt es sich um einen kurzen DNA-Einzelstrang.

1. Erzeuge ein Menge zufälliger Pfade durch den Graphen (möglichst viele unterschiedliche).
2. Behalte die Pfade, die in v_{in} beginnen und in v_{out} enden.
3. Behalte die Pfade, deren Länge der Anzahl der Knoten entspricht.
4. Behalte die Pfade, die jeden Knoten mindestens einmal enthalten.
5. Ist die Menge nicht leer, so wurde ein Hamiltonpfad gefunden.

Algorithmus 3.1 : Nichtdeterministischer Algorithmus zur Lösung des Hamiltonpfad Problems. Der Pfad muss in v_{in} beginnen und in v_{out} enden. Nur wenn in Schritt 1 alle möglichen Pfade erzeugt wurden, folgt aus der leeren Menge in Schritt 5 die Nichtexistenz eines Hamiltonpfades.

O_j übereinstimmt. Ausnahmen bilden Kanten mit $i, j \in \{v_{\text{in}}, v_{\text{out}}\}$, in diesem Fall werden alle 20 Basen von $O_{v_{\text{in}}}$ bzw. $O_{v_{\text{out}}}$ übernommen. Abbildung 3.2 skizziert noch einmal die wichtigsten Elemente der Pfadcodierung.

Für die Umsetzung des ersten Schritts werden für jeden Knoten $i \in N \backslash \{v_{\text{in}}, v_{\text{out}}\}$ 50 Picomol[4] (pmol) des zu O_i komplementären Oligonukleotid $\overline{O_i}$ und für jede Kante $i \to j$ 50 pmol $O_{i \to j}$ gemischt. Die dann selbstständig stattfindende Hybridisierung, eine Anlagerung komplementärer Einzelstränge und die Ausbildung von Wasserstoffbrückenbindungen zwischen diesen, führt zu DNA Molekülen, die zufällige Pfade im Graph codieren. Im zweiten Schritt werden mittels der Polymerasekettenreaktion jene Stränge vervielfältigt, an die Oligonukleotide $O_{v_{\text{in}}}$ und $\overline{O_{v_{\text{out}}}}$ binden. Hierdurch vervielfältigen sich jene DNA-Moleküle exponentiell, die in v_{in} beginnende und in v_{out} endende Pfade codieren. Ist $|N|$ die Anzahl der Knoten im Graph, so werden im dritten Schritt mittels der Agarose-Gelelektrophorese jene doppelsträngige DNA Sequenzen selektiert, die sich im Bereich $|N| \times 20$ Basenpaare absetzen. Dieses sind jene Sequenzen, die einen Pfad der Länge $|N|$ codieren. Im vierten Schritt werden nun zunächst jene Moleküle isoliert, die als Teilsequenz das Oligonukleotid O_1 enthalten. Dieses geschieht mittels des Komplements $\overline{O_1}$, welches mit einem magnetischen Partikel verbunden wurde. Dieser Vorgang wird auf der jeweils isolierten Teilmengen für alle Oligonukleotide O_i mit $i \in N \backslash \{v_{\text{in}}, v_{\text{out}}\}$ wiederholt. Die Sequenzen in der zuletzt selektierten Teilmenge codieren Pfade, die alle Knoten durchlaufen.

Für den letzten Schritt vervielfältigt Adleman noch einmal die verbleibenden Moleküle mittels Polymerasekettenreaktion und analysiert anschließend mit der Gelelektrophorese ob Sequenzen verlieben sind, welche die erforderliche Länge

[4] 50 Picomol entsprechen ca. 3×10^{13} Oligonukleotiden.

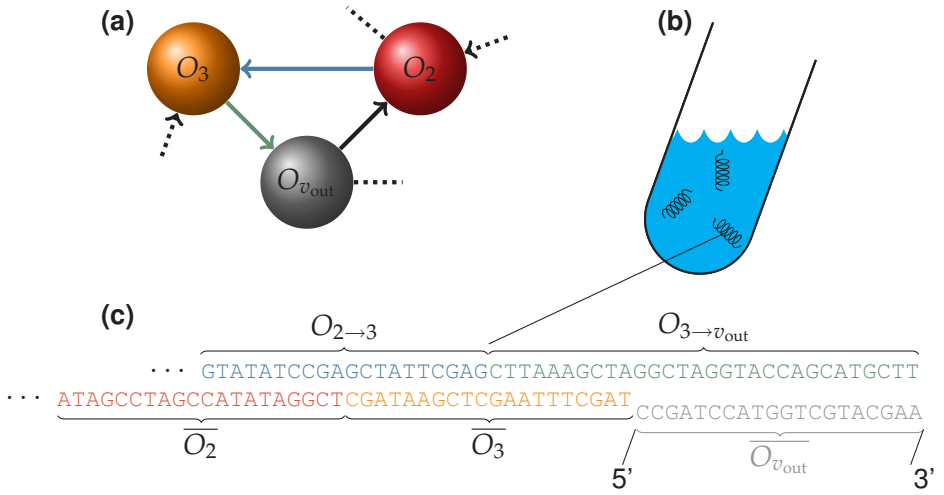


Abbildung 3.2.: Jedes DNA-Molekül (c), welches durch Bindung der Oligonukleotidstränge während des ersten Schritts (b) in Adlemans Experiment entsteht, codiert einen Pfad im Graphen (a).

besitzen. Ist dieses der Fall, so existiert ein Hamiltonpfad. Ansonsten existiert entweder kein solcher Pfad, oder er wurde im ersten Schritt des Algorithmus zufällig nicht erzeugt.

3.2. Spatial Computing

Sowohl die gängige Betrachtungsweise von Algorithmen als auch die meisten Programmiersprachen und heutigen Rechnerarchitekturen spiegeln Prinzipien wider, welche auf die erste erfolgreiche Computerarchitektur, den von-Neumann-Rechner, zurückgehen. Sequenzialität und die Abhängigkeit von der Zuverlässigkeit aller Komponenten sind die meist kritisierten Prinzipien. Während die zweite Kritik ihren Ursprung bei von Neumann [80] selbst hat, ist Backus [6] der bekannteste Kritiker der Sequenzialität, welche er als von-Neumann-Flaschenhals bezeichnet.

„Spatial Computing“ oder auch „Space-Oriented Computation“ ist eine junge Disziplin, die versucht, den Teil der Trends jenseits des von-Neumann-Modells zusammenzufassen, die den Raum (als Gegenstück zur Zeit) explizit als Bestandteil ihres Systems betrachten. Ein „spatial computing system“ definiert sich nach [111] wie folgt:

> A general purpose computing system where position matters for performance or functionality and is embedded in a space whose geometry is defined by communication cost.

Auch einige der zuvor vorgestellten Systeme, wie die Pfadberechnung von Adleman, verlagern einen Teil der Berechnungskomplexität von der Zeit in den Raum. Sie genügen aber zumeist nicht dem Anspruch allgemeiner Verwendbarkeit. Im

Folgenden werden zum einen zwei alternative Architekturen vorgestellt, die ihrer Berechnungen räumlich organisieren, zum anderen das eher theoretisch motivierte Membrane-Computing.

3.2.1. Rechnerarchitekturkonzepte

Field Programmable Gate Arrays (FPGAs) gelten als Kandidaten für die erste Generation der um eine „Raumkomponente“ erweiterten Computer. Daneben gibt es Ansätze wie Amorphous Computing oder BLOB[5] Computing, welche neue Rechnerarchitekturen propagieren, bei denen sich eine große Anzahl Daten verarbeitender Einheiten (DVE) zur Laufzeit dezentral organisieren, um parallel und asynchron Berechnungen durchzuführen und Daten zu speichern. Kommunikation zwischen ihnen ist lokal begrenzt. Bei entsprechender Organisation der DVEs ist die Parallelität nur durch den zur Verfügung stehenden Raum beschränkt, in dem die DVEs angeordnet sind.

Während die DVEs beim Amorphous Computing unregelmäßig im Raum verteilt sind und die Organisationsmechanismen durch Differenzierungsmechanismen biologischer Zellen motiviert werden, präsentieren die Autoren von BLOB Computing ein zweidimensionales Gitter mit DVEs und greifen für die räumliche Organisation auf physikalische Metaphern zurück. BLOB Computing geht einen Schritt weiter als Amorphous Computing, indem neben der Architektur und den Mechanismen zur Selbstorganisation der Komponenten ein Modell der Programmierung präsentiert wird. Bei diesem Modell enthalten die Algorithmen bereits Informationen über die funktionale Differenzierung der DVEs und den Zeitpunkt der Ausdifferenzierung.

Amorphous Computing

Abelson u. a. [1] erwarten in naher Zukunft die kostengünstige Produktion unzähliger kleiner Recheneinheiten, die sowohl technischer als auch biologischer Natur sein können. Als Masse bilden sie ein Medium[6] für „Amorphous Computing“, das sich durch mehrere Eigenschaften auszeichnet. Die DVEs, Partikel genannt, sind unregelmäßig angeordnet und kommunizieren asynchron in einer lokalen Nachbarschaft miteinander, z. B. elektronische Partikel über Funk oder organische Einheiten über Botenstoffe. Die Partikel besitzen keine Information über ihre Position. Sie können unter Umständen mit Sensorik ausgestattet sein oder in anderer Form mit ihrer Umwelt interagieren. Für sich genommen haben sie einen kleinen Speicher

[5] BLOB ist der englische Begriffe für Klumpen und wurde von Gruau u. a. als Bezeichner für die sich räumlich ausbildenden funktionellen Einheiten gewählt, die sich in Form und Größe unterscheiden.

[6] Die Form des Mediums ist dabei irrelevant. So ist ein mögliches Anwendungsszenario die „intelligente Farbe“: ausgestattet mit entsprechender Sensorik und integriert in einer Farbe könnten die Partikel als Brückenanstrich Verkehrs- und Windbelastungen sowie strukturelle Veränderungen melden oder in Gebäuden über die Wahrnehmung von Vibrationen die Anwesenheit von Personen anzeigen.

und eine geringe Rechenfähigkeit. Die Wahrscheinlichkeit eines Defekts ist hoch und muss explizit berücksichtigt werden.

Die Partikel sind identisch programmiert. Das in ihnen enthalten Programm muss zur Laufzeit für die Ausdifferenzierung sorgen, damit die Partikel durch kohärentes Verhalten gemeinsam die Problemlösung assemblieren. Amorphous Computing lässt sich hierbei von den biochemischen Mechanismen natürlicher Zellen inspirieren, die in Organismen definierte Strukturen und funktionale Einheiten bilden. Unter anderem stellen Abelson u. a. die Nachahmung einiger biologischer Prozesse vor, wie z. B. das Wachstum, seine Ausrichtung nach einem Stimulus (Tropismus) oder die gezielte Hemmung des Wachstums. Mit solchen Mechanismen gelingt es in der Simulation, Differenzierungsmuster auf planar angeordneten Partikeln zu erzeugen.

Ihr Ziel ist es, zu neuen Methoden für den Aufbau und die Programmierung von Rechnern zukommen. Sie sind dabei weniger an der Selbstorganisation an sich oder der Emergenz von Funktionalität interessiert, sondern an der gezielten technischen Nutzung solcher Verfahren.

BLOB Computing

Gruau u. a. [50] kritisieren an der heute vorherrschenden von-Neumann-Architektur, dass sie, neben der universellen Einsetzbarkeit, vor allem für eine kompakte Implementierung konzipiert wurde. Diese Kompaktheit wird auf Kosten der Ausführungszeit ermöglicht, indem auch unabhängige Teilaufgaben sequenziell abgearbeitet werden. Moderne Erweiterungen, wie Multi(core)prozessoren, Pipelines oder Sprungvorhersagen, reduzieren zwar die Kompaktheit zugunsten von mehr Parallelität, basieren aber weiterhin auf der von ihr geprägten Architektur. Gruau u. a. nutzen durch eine großzügig konzipierte Architektur und einem eigenen Programmiermodell explizit den Raum, um die auszuführenden Berechnungen in diesem zu verteilen. Es handelt sich bei der Architektur um ein reguläres zweidimensionales Gitter von Zellen, die die Fähigkeit besitzen, Daten zu speichern und diese zu verarbeiten. Jede Zelle kann mit den acht ihr angrenzenden Zellen kommunizieren.

Mehrere Zellen bilden einen BLOB und werden als eine datenverarbeitende Einheit betrachtet. Das Programm eines BLOBs, beschrieben durch einen Zustandsautomaten, ist unterteilt in sogenannte Partikel, von denen jede Zelle maximal einen enthält. Jedes Partikel ist verknüpft mit einem Zustandsübergang. Sind die Bedingungen für den Übergang erfüllt, so werden beim Zustandsübergang die im Partikel beschriebenen logischen und arithmetischen Operationen ausgeführt. Das Ergebnis wird ggf. nach außen kommuniziert. Innerhalb eines BLOB geschieht dieses durch Broadcasting. Da Kommunikation nur zwischen benachbarten Zellen möglich ist, reichen alle Zellen die Information bis an die BLOB-Grenzen weiter. Die Kommunikation zwischen BLOBs geschieht über spezielle (Link-)BLOBs, die sich mit den zu verbindenden BLOBs jeweils eine Zelle teilen, welche speziell markierte Nachrichten in den angrenzenden BLOB überträgt.

Ein Netzwerk von BLOBs entsteht durch eine Klasse von Operationen, welche die räumliche Struktur des Algorithmus spezifiziert und Teil ihres „Architektur freundlichen“ Programmiermodells ist. Dieses Modell integriert die im Algorithmus enthaltene Parallelität und verteilte Datenhaltung explizit in die Sprachsemantik. Die Operationen zur Netzwerkausbildung, welche z. B. die Duplikation und das Zusammenfassen von Einheiten beinhaltet, machen die Verknüpfung von Raum und Parallelität explizit.

BLOBs, die zur Laufzeit entstehen, müssen dynamisch platziert werden. Dieses geschieht durch Zuordnung von benachbarten Zellen, die Zuordnung kann zur Laufzeit wieder geändert werden. Die Anordnung der BLOBs im Raum ist entscheidend für die Performance des Systems, so sollten z. B. häufig kommunizierenden BLOBs nahe beieinander liegen, um die Signallaufzeit zu verbessern. Die hierfür verwendete Heuristik bedient sich eines physikalischen Modells und simuliert hierfür Elastizität, Drücke, usw.

Heutige Programmierkonzepte und Sichtweisen sind stark geprägt von der sequenziellen und zentral organisierten Arbeitsweise der von-Neumann-Architektur. Am Beispiel der von Gruau u. a. beschriebenen Sortieralgorithmen wird eine andere Sichtweise vorgestellt. Bei diese Sichtweise wird explizit die Nutzung des Raumes in den Algorithmus integriert. Ihre erste Realisierung ist in Abb. 3.3 dargestellt, er besteht aus zwei Phasen. Zunächst werden BLOBs für die Sortierung einer festen Anzahl von Werten gebildet. Diese dienen in der zweiten Phase zur Speicherung und Sortierung eingehender Werte. Abbildung 3.5a zeigt die Entwicklung des Systems und die anschließende Sortierung von Werten. Bei der zweiten Realisierung (Abb. 3.4 & 3.5b) geschehen Sortierung und Entwicklung simultan, wodurch die Sortierung einer vorher nicht bekannten Anzahl an Werten möglich und nur durch den zur Verfügung stehenden Raum beschränkt ist.

3.2.2. Membrane Computing

Membrane Computing will aus Erkenntnissen der Zellbiologie neue Ideen und Modelle für die Informatik gewinnen. Hierzu führt Păun [92] eine Klasse von teils Turing-vollständigen Modellen (P-Systeme) ein, die hauptsächlich für verteilte und parallele Berechnungen gedacht sind und in denen der Begriff der Membran eine zentrale Rolle spielt. Membrane trennen hierarchisch Multimengen von Objekten in einzelne Regionen voneinander ab. Neben diesen Objekten enthalten die Regionen Reaktionsregeln[7], die die Dynamik innerhalb der Membrangrenzen beschreiben. Diese Regeln definieren die benötigten Edukte, die erzeugten Produkte sowie die Kommunikation über Membrangrenzen hinweg. Dabei existieren unterschiedliche Kommunikationsformen, die sich in der Membrandurchlässigkeit unterscheiden. So ist einerseits die Kommunikation über die äußere Membran mit der nächst

[7] Păun spricht von Evolutionsregeln, es handelt sich um die Beschreibung chemischer Reaktionen und ist hier nicht mit dem Evolutionsbegriff zu verwechseln, wie er bei den evolutionären Algorithmen Verwendung findet.

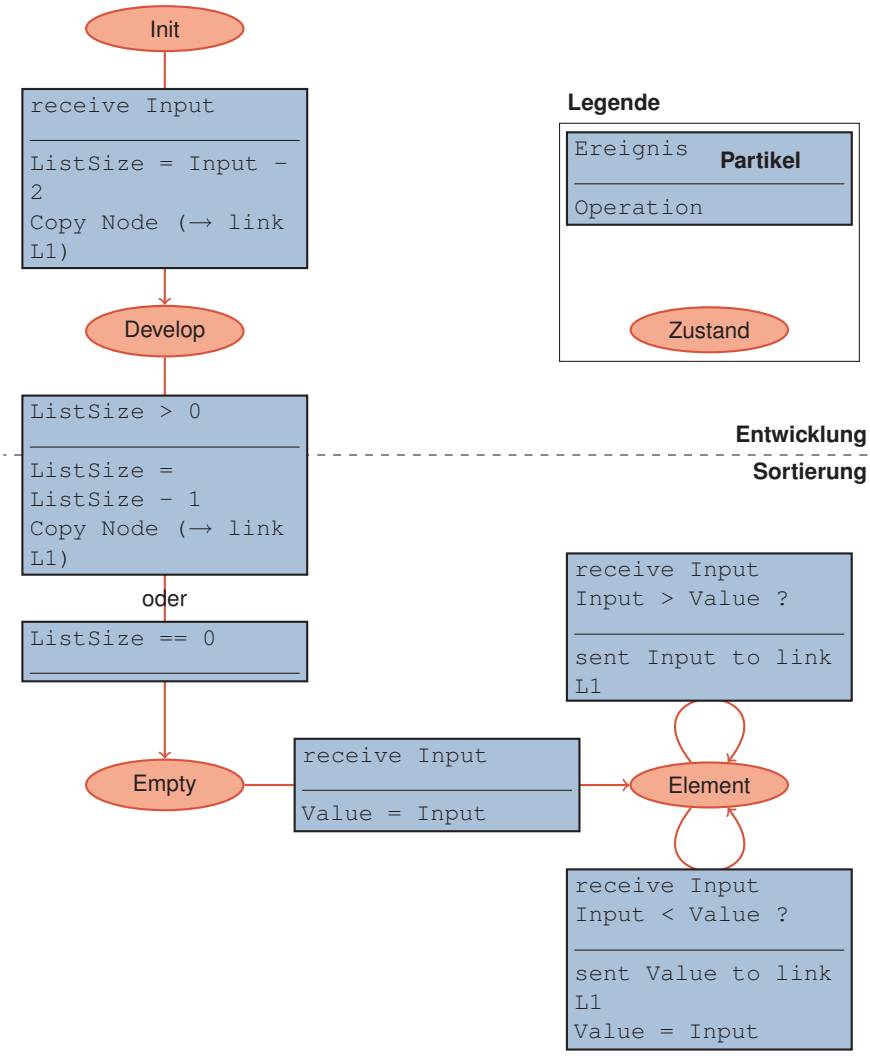


Abbildung 3.3.: BLOB-Computing-Automat zur Programmierung eines Sortieralgorithmus. Die Entwicklungsphase ist von der Sortierphase getrennt [50]

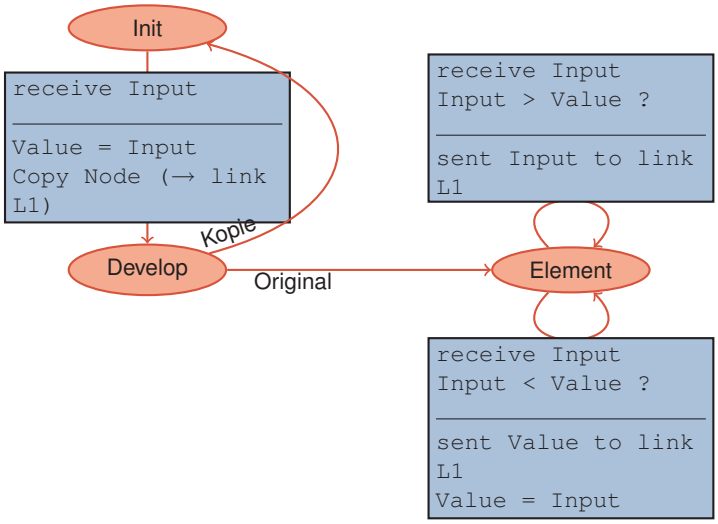
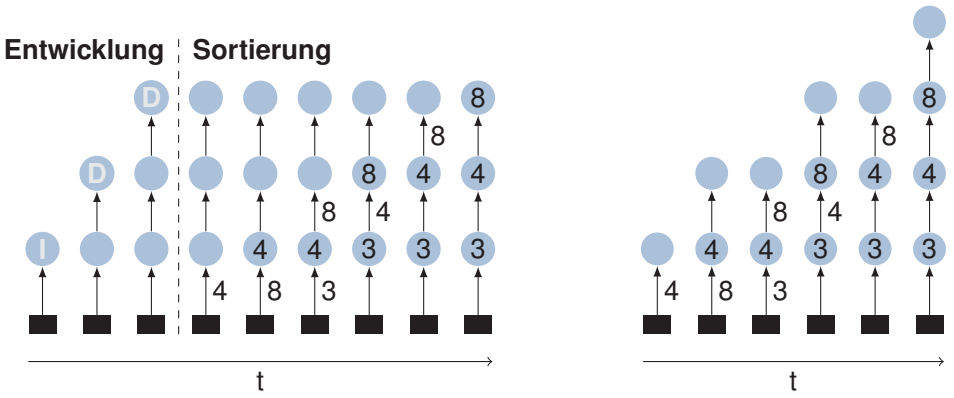


Abbildung 3.4.: Automat für die simultane Entwicklung und Sortierung beim BLOB-Computing. Die Symbolik entspricht der in Abb. 3.3 verwendeten [50]



(a) Separate Entwicklungsphase **(b)** Simultane Entwicklung

Abbildung 3.5.: Zeitlicher Verlauf der BLOB-Entstehung und Sortierung. Jeder Punkt stellt einen datenverarbeitenden BLOB in unterschiedlichen Zuständen dar. Enthält der Punkt einen Wert, so befindet sich der BLOB im Zustand „Element“, enthält er keinen Wert, so ist der Zustand „Empty“, ‘I’ steht für „Init“ und ‘D’ für „Develop“. Der linke Verlauf entspricht dem unter Verwendung des in Abb. 3.3 skizzierten Automaten, der rechte Verlauf entspricht dem des Automaten aus Abb. 3.4.

höheren Hierarchieebene möglich, andererseits kann über die inneren Membranen mit den enthaltenen Regionen kommuniziert werden.

Die Menge der Objekte aller Regionen beschreiben die Konfiguration eines Systems. Eine Reaktion verändert die Konfiguration und erzeugt so einen Übergang. Eine Sequenz von Übergängen erzeugt die Berechnung. Ist die Berechnung erfolgreich, so hält das System, da keine Reaktion der vorhandenen Objekte mehr möglich ist. Resultate lassen sich durch Zählen von Objekten in den unterschiedlichen Bereichen ermitteln oder sind in der Sequenz jener Objekte codiert, die im Laufe der Ausführung die äußere Membran verlassen haben.

Păun [93] sieht unter anderem in Membrane Computing ein „bio-inspired framework for distributed parallel processing of multisets“ und macht deutlich, dass er eine Umsetzung eher als Hardwarelösung, denn als biologisches oder chemisches Experiment sieht. Um die Lösung komplexer Probleme effizient zu berechnen, bedient sich auch Membrane Computing des Raums (Păun und Rozenberg [95]):

> Systems with an enhanced parallelism are able to trade space for time and solve in this way (at least in principle), by making use of an exponential space, intractable problems in a feasible time.

Dieses setzt voraus, dass es eine Möglichkeit gibt, mit linearem Zeitaufwand exponentiell viel Raum zu nutzen. Membrane Computing versucht dieses in der Theorie über Mechanismen wie Membranteilung und -erzeugung.

Teil II.

Algorithmische Chemien

4. Algorithmische Chemien

> [...] getting the right answer may be the wrong idea; [...] the fundamental question is how to abstractly structure systems so we get the acceptable answers, with high probability, even in the face of unreliability.
>
> *(Abelson u. a. [1])*

4.1. Motivation

Einfluss der von-Neumann-Architektur

Durch Eleganz und Einfachheit hat das von-Neumann-Prinzip Jahrzehnte lang weit mehr als nur die Disziplin der Computerarchitekturen in der Informatik geprägt. Backus [6] etwa bezeichnet konventionelle Programmiersprachen als komplexe von-Neumann-Rechner und klassifiziert sie nach ihrem „intellektuellen Vater“ als von-Neumann-Sprachen. Variablen imitieren in ihnen die Speicherzellen, Zuweisungen beinhalten die Speicheroperationen und die Arithmetik. Die Kontrollstrukturen stellen bedingte Sprünge dar. Auf einer höheren Abstraktionsebene ist auch die gängigen Betrachtungsweise von Algorithmen geprägt von der von-Neumann-Architektur. Cormen u. a. [35] schreiben etwa in ihrem Standardwerk über Algorithmen:

> Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

Der hier beschriebene sequentielle Ablauf hat seien Ursprung in der programmzeigergesteuerten von-Neumann-Architektur. Ihre Beschreibung eines Algorithmus ist nicht die einzig mögliche. Dies wird deutlich bei Betrachtung von Definitionen unter Berücksichtigung anderer Architekturen oder Modelle. So schreiben Gruau u. a. [50] über Algorithmen beim BLOB-Rechner (siehe auch Kap. 3.2.1):

> [...] the blob computing model specifies the algorithm spatial structure. [...] By expressing its algorithm as an automation using Cellular Encoding, the program provides implicit parallelism to the architecture. [...] As a result, a program is no longer a sequence of instructions with a single program counter but a large set of automata that evolve concurrently.

Păun und Rozenberg [95] (vgl. Kap. 3.2.2) beschreiben Berechnungen wie folgt:

> In each time unit a transformation of a configuration of the system takes place by applying the rules in each region, in a nondeterministic and maximally parallel manner. [...] In this way, one gets transitions between the configuration of the system. A sequence of transitions is called computation. A configuration is halting, if no rule is applicable in any region [...].

Der Algorithmus ist bei Păun und Rozenberg in der Beschreibung des Membransystems enthalten. Diese besteht, neben der Membranstruktur und den anfänglich in den Membranen enthaltenen Elementen, vor allem aus den Reaktionsregeln, welche die Transformationen beschreiben.

Bei der genetischen Programmierung fungieren Individuen, je nach Abschnitt des Evolutionszyklus (vgl. Abb. 1.1), sowohl als reine Datenstruktur als auch als algorithmische Beschreibung einer Berechnung. Die Individuen werden für die Durchführung der Berechnung in einer Datenstruktur repräsentiert, die ihre deterministische Ausführung in vordefinierten sequenziellen Schritten ermöglicht. Der Einfluss des von-Neumann-Prinzips reicht hier also bis auf die Ebene der Datenstrukturen und beeinflusst damit alle Phasen der Evolution. Beispielsweise ist die Rekombination so realisiert, dass die Sequenz der Berechnungsschritte so weit wie möglich beibehalten wird, indem nur an wenigen Stellen ein Übergang zwischen den Sequenzen der Eltern stattfindet.

Kritik an der von-Neumann-Architektur

Die von-Neumann-Architektur ist im Laufe der Jahre nicht unkritisiert geblieben, wobei im Folgende zwei Formen der Kritik von besonderer Bedeutung sind:

1. die Sequenzialität in der Ausführung sowie im Speicherzugriff und
2. der Zwang zu deterministischen Verhalten aller Komponenten.

Beim heutigen Rechner, als Vertreter der von-Neumann-Architektur, wird die Zeit durch den Systemtakt in Abschnitte unterteilt und damit die Anzahl möglicher Ereignisse in einem Zeitraum von vorneherein beschränkt. Der Programmzeiger dient der sequenziellen Abarbeitung eines Programms. Paralleles Rechnen erfordert zusätzlichen Aufwand, etwa für das Scheduling. Zudem erfordert die Umsetzung der meist sequenziell formulierten Algorithmen auf parallele oder verteilte Systeme ein hohes Maß an Synchronisation auf Kosten der Effizienz. Die sequenzielle Ausrichtung der von-Neumann-Architektur zeigt sich neben der eigentlichen Ausführung auch in dem wortweisen Zugriff auf den Speicher. Diese Form des Zugriffs bezeichnet Backus [6] als von-Neumann-Flaschenhals.

Zudem wird ein hoher Aufwand betrieben, um ein deterministisches Verhalten sicherzustellen. So werden Informationen binär codiert und bei der Übertragung durch zusätzliche Informationen zur Fehlererkennung und/oder -korrektur ergänzt. Der Ursprung dieser zweiten Kritik geht zum Teil auf von Neumann selbst

zurück und wird von jenen weiter ausgeführt, die sich mit alternativen Rechnerarchitekturen befassen. Schon von Neumann [80] beschäftigt sich mit Fehlern und nichtdeterministischen Verhalten in logischen Schaltungen. Er fordert, dass ein Versagen in logischen Schaltungen nicht als irreführender Fehler betrachtet werden sollte, sondern als essentieller Teil des gesamten Prozesses. Vorbild seiner Überlegungen ist dabei unter anderem das menschliche Gehirn, welches als kompliziertes Schaltnetzwerk mit langen Reizreaktionsketten auch unter Fehlleistungen einzelner Komponenten zuverlässige Resultate des Ganzen erzielt.

Nichtdeterministisches Verhalten in informationsverarbeitenden Systemen der Gegenwart und Zukunft

Die Weiterentwicklung heutiger Technologie könnte zu Systemen führen, in denen die Komponenten keine exakt bestimmbares Verhalten mehr aufweisen. So führt die weitere Reduzierung der Größe von traditionellen, auf Halbleitern basierenden Transistoren dazu, dass die einzelnen Transistoren vermehrt in ihren technischen Eigenschaften, wie etwa der Performance, variieren [91]. Somit sind auch die hieraus aufgebauten Komponenten mit Unsicherheit behaftet und für die klassischen deterministischen Ansätze beim Chipdesign zunehmend ungeeigneter.

Zum Teil wird die Unsicherheit auch gezielt in Kauf genommen. So steigt der Energiebedarf für das Speichern von Zuständen exponentiell mit der Wahrscheinlichkeit, dass die Zustände korrekt bestimmt werden. Dabei ist diese Wahrscheinlichkeit klassischerweise nahezu eins. Ansätze von Palem [89] nutzen diese Erkenntnis bei der Entwicklung stromsparender Coprozessoren für die Signalverarbeitung oder für probabilistische Algorithmen. Die hier verwendeten probabilistischen Bits (PBits) nehmen, wie klassische Bits, nur zwei unterschiedliche Zustände an. Diese können aber nur mit einer Wahrscheinlichkeit kleiner eins korrekt bestimmt werden.

Auch bei der Konzeption von Computersystemen der Zukunft spielt die Unzuverlässigkeit bei der Ausführung schon heute eine essenzielle Rolle. So handelt es sich beim Quantum Computing potenziell bei jedem Gatter um eine analoge Operation, da der Zustand einer Quantum Superposition von einem kontinuierlichen Wert abhängt. Damit ist nach Shor [105] die tatsächliche physikalische Realisierung mit einer gewissen Ungenauigkeit behaftet. Als Beispiel nennt er die Rotation eines Quantumbits um einen bestimmten Winkel, ein typisches Quantengatter. Hierbei wird der Zielwinkel naturgemäß nicht exakt erreicht. Auch die einzelnen Schritte beim DNA-Computing sind mit Unsicherheit behaftet. Beispielsweise kann im Experiment von Adleman [2] (Kap. 3.1.2) die Bindung der Oligonukleotide nicht garantiert werden[1].

Letztendlich arbeiten beim Amorphous Computing (Kap. 3.2.1) per Definition eine große Anzahl unzuverlässiger, asynchroner Einheiten miteinander, die zudem

[1] Abhilfe schafft hier die massive Parallelität, die nicht nur für die zeitgleiche Betrachtung möglichst vieler möglicher Lösungen sorgt, sondern auch dafür, dass die richtige Lösung häufig genug erzeugt wird, um messbar zu sein.

in nicht vordefinierter, irregulärer und zeitlich variierender Weise untereinander verbunden sind. Für Abelson u. a. [1] gehen mit einem solchen System zwangsläufig fundamentale Änderungen in der Konstruktion und Programmierung einher. Sie suchen hierfür nach geeigneten Programmiermethoden und Organisationsprinzipien. Anstelle von Redundanz zur Fehlererkennung und dem Austausch defekter Komponenten proklamieren sie, dass allein das Erlangen der richtigen Antwort der falsche Weg ist. Viel mehr gilt es, ein solches System so zu strukturieren, dass akzeptable Ergebnisse auch dann mit hoher Wahrscheinlichkeit erlangt werden, wenn es zum Versagen einzelner Komponenten kommt.

Zum einen macht die Aussicht auf die Verfügbarkeit alternativer Systeme in nicht all zu ferner Zukunft die Evolution von Lösungen auf diesen Systemen und den damit verbundenen Erkenntnisgewinn interessant. Zum anderen geht mit diesen Systemen eine andere Betrachtungsweise von Algorithmen einher, und damit der Wunsch nach alternativen Repräsentationen und angemessenen Suchoperatoren in der genetischen Programmierung, die selber bisher stark geprägt ist vom von-Neumann-Prinzip. Die oben betrachteten alternativen Systeme verhalten sich, durch Nichtdeterminismus, paralleles und teils asynchrones Agieren der Komponenten sowie dezentraler Organisation, wesentlich naturanaloger als klassische Systeme. Mit den von ihnen inspirierten Repräsentationen ist auch die Hoffnung auf eine Steigerung der Evolutionsfähigkeit verbunden.

Evolutionsfähigkeit

Evolutionsfähigkeit Der Begriff Evolutionsfähigkeit stammt ursprünglich aus der Biologie. *Die Evolutionsfähigkeit ermöglicht einem Stamm (Phylum), auf veränderte Selektionsbedingungen zu reagieren. Hierfür muss für seine Individuen zum Zwecke der Anpassung die Möglichkeit bestehen, in überlebensfähiger und vererbbarer Weise zu variieren.* Die Vielfalt der Arten ist das heute beobachtbare Resultat erfolgreicher Diversifikation. Als Grundlage evolutionsfähiger Systeme gilt eine geeignete Integration von Flexibilität und Robustheit (Conrad [34], Kirschner und Gerhart [61]). So muss z. B. der Entwicklungprozess stabil gegenüber äußeren Einflüssen, aber im beschränkten Maße anfällig für genetische Variation sein.

Unterschiedliche Mechanismen in natürlichen Systemen werden für die Erreichung dieser – zunächst widersprüchlich erscheinenden – Ziele als wichtig erachtet. Offensichtlichster Vertreter einer gewissen Robustheit ist der *Bauplan*, der jeder Spezies zugrunde liegt. Die heute bekannte Artenvielfalt entstand erst nach Etablierung einiger weniger dutzend Stämme mit jeweils eigenem Bauplan. Beipiele für Stämme aus dem Tierreich sind die Weichtiere oder Gliederfüßer. Die Hox-Gene gelten als wichtiger Teil eines Bauplans und regulieren die *räumliche Separation* bei der Expression. Von ihnen ist bekannt, dass sie im Laufe der Evolution konstant geblieben sind. Die durch sie erzeugte räumliche Trennung ermöglicht erst die weitgehend unabhängige und eigenständige Entwicklung verschiedener Strukturen und Komponenten eines Organismus (vgl. Mayr [74]). Neben der so entstandenen Steigerung der Robustheit trägt diese Aufteilung aber auch maßgeblich zur Flexibili-

tät bei, indem sie die Auswirkungen mutativer Änderungen auf den Phänotypen in ihrem Ausmaß beschränkt, wodurch die Wahrscheinlichkeit einer tödlichen Änderung sinkt. Eine nur *schwache Verflechtung* von Komponenten trägt als zusätzlicher Mechanismus weiter zur Begrenzung der Auswirkungen bei.

Andere Formen der Separation, die im Grunde alle die Variation beschränken und dadurch erst handhabbar machen, finden sich bei der *Zelldifferenzierung* oder bei der Unterteilung des Genoms in unabhängig voneinander exprimierter Gene. Hinzu kommen weitere identifizierte Mechanismen zur Steigerung der Evolutionsfähigkeit. Zum einen ist dieses die Fähigkeit, *erforschendes Verhalten* mittels epigenetischer Variation und Selektion zu zeigen. Zum anderen sind dieses Mechanismen wie *Pufferung*, die das Ausmaß einer Änderung reduzieren und somit graduelle Änderungen ermöglichen, und *Redundanzen*, die die alte Funktionalität schützen, während neue entstehen. Aus den letzten beiden Mechanismen leitet Conrad auch die Bedeutung von *Neutralität* für die Evolution ab.

Evolutionäre Suche und Struktur

Conrad [34] hält fest, dass Suche und Struktur untrennbar miteinander verknüpft sind. Suche kann erfolgen, weil die Struktur geeignet ist – die Struktur ist geeignet, weil die Suche zu ihr geführt hat. Bei der natürlichen Evolution wurde das darwinistische System während des Präkambriums durch Etablierung der Baupläne und anderer Mechanismen so adaptiert, dass die Organisationsstruktur der Stämme und der dazugehörigen Arten der Evolution dient. Das bedeutet für ihn, dass sie sich in einer Fitnesslandschaft befinden, die sich für „Hill Climbing“, sprich die Anpassung in Einzelschritten, eignet.

Conrad [32] spricht von einem Trade-off zwischen Programmierbarkeit und Evolutionsfähigkeit und hält herkömmliche strukturierte Programmierung und Evolution für unvereinbar. Einen Weg, die Evolutionsfähigkeit von Computerprogrammen zu steigern, sieht er in der Integration eines Entwicklungsprozesses und anderer Mechanismen, wie der Redundanz. Dieses soll in den Programmen kleine (graduelle) Änderungen ermöglichen, die ihm für formale Systeme nicht realisierbar erscheinen. Molekulare Rechner als gänzlich andere Computerarchitektur hält er daher für besonders geeignet[2]. Später konkretisiert Conrad [34] seine Kritik an der Evolutionsfähigkeit gängiger Repräsentationen von Programmen. Vor allem die Kontextabhängigkeit ihrer Bestandteile machen sie für Mechanismen der Evolution, wie etwa der Rekombination, unbrauchbar. Darstellungen in Form regelbasierter Programme hält er für geeigneter, da ihnen Regeln einzeln entfernt und hinzugefügt werden können. Damit besitzen die Regeln eine Unabhängigkeit, die sich zum Teil auch in der Organisation genetischer Information wiederfindet

[2] Er stellt fest, dass dieses unter hohem Simulationsaufwand auch von einem von-Neumann-Rechner geleistet werden kann (Turing-Church-These). Voraussetzung für eine solche Simulation ist aber, dass Struktur und Funktion der im Molekularrechner enthaltenen Proteine aus der Primärstruktur ermittelt werden können.

und dort ermöglicht, dass Gene sich unabhängig voneinander entwickeln und zu nützlichen Eigenschaften kombiniert werden.

Gemeinsame Merkmale

Obwohl sich die Alternativen zur von-Neumann-Architektur sowohl untereinander, als von den natürlichen Systemen stark unterscheiden, lassen sich doch Gemeinsamkeiten der Systeme definieren. Diese sind (asynchrone) Parallelität, Nichtdeterminismus und die Notwendigkeit zur Selbstorganisation der Komponenten.

Parallelität, Nichtdeterminismus und die Notwendigkeit zur Selbstorganisation der Komponenten sind Eigenschaften, die auch in Systemen aus der ALife-Forschung von großer Bedeutung sind, da es sich dabei um Merkmale handelt, die auf allen Ebenen lebendiger Systeme anzutreffen sind (vgl. Abb. 2.1). Die künstliche Chemie ist die Teildisziplin der ALife-Forschung, die hier von Bedeutung sein wird. Sie ist häufig motiviert von Prozessen der chemischen, präbiotischen Evolution, welche zur Entstehung der für Lebewesen notwendigen, komplexen, organischen Moleküle geführt hat. Sie stellt damit die Frage nach der „arrival of the fittest“ (Fontana und Buss [45]). Diese Frage wird durch die synthetische Evolutionstheorie nicht geklärt. Die hier gewonnenen Erkenntnisse über Organisationsprozesse fließen ein in die Entwicklung informationsverarbeitender, selbstorganisierender Systeme. Einige der Systeme wurden bereits in Kap. 3.1 beschrieben und bieten eine erste Anregung für die Bestimmung einer Ersatzarchitektur und der Repräsentation einer Lösung auf dieser.

Die meisten künstlichen Chemien sind für eine spezielle Aufgabe von Hand konstruiert worden. Damit beantworten sie nicht die Frage, wie Algorithmen, jenseits der betrachteten Anwendung, in chemischer Form formuliert werden können. Auch intuitiv richtig erscheinende Reaktionsschemata müssen aufgrund der Dynamik nicht zum gewünschten Ergebnis führen, ein Beispiel befindet sich in Kap. 3.1.1. Conrad [33] stellt fest, dass solche Systeme u. U. nicht im klassischen Sinn programmiert werden können. Eine hierfür notwendige vollständige Beschreibung bedarf der Reduzierung der Komponenten und Interaktionen und als Folge eine Reduzierung der Effizienz. Universalität, Effizienz und Evolutionsfähigkeit sind seiner Meinung nach widersprüchliche Eigenschaften. Zauner [118] nennt neuronale Netze als Beispiel für Systeme ganz ohne Programmierbarkeit, und vermutet für biomolekulare Systeme in den Evolutionsstrategien eine mögliche Methode, um gewünschtes Verhalten zu erzeugen.

4.2. Chemie

In dem Kontext künstlicher Chemien liefert das Konzept des Reaktors den Rahmen für die im Folgenden verwendete Ersatzarchitektur mit den gewünschten Merkmalen. Die Ersatzarchitektur wird zunächst genauer spezifiziert. Dieses ge-

schieht unter Berücksichtigung der linearen genetischen Programmierung (LGP) als späteres Referenzsystem. Dabei wird zuerst der Programmzeiger sowohl bei der Ausführung, als auch bei der Rekombination in einem LGP-System explizit gemacht und anschließend so verändert, dass er ein stochastisches Verhalten aufweist. Dieses führt zur Auflösung der sequenziellen Reihenfolge von Instruktionen, wie sie für gewöhnlich mit Algorithmen assoziiert wird. Programme gleichen nunmehr Multimengen von Instruktionen, die unter Umständen bei jeder Ausführung eine andere Verhaltensvariante liefern. Instruktionen interagieren während der Ausführung miteinander, indem sie das Resultat einer Instruktion nehmen, transformieren und an die nächste Instruktion weiterreichen. Die Organisation geschieht über Muster in Form der Adressen von Ein- und Ausgabeargumenten. Die Instruktionsmultimenge wird im Folgenden als *algorithmische Chemie* bezeichnet, wobei dieser Ausdruck als Sammelbegriff für jede Form von künstlicher Chemie verstanden wird, die einen Algorithmus codiert.

Die algorithmischen Chemien werden mittels genetischer Programmierung bestimmt. Die Evolution von Lösungen ist möglich, da auch Multimengen von Instruktionen, die in zufälliger Reihenfolge ausgeführt werden, eine Semantik haben. Sie zeigt sich bei Betrachtung des Datenflusses. Bestimmte Speichereinheiten, im Folgenden als Moleküle bezeichnet, dienen als Ein- und Ausgabeargument der Instruktionen, die im Kontext algorithmischer Chemien auch Reaktionen genannt werden. Wenn ein Molekül zum einen zur Ausgabe, zum anderen zur Eingabe dient, so sind die beteiligten Reaktionen verbunden und ein potenzieller Datenfluss zwischen diesen beiden hergestellt. Es ist diese Datenflusslogik, die dem Einfluss der Evolution unterliegt. Die explizite Nichtdeterminiertheit der genetischen Programmierung wird mit der impliziten Nichtdeterminiertheit algorithmischer Chemien zusammengeführt und trägt der mit ihr behafteten Unsicherheit Rechnung.

4.2.1. Moleküle, Reaktionen und Reaktor

Losgelöst vom eigentlichen Lehrverfahren wird zunächst die zugrunde liegende Chemie betrachtet. Die Repräsentationen von Information als Molekül im Bereich künstlicher und natürlicher, informationsverarbeitender Chemien sind zumeist sehr problemspezifisch konstruiert, z. B. als sortierte Teilsequenzen oder Pfade in einem Graphen (vgl. Kap. 3.1). Andere künstliche Chemien, wie die Primzahlen generierende Chemie von Banzhaf u. a. [14], verwalten Informationen in universeller Granularität. So sind Moleküle bei der Primzahlenchemie beliebige ganzzahlige Werte, die Regel

$$s_1 + s_2 \rightarrow s_3 = \frac{s_1}{s_2}, \text{falls } s_1 \bmod s_2 = 0 \text{ und } s_1 \neq s_2$$

führt bei wiederholter Anwendung dazu, dass sich der Anteil der Primzahlen im Reaktor erhöht. Allerdings lassen sich in einem solchen System, in dem nur eine Molekülart existiert (hier Werte aus $\mathbb{N}^+$), die anzuwendenden Regeln nicht anhand der benötigten Eduktarten unterscheiden. Dies beschränkt die möglichen

Freiheitsgrade für die Entstehung von Selbstorganisationsmechanismen. Im Fall der Primzahlen generierenden Chemie gibt die Bedingung eine Relation an, in der die Molkeüle zueinander stehen müssen.

Molekül *Moleküle in der algorithmischen Chemie befinden sich in einem Zustand, der durch einen Wert aus* $\mathbb{R}$ *beschrieben wird. Dieser Wert entspricht einer Eingabe an die Chemie, einem Zwischenergebnis oder einem konstanten Wert. Jedes Molekül ist bijektiv einer Molekülart zugeordnet, über die es adressiert wird.* Durch diese Definition kann eine Chemie auf unterschiedlichen Molekülarten arbeiten, deren mögliche Zustände hinreichend problemunspezifisch sind. Die Anzahl der Moleküle und damit die Größe des Reaktors ist konstant. Aufgrund der bijektiven Zuordnung von Molekül zu Molekülart wird im Folgenden vereinfacht von Molekülen gesprochen.

In ihrer globalen Verfügbarkeit gleicht die Darstellung Ansätzen, die mit implizit modellierten Molekülen arbeiten. Der Nachteil ist, dass diese Chemien kein konstruktives Verhalten bzgl. existierender Datenstrukturen zeigen können. So können Informationen nicht in einer Datenstruktur verknüpft werden, die zu einem späteren Zeitpunkt ihre gemeinsame Adressierung erlaubt. Diese wäre zum Beispiel bei der Verknüpfung in einer Liste der Fall. Konstruktivität existiert hier vielmehr dann, wenn in den Molekülen ein Zustand erzeugt wird, der vorher in diesen nicht zu beobachten war.

Reaktion *Eine Reaktion beschreibt die mögliche Zustandsänderung eines Moleküls. Der neue Zustand ist dabei abhängig von einer Funktion f und ggf. den Zuständen anderer Moleküle der Chemie.* In Anlehnung an die künstlichen Chemien wird eine Reaktion r in der algorithmischen Chemie wie folgt beschrieben:

$$r : \alpha_1 + \cdots + \alpha_n + \beta \rightarrow \alpha_1 + \cdots + \alpha_n + (\beta = f(\alpha_1, \cdots, \alpha_n)). \tag{4.1}$$

Edukt *Die Menge der an einer Reaktion beteiligten Moleküle werden als Edukte der Reaktion bezeichnet.* Die Edukte der Reaktion 4.1 sind die Moleküle $\alpha_1, \cdots, \alpha_n$ sowie das Molekül β. Während $\alpha_1, \cdots, \alpha_n$ nur als Katalysatoren dienen und somit im Anschluss an die Reaktion wieder in ihrem alten Zustand vorliegen, ändert sich der Zustand

Produkt des Moleküls β. *Moleküle, die an einer Reaktion beteiligt sind und deren Zustand sich durch Ausführung der Reaktion ändert, werden als Produkt der Reaktion bezeichnet.* Die Reaktion 4.1 wird im Folgenden auch kürzer als

$$r : \beta \leftarrow f(\alpha_1, \cdots, \alpha_n)$$

geschrieben. Für die Funktion f finden je nach Problem unterschiedliche arithmetische, trigonometrische und boolesche Funktionen aus Tab. 5.1 Verwendung.

Algorithmische Chemie *Eine algorithmische Chemie wird dann beschrieben durch eine Multimenge $\mathcal{R}$ von Reaktionen und den Zuständen der verwendeten Moleküle.*

4.2.2. Dynamik

Eine gängige Beschreibung der Dynamik künstlicher Chemien über die Molekülkollision ist in Alg. 2.1 wiedergegeben. Dabei wird eine bestimmte Anzahl Moleküle

gegeben : Moleküle A, Reaktionsmultimenge $\mathcal{R}$ und Terminierungskriterium
1 **repeat**
2 | $r \leftarrow$ `wähle zufällig(`$\mathcal{R}$`);`
3 | `ausführen(`r, A`);`
4 **until** `terminate();`

Algorithmus 4.1 : Vereinfachte Darstellung des für die Dynamik algorithmischer Chemien zuständigen Algorithmus. Reaktionen werden solange aus der Menge (evolvierter) Reaktionen zufällig gezogen und auf der Menge der Moleküle A ausgeführt, bis das Terminierungskriterium erreicht wurde.

dem Reaktor entnommen, und von allen Reaktionen, deren Edukte durch die entnommenen Moleküle abgedeckt werden, wird eine Reaktion zufällig gewählt und ausgeführt. Die relative Häufigkeit der Moleküle nimmt somit Einfluss auf die Dynamik des Systems. Auch im oben definierten Reaktor sind die Moleküle explizit modelliert, allerdings per Definition mit genau einem Molekül pro Molekülart. Da somit die Dynamik nicht aufgrund veränderter Molekülkonzentrationen variiert und eine Abhängigkeit der Ausführungshäufigkeit allein von der Arität einer Funktion (der Anzahl benötigter Edukte) nicht gewünscht ist, wird der Algorithmus zur Ausführung der algorithmischen Chemie angepasst.

Die veränderte Variante ist in Alg. 4.1 skizziert. Dabei wird bis zum Erreichen des Terminierungskriteriums in jedem Schritt eine der Reaktionen zufällig gezogen und zur Ausführung gebracht. Die Reaktionen befinden sich in einer Multimenge, denn jede Reaktion kann mehrfach vorhanden sein. Da sie gleichverteilt gezogen werden, bestimmt sich ihre durchschnittliche Ausführungsfrequenz über ihre relative Häufigkeit in der Chemie. Während die Anzahl der Moleküle je Molekülart jeweils auf eins fixiert ist, wird ihre Reaktivität durch ihre Verwendung in den Edukten der Reaktionen bestimmt, d. h. je häufiger ein Molekül in den Edukten der Reaktionen Verwendung findet, desto häufiger ist es an einer Reaktion beteiligt. Eine Reaktion ist nun ohne Bedingung anwendbar, muss aber nicht zu einer Änderung des Systemzustands führen. Ändern sich zwischen den wiederholten Ausführungen einer Reaktion $r : \beta \leftarrow f(\alpha_1, \cdots, \alpha_n)$ die Edukte $\alpha_1, \cdots, \alpha_n$ nicht, so ändert sich bei einer deterministischen Funktion f auch der Zustand von β aufgrund der Reaktion nicht. Die Reaktion ist dann *elastisch*.

5. Genetische Programmierung einer Algorithmischen Chemie

Eine problemspezifische algorithmische Chemie wird mittels eines evolutionären Algorithmus, der *genetischen Programmierung*, erlernt. Die Lerngüte der Evolution algorithmischer Chemien wird mit einem Referenzsystem verglichen. Hierzu wird lineares GP als klassisches, evolutionsbasiertes maschinelles Lernverfahren herangezogen. Die Abweichung der beiden Ansätze voneinander wird auf das Wesentliche reduziert, um die Ursache etwaiger Unterschiede hierauf zurückführen zu können.

Die Abbildung 5.1 skizziert die wichtigsten Stationen der genetischen Programmierung, auf die in den weiteren Unterkapiteln genauer eingegangen wird. Ausgangspunkt ist eine Initialpopulation von λ Individuen, die jeweils eine zufällig erzeugte algorithmische Chemie enthalten. Diese werden auf einer Trainingsmenge zur Ausführung gebracht und mittels einer Fitnessfunktion bewertet. Die μ besten Individuen bilden die Eltern der ersten Generation. Sie erzeugen λ Nachkommen. Mit der *Rekombinationswahrscheinlichkeit* p_c geschieht dieses durch Rekombination zweier Individuen, mit der Wahrscheinlichkeit $(1 - p_c)$ durch Reproduktion eines Individuums der Elternpopulation. Im Anschluss werden die so erzeugten Nachkommen mutiert, wobei jede evolvierte Information mit der *Mutationswahrscheinlichkeit* p_m verändert wird. Die Nachkommen werden wiederum ausgeführt und bewertet. Wenn das Terminierungskriterium nicht erreicht ist, wird die Menge der μ besten Nachkommen als Elternpopulation der nächsten Generation gewählt.

5.1. Individuen

Informationsträger sind die Individuen, in ihnen müssen

- die Reaktionsmultimenge,
- die Zustände der konstanten Moleküle,
- und die Wahl des Moleküls, dessen Zustand als Ergebnis interpretiert wird,

codiert werden. Abbildung 5.2 stellt den Reaktor bei der Ausführung einer Chemie dar. Hervorgehoben sind die Komponenten, die entweder evolviert oder in Form der Eingabe durch externe Werte, z. B. aus der Trainingsmenge, gesetzt werden.

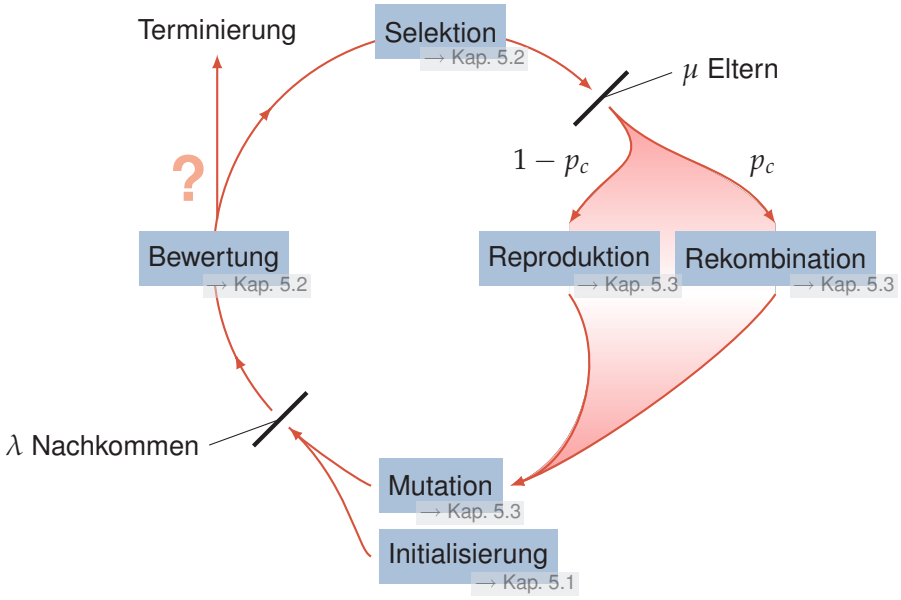


Abbildung 5.1.: Stationen, die die Population beim Erlernen algorithmischer Chemien durchläuft. Die initial erzeugten λ Individuen werden bewertet und die μ besten als Eltern selektiert. Durch Reproduktion, Rekombination und Mutation werden λ Nachkommen erzeugt, die sich wiederum der Bewertung und Selektion stellen. Dieses wiederholt sich solange, bis ein Terminierungskriterium erfüllt wurde.

Molekül

Während der Reaktor bei künstlichen Chemien die Moleküle enthält und eine zufällig gezogene Teilmenge die Wahl der Reaktion beschränkt, enthält er im Fall der algorithmischen Chemie eine Reaktionsmultimenge. Die für die Reaktion notwendigen Moleküle stehen permanent in der Umgebung zur Verfügung. Das System unterscheidet drei Molekültypen. Der erste Typ dient der *Interaktion* der Reaktionen, indem diese Moleküle nicht nur als Edukt sondern auch als Produkt einer Reaktion verwendet serden können. Zu Beginn der Ausführung wird ihr Zustand mit dem Wert 0 beschrieben. Der Zustand eines dieser Moleküle wird am Ende der Ausführung als Ausgabe interpretiert. Die Wahl wird ebenfalls im Individuum evolviert. Der zweite Typ dient der *Eingabe*. Die Zustände dieser Moleküle werden durch Eingabedaten des betrachteten Fallbeispiels bestimmt und dürfen durch die Reaktionen nicht verändert werden. Beim dritten Typ handelt es sich um *zustandskonstante* Moleküle. Die Konstantheit bezieht sich auf den Zeitraum der Ausführung. Der Zustand dieser Moleküle unterliegt im Individuum der Evolution. Als Bestandteil des Individuums, werden die Werte vererbt und unter Umständen mutiert. Während der Ausführung der Chemie bleiben sie unverändert. Sie entsprechen damit den evolvierten Konstanten in einem linearen GP System[1].

[1] Baum-GP arbeitet zumeist mit speziellen, als „ephemeral random constant“ bezeichneten, Terminalen. Diese werden bei der ersten Verwendung durch eine gleichverteilte Zufallszahl ersetzt und

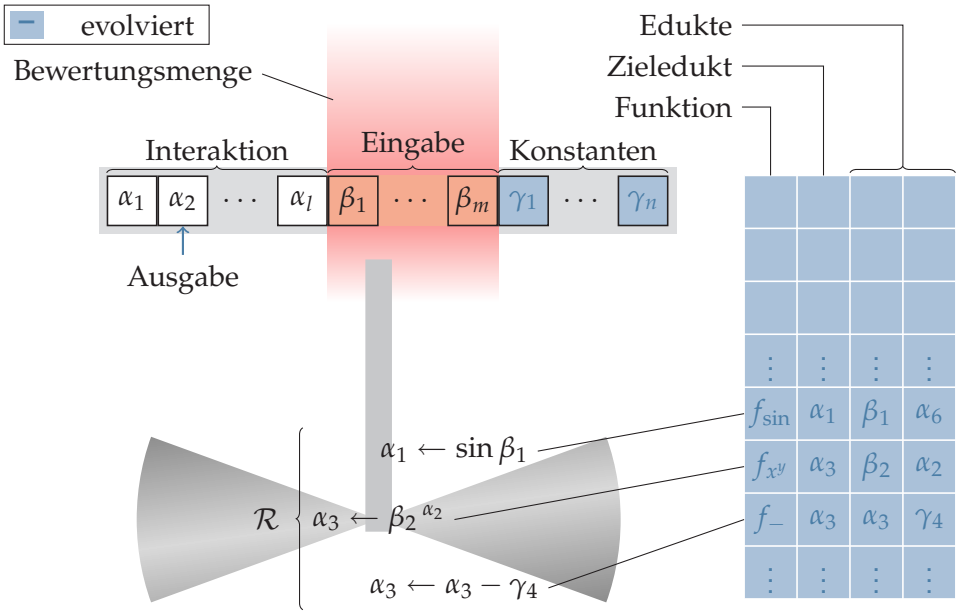


Abbildung 5.2.: Die im Individuum evolvierten Komponenten gelangen zur Ausführung in den Reaktor. Zu diesen Komponenten zählen die Regeln sowie die Zustände der konstanten Moleküle. Zusätzlich wird ein Molekül bestimmt, dessen Zustand, nach Durchführung aller Reaktionen, als Ergebnis interpretiert wird. Ein Teil der Molekülezustände werden von außen durch die jeweilige Eingabe des aktuell betrachteten Fallbeispiels gesetzt.

Bei den letzten beiden Molekültypen handelt es sich um Terminale im in der Chemie codierten Datenfluss, da ihr Zustand nicht durch eine Reaktion verändert werden kann.

Reaktionen

Eine *Reaktion* besteht aus den Edukten, d. h. einer Menge von Molekülen beliebigen Zustands, der Information darüber, welches der Edukte aufgrund der Reaktion eine Zustandsänderung erfährt, und einer Funktion, welche diese Zustandsänderung beschreibt.

Tabelle 5.1 listet die im System für die Bildung von Reaktionen zur Verfügung stehenden arithmetischen, trigonometrischen und booleschen Funktionen auf. Aus ihnen wird problemspezifisch eine Teilmenge gewählt, die in den Reaktionen zur Anwendung kommen darf. Dabei findet bei jeder Reaktion genau eine Funktion Anwendung und bestimmt mit ihrem Funktionswert den Zustand eines Moleküls (Gl. 4.1). Denkbar sind in einer algorithmischen Chemie aber auch Reaktionen, bei denen mehrere Moleküle durch verschiedene Funktionen eine Änderung erfahren, wie z. B. in der Reaktion

$$\alpha + \beta \rightarrow (\alpha = \min(\alpha, \beta)) + (\beta = \max(\alpha, \beta)),$$

können anschließend nicht mehr verändert werden.

Tabelle 5.1.: Menge der Funktionen, die bei der Bildung von Reaktionen für eine algorithmische Chemie verwendet werden können. Die Zustände der Moleküle $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ werden vor der Verwendung einer logischen Funktion zunächst in einen booleschen Wert verwandelt (siehe Text).

	Bezeichner	Arität n	Funktion ($f(\alpha_1, \ldots, \alpha_n) =$)
arithmetische Funktionen			
1.	ADD	2	$\alpha_1 + \alpha_2$
2.	SUB	2	$\alpha_1 - \alpha_2$
3.	MULT	2	$\alpha_1 \cdot \alpha_2$
4.	DIV	2	α_1 / α_2 wenn $\alpha_2 \neq 0$, sonst 0
5.	POWER	2	${\alpha_1}^{\alpha_2}$
trigonometrische Funktionen			
6.	SIN	1	$\sin(\alpha_1)$
7.	COS	1	$\cos(\alpha_1)$
8.	TAN	1	$\tan(\alpha_1)$
boolesche Funktionen			
9.	AND	2	$\alpha_1 \wedge \alpha_2$
10.	OR	2	$\alpha_1 \vee \alpha_2$
11.	NAND	2	$\overline{\alpha_1 \wedge \alpha_2}$
12.	NOR	2	$\overline{\alpha_1 \vee \alpha_2}$
13.	NOT	1	$\overline{\alpha_1}$
14.	XOR	2	$(\alpha_1 \vee \alpha_2) \wedge (\overline{\alpha_1 \wedge \alpha_2})$

die als Grundlage einer Sortierchemie dienen könnte.

Der Zustand der Moleküle wird unabhängig von der gewählten Funktionsmenge durch eine reellwertige Zahl repräsentiert, daher bedarf es vor der Anwendung boolescher Funktionen der Abbildung auf einen booleschen Wert sowie der Darstellung des Funktionswerts als reellwertige Zahl. Ist der Zustand eines Moleküls α größer 0, so wird dieser als „wahr“ interpretiert. Ist der Zustand kleiner oder gleich 0, so wird er als „falsch“ interpretiert. Ist der Funktionswert „wahr“, so wird dieses als 1 dargestellt, ist er „falsch“ so wird dieses als 0 dargestellt. Zusätzlich wird die Division durch 0 abgefangen. Funktions- und Terminalmenge sind damit abgeschlossen.

Jede Reaktion wird im Genom durch ein Tupel fester Größe codiert. Der erste Tupeleintrag wählt die Funktion. Ihr Funktionswert bestimmt den neuen Zustand des Moleküls, welches durch einen weiteren Tupeleintrag gewählt wird. Die restlichen Einträge des Tupels bestimmen die Argumente der Funktion. Die Anzahl der codierten Argumente entspricht der maximalen Arität in der Menge verwendeter Funktionen. Nicht benötigte Argumente werden ignoriert.

Initialisierung

Die im Genom codierten Informationen müssen zu Beginn der Evolution in jedem Individuum mit einem Wert versehen werden. Die Zustände der konstanten Moleküle $\gamma_1, \ldots, \gamma_m$ erhalten initial einen normalverteilten Zufallswert mit Mittelwert 0 und einer Standardabweichung von 1. Die Wahl des Moleküls, dessen Zustand nach Durchführung der letzten Reaktion als Ergebnis interpretiert wird, geschieht gleichverteilt aus allen für die Interaktion bestimmten Molekülen, da nur diese in ihrem Zustand veränderbar sind.

Die Anzahl der zu Beginn vorhandenen Reaktionen ist durch die *Initialkardinalität* vorgegeben. Jede Regel der Multimenge wird unabhängig voneinander zufällig erzeugt. Die Funktion wird gleichverteilt aus der Menge der erlaubten Funktionen bestimmt. Das Produkt, welches bei der Ausführung den Funktionswert annimmt, wird zufällig aus allen für die Interaktion der Reaktionen bestimmten Molekülen gewählt. Die restlichen Edukte, die als Argumente der Funktion fungieren, werden unabhängig voneinander aus der Gesamtmenge der Moleküle bestimmt.

Ausführung

Vor Beginn einer jeden Ausführung werden zuerst die Zustände der Moleküle im Reaktor gesetzt. Für die Interaktion der Reaktionen bestimmte Moleküle erhalten den Wert 0. Die während der Ausführung zustandskonstanten Moleküle nehmen die im Individuum evolvierten Werte an. Die verbleibenden Moleküle dienen der Kommunikation mit der Chemie und werden für jedes Fallbeispiel mit dessen Eingabewerten belegt.

Die Ausführung der algorithmischen Chemie geschieht durch zufällige Wahl der Reaktion aus der evolvierten Reaktionsmultimenge (Alg. 4.1). Wie eingangs bereits erwähnt, werden die Unterschiede zwischen der Evolution algorithmischer Chemien und dem linearen GP für einen späteren Vergleich auf ein Minimum reduziert. Die Umsetzung erfolgt im Hinblick hierauf. Verhaltensunterschiede werden zunächst auf den Programmzeiger begrenzt.

In Abb. 5.3a ist die Ausführung eines linearen Programms dargestellt. Zu jedem Zeitpunkt wird genau eine Instruktion ausgeführt. Bestimmt wird diese Instruktion durch den Programmzeiger. Wird die Wahl der Position X, an der sich die auszuführende Instruktion befindet, als Wahrscheinlichkeitsvariable betrachtet, so stammt diese bei einem linearen Programm aus einer diskreten Verteilung. Diese Verteilung wird parametrisiert über die Position z des Programmzeigers

$$P_{\mathrm{LGP}}(X = x, z) = \begin{cases} 1 & \text{wenn } x = z, \\ 0 & \text{sonst.} \end{cases} \tag{5.1}$$

Der Parameter sorgt dafür, dass jene Instruktion ausgeführt wird, auf die der Programmzeiger verweist. Nach der Ausführung verweist der Programmzeiger auf die Folgeinstruktion. Bei der algorithmischen Chemie erfolgt die Ausführung der

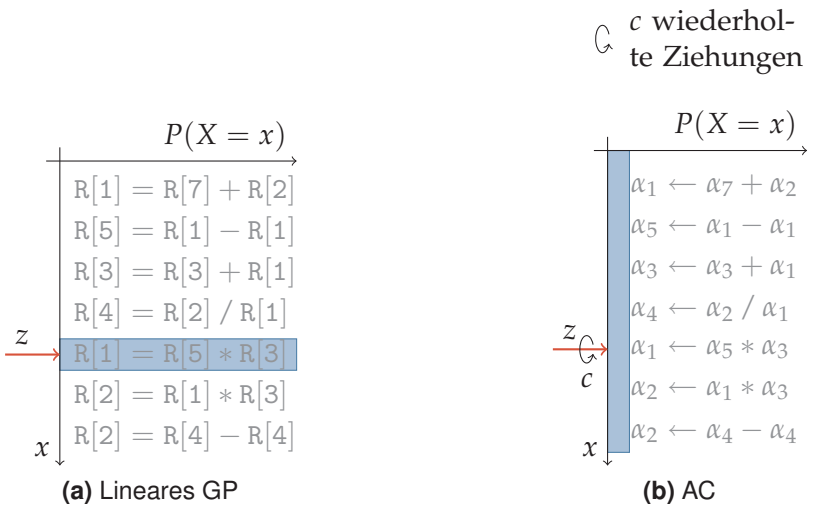


Abbildung 5.3.: Die Unterschiede in der Ausführung (a) eines linearen Programms und (b) einer algorithmischen Chemie lassen sich zurückführen auf Unterschiede in der Zuordnung der Programmzeigerposition zur ausgeführten Reaktion/Instruktion. Die blaue Fläche stellt diese Zuordnung als Wahrscheinlichkeitsfunktion dar. Während bei linearen Programmen eine diskrete zeitabhängige Funktion zu jedem Zeitpunkt exakt eine Instruktion bestimmt, führt die bei der algorithmischen Chemie gewählte Funktion zur zufälligen Wahl der Reaktionen.

Reaktionen in zufälliger Reihenfolge (Abb. 5.3b), bei einer Reaktionsmultimenge der Kardinalität $|\mathcal{R}| = n$ wird die Verteilung durch die diskrete Gleichverteilung

$$P_{\mathrm{AC}}(X = x, z) = \frac{1}{n}$$

ersetzt. Die Parametrisierung der Verteilung mit dem Programmzeiger z deutet lediglich an, dass die Reaktionsmultimenge im Rechner als lineare Datenstruktur verwaltet wird und somit jede Reaktion über eine Position $1 \leq x \leq n$ angesprochen werden kann. Die Wahl der auszuführenden Reaktion ist bei der algorithmischen Chemie unabhängig vom Programmzeiger, auch wenn sich dieser über die Elemente der Reaktionsmultimenge bewegt.

Für die Beschreibung des Verhaltens algorithmischer Chemien über den hier wirkungslosen Programmzeiger gibt es drei Gründe. Zunächst ist diese Sichtweise nützlich, um in Kap. 5.3 auf ihr aufbauend die Rekombination algorithmischer Chemien direkt aus der Rekombination zweier linearer Programme abzuleiten. Desweiteren basiert auch die eigentliche Implementierung auf dem Austausch der Methode für die Zuordnung eines virtuellen, zeitabhängigen Programmzeigers zur auszuführenden Reaktion bzw. Instruktion. Im Ausblick (Kap. 13) ermöglicht es diese Sichtweise, die Verwendung anderer Wahrscheinlichkeitsverteilungen zu motivieren und so ein Verhalten zwischen dem der algorithmischen Chemien und dem Verhalten linearer Programme zu erhalten.

Die Assemblierung von Funktionalität bei der Ausführung einer algorithmischen Chemie erfordert einen Mehraufwand bzgl. der Anzahl ausgeführter Reaktionen. Instruktionen in einem linearen Programm erwarten implizit, dass die von ihnen als Argumente verwendeten Register bereits durch vorangegangene Instruktionen des Programms angemessen belegt worden sind. Die Evolution versucht dieses durch eine entsprechende Sequenzialität der Instruktionen sicherzustellen. Auch eine Reaktion in der algorithmischen Chemie benötigt für die erfolgreiche Ausführung die Herstellung geeigneter Zustände in ihren Edukten durch weitere Reaktionen der Chemie. Es besteht bei ACs nicht die Möglichkeit, durch Evolution weiterer Information, eine Reihenfolge sicherzustellen. Im Gegensatz zu den meisten anderen Repräsentationen enthält die Position der Reaktion im Genom keine Semantik. Idealisiert betrachtet existiert nicht einmal eine Position. Einzig die Menge der zuvor stattgefundenen Reaktionen sowie ihre Eindeutigkeit (siehe Kap. 6.2), können die Wahrscheinlichkeit erhöhen, dass die Edukte durch entsprechende Reaktionen in den gewünschten Zustand versetzt wurden.

Eine Reduzierung der evolvierten Informationsmenge um den Zeitpunkt der Ausführung wird hier durch einen höheren Aufwand bei der Ausführung erkauft, der durch die Selbstorganisation nötig wird. Funktionalität wird von einer Reaktion ausschließlich über die Verwendung entsprechender Edukte angesprochen und durch Änderung des Zustands eines weiteren Moleküls zur Verfügung gestellt. Damit kommt den Molekülen eine ähnliche Aufgabe zu, wie den Spezifitäten und Profilen im „Enzyme Genetic Programming“ [69, 70]. Die Moleküle beschreiben den Kontext, in dem die Funktion der Reaktion verwendet werden soll. Die Aufgabe übernehmen bei den Instruktionen im Genom eines linearen Individuums die Register zusammen mit der Position der Instruktion im Genom.

Dem Umstand, dass die Wahrscheinlichkeit für die erfolgreiche Assemblierung einer algorithmischen Chemie über die Anzahl ausgeführter Reaktionen gesteigert werden kann, wird durch Bestimmung eines *Vielfachen* $c \in \mathbb{N}$ Rechnung getragen. Das Produkt aus der Anzahl Reaktionen der Chemie und dem Vielfachen c ergibt die Anzahl der tatsächlich ausgeführten Reaktionen. Technisch bestimmt das Vielfache, wie viele Reaktionen an einer Programmzeigerposition entnommen und ausgeführt werden. Bei linearen Programmen beträgt der Wert 1. Auch andere Systeme (z. B. das in Kap. 1.1.2 beschriebene PADO [113]) führen gezielt ein Vielfaches der in ihnen evolvierten Instruktionen aus. Die Anzahl ausgeführter Reaktionen hätte auch absolut bestimmt werden können, dann hätte aber die Gefahr einer systematischen Tendenz zu kleinen Individuen bestanden. Denn bei konstanter Anzahl ausgeführter Reaktionen, wären die Reaktionen kleiner Individuen häufiger ausgeführt und somit die Wahrscheinlichkeit zur vollständigen Assemblierung erhöht worden.

Diese wiederholte Ausführung von Instruktionen erscheint zunächst als Verschwendung von Rechenzeit. Während es immer möglich ist, eine Verhaltensvariante in ein lineares Programm zu verwandeln, um dieser Kritik zu entgehen, sind auch hardwareseitige Beschleunigungen vorstellbar, zum Beispiel durch eine große Anzahl an Prozessoren, welche auf derselben Multimenge arbeitet. Im Ex-

tremfall wird jede Reaktion wiederholt von einem eigenen Prozessor ausgeführt und so das Programm in kürzest möglicher Zeit, bestimmt durch die Tiefe des Datenflusses, abgearbeitet. Angepasste Varianten von Datenflussarchitekturen, z. B. die Wavescalar-Architektur [112], haben das Potenzial, diese Geschwindigkeitssteigerung mit geringerem Aufwand zu erreichen.

Zum besseren Verständnis sei noch einmal betont, dass ein Programmzeiger zur Beschreibung des zufälligen Verhaltens algorithmischer Chemien nicht nötig ist. Die gewählte Beschreibung über den Programmzeiger dient hier zunächst dazu, die Unterschiede der Systeme an einem Ort zu konzentrieren. Bei der Implementierung der Ausführungsprozedur unterscheiden sich algorithmische Chemien und lineare Programme damit in zwei Punkten

1. Die Funktion, welche die zu einem Programmzeiger gehörige Instruktion bzw. Reaktion ermittelt, gibt für lineare Programme die vom Zeiger referenzierte Instruktion zurück und wählt für algorithmische Chemien zufällig eine Reaktion.

2. Während der Programmzeiger bei linearen Programmen nach jeder Ausführung auf die im Speicher folgende Instruktion des Individuums gesetzt wird, bewegt sich dieser bei ACs erst nach einer durch das Vielfache c vorgegebenen Anzahl Wiederholungen über die nur durch die Speicherung gegebene Ordnung der Reaktionsmultimenge.

5.2. Bewertung und Selektion

Die Bewertung der Individuen geschieht auf der Grundlage einer Menge von problemspezifischen Fallbeispielen[2]. Jedes Fallbeispiel $(\vec{i}, o)$ besteht aus einem Vektor von Eingaben $\vec{i}$ und der dafür vom System zu erlernenden Ausgabe o. Wie in Abb. 5.2 dargestellt, wird für jedes Fallbeispiel k ein Teil der vorhanden Moleküle in einen von den Eingabewerten $\vec{i_k}$ bestimmten Zustand versetzt und anschließend die Chemie ausgeführt. Der Zustand eines per Evolution bestimmten Moleküls wird nach der Ausführung als Ausgabe o'_k interpretiert. Aus der Folge gewünschter Ausgaben $\vec{o}$ und den realisierten Ausgaben $\vec{o}\,'$ berechnet die Fitnessfunktion $f(\vec{o}, \vec{o}\,')$ die Güte des Individuums. Die Fitnessfunktion ist abhängig von der Anwendung und wird zusammen mit den Anwendungen in Kap. 8 beschrieben.

Die nichtdeterministische Ausführung einer algorithmischen Chemie unter dem Gesichtspunkt der Genotyp-Phänotyp-Abbildung betrachtet verdeutlicht, dass bei der Auswertung einer Menge von Fallbeispielen nicht nur ein Phänotyp, sondern eine ganze Menge möglicher, nicht zwangläufig gleichwahrscheinlicher Verhaltensvarianten bewertet wird (siehe Abb. 5.4). In den beiden Extremfällen wird entweder jedes Fallbeispiel auf eine anderen Phänotypvariante abgebildet oder bei jeder Abbildung derselbe Phänotyp erzeugt. Jedes Individuum speichert eine Verteilung

[2]Der in der englischsprachigen Literatur verwendete Begriff ist „fitness case“.

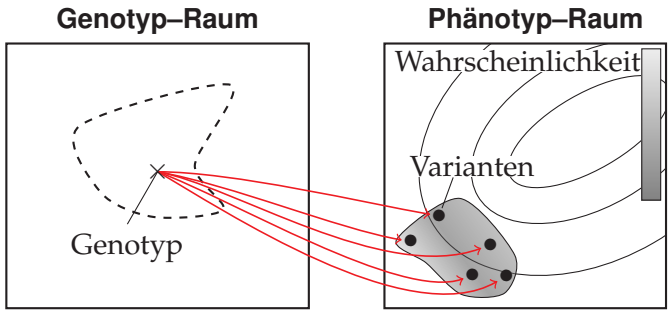


Abbildung 5.4.: Ähnlich dem linearen GP, können bei der algorithmischen Chemie unterschiedliche Genotypen dieselbe Funktionalität aufweisen (gestrichelte Linie), so kann z. B. die im Genom codierte Liste der Reaktionen beliebig permutiert werden. Im Unterschied zum linearen GP vollzieht sich die Abbildung auf den Phänotypen aber nichtdeterministisch, wodurch bei jeder Ausführung unter Umständen ein anderer Phänotyp betrachtet wird. Die Wahrscheinlichkeit, mit der die einzelnen Phänotypen realisiert werden, hängt von der Zusammensetzung der Chemie ab. Die erzeugten Phänotypen können für sich genommen eine unterschiedliche Fitness, hier durch die Höhenlinien angedeutet, aufweisen. Die Fitnessbewertung des Individuums geschieht jedoch auf der Grundlage aller erzeugten Phänotypen.

möglicher Verhaltensvarianten, vergleichbar mit den Verteilungen in den EDAs aus Kap. 1.3. Bei den EDAs dient eine gemeinsame Verteilung zur Repräsentation der Elternpopulation. Nachkommen werden aus dieser gezogen und anschließend die Verteilung auf der Grundlage selektierter Individuen angepasst. Bei der genetischen Programmierung einer algorithmischen Chemie hingegen kann jedes Individuum als Verteilung aufgefasst werden. Für jedes Fallbeispiel zeigt das Individuum unter Umständen eine andere Verhaltensvariante. Die Verhaltensvarianten werden dann gesamtheitlich bewertet – die Fitness des Individuums. Eine „Anpassung“ der Verteilungen geschieht in den Nachkommen durch Rekombination und Mutation selektierter Individuen.

Die Expression eines Genotypen auf eine Menge unterschiedlicher Phänotypen stellt eine Form von *epigenetischer Variation* dar. Epigentische Variation und Selektion bezeichnen Kirschner und Gerhart [61] als „erforschenden Mechanismus“. Hierbei wird häufig eine große Anzahl Konfigurationen oder Zustände erzeugt, von denen andere Komponenten eine gewünschte Funktionalität selektieren. Das adaptive Immunsystem ist ein Beispiel für die Existenz solcher Mechanismen in Lebewesen. Hier wird ständig eine große Vielfalt an T-Zellen mit unterschiedlichen Antigenrezeptoren erzeugt. Wird ein Rezeptor durch eine Zelle, die das entsprechende Antigen trägt, aktiviert, so wird die T-Zelle zur Vermehrung angeregt. Zudem sendet sie Botenstoffe aus, die etwa die Produktion von Antikörpern anregen. Als weiteres Beispiel nennen Kirschner und Gerhart die Verbindung zwischen Neuronen. Nach einer anfänglich übermäßigen Produktion von Verbindungen findet hier eine Auswahl und Beschneidung in Abhängigkeit neuraler Aktivität statt. Epigenetische Variation und Selektion dienen der Evolutionsfähigkeit, indem sie

die Anforderungen an andere Komponenten oder genauen Umstände verringern. Zudem reduzieren sie die Anzahl an Mutationen, die ein System unterlaufen muss, um neue Funktionalität hervorzubringen. Im Unterschied zur Selektion epigenetischer Variation werden die einzelnen Verhaltensvarianten einer algorithmischen Chemie nicht direkt selektiert, sondern führen über ihre durchschnittliche Eignung ggf. zu einem Selektionsvorteil.

In der genetischen Programmierung werden typischerweise drei Mengen von Fallbeispielen unterschieden:

- die Trainingsmenge,
- die Testmenge und
- die Validierungsmenge.

Die Güte auf der *Trainingsmenge* ist dabei die Einzige, die die Selektion der Individuen und damit die Evolution beeinflusst. Dem entsprechend wird bei der Bewertung des evolutionären Fortschritts in Abhängigkeit der Anzahl der betrachteten Fallbeispiele oder stattgefundenen Reaktion nur der Aufwand berücksichtigt, der bei Auswertungen auf Fallbeispielen der Trainingsmenge entstanden ist. Die Trainingsmenge wird zumeist aus einer größeren Menge von Fallbeispielen in jeder Generation neu zusammengestellt. Die Trainingsmengengröße ist wählbar. Bei der Bestimmung der Größe muss der Aufwand gegen die Unsicherheit abgewägt werden. Letzteres macht sich bei zu klein gewählten Trainingsmengen dadurch bemerkbar das der Fokus, den eine kleine Trainingsmenge auf die Problemstellung legt, stark schwankt[3]. Zudem führt bei algorithmischen Chemien eine größere Trainingsmenge zu einer exakteren Bewertung des Phänotypraums, der zu einem Individuum gehört (vgl. Abb. 5.4).

Die Trainingsmenge wird aus der Menge hierfür zu Verfügung gestellter Fallbeispiele mittels stochastischer Teilmengenselektion (SSS, „stochastic subset selection“) gewählt. Bei der algorithmischen Chemie kann die gewählte Trainingsmengengröße die Anzahl der Fallbeispiele überschreiten. Eine Mehrfachbetrachtung einzelner Fallbeispiele ist hier, im Gegensatz zum linearen GP, dienlich, da die erneute Betrachtung eines Fallbeispiels u. U. auf einer anderen phänotypischen Ausprägung des Individuums erfolgt. Bei Mehrfachverwendung der Fallbeispiele wird sichergestellt, dass sich die Häufigkeit zweier Fallbeispiele in der Trainingsmenge höchstens um 1 unterscheidet.

Neben der Trainingsmenge existieren zwei weitere Mengen an Fallbeispielen im System. Mit der *Testmenge* wird geprüft, welche Güte Individuen auf einer Menge im Laufe der Evolution nicht erlernter Fallbeispiele haben. Die durchschnittliche Güte wird in regelmäßigen Abständen für die Individuen der Elternpopulation protokolliert. Die Generalisierung des besten Individuums auf der Testmenge wird

[3] Indem zum Beispiel bei einem Klassifikationsproblem Klassen wechselweise die Trainingsmenge dominieren.

noch einmal mithilfe der *Validierungsmenge* überprüft. Auch die hier ermittelte Güte wird protokolliert.

Die Eltern der Folgegeneration werden aus den Nachkommen der aktuellen Generation mittels *Kommaselektion* gewählt. Diese Form der Selektion hat ihren Ursprung im Bereich der Evolutionsstrategien [21] und ist außerhalb der Evolutionsstrategien auch als „Truncation"-Selektion bekannt. Bei diesem Verfahren werden die λ Nachkommen anhand ihrer auf der Trainingsmenge ermittelten Güte sortiert und die besten $\mu < \lambda$ Individuen als Elternpopulation der nächsten Generation gewählt. Der *Selektionsdruck* ν bestimmt sich aus dem Verhältnis zwischen der Anzahl Nachkommen und der Anzahl aus ihnen selektierter Eltern zu $\frac{\lambda}{\mu}$. Die Anzahl der Eltern und der Selektionsdruck sind Parameter des Systems. Die Anzahl der Nachkommen ergibt sich aus dem gerundeten Produkt ihrer Werte, $\lambda = \lfloor \mu \cdot \nu \rfloor$.

5.3. Rekombination und Mutation

Mit der Rekombinationswahrscheinlichkeit p_c werden die Nachkommen durch Rekombination zweier Eltern und mit der Wahrscheinlichkeit $(1 - p_c)$ durch Reproduktion eines Elter erzeugt. Die Wahl der Eltern aus der aktuellen Population geschieht in beiden Fällen zufällig. Die erzeugten Nachkommen werden erst nach ihrer Selektion zu potenziellen Eltern.

Die *Reproduktion* geschieht durch Erzeugung einer identischen Kopie des Elters und unterscheidet sich bei der algorithmischen Chemie nicht von der Evolution linearer Programme. Bei der *Rekombination* zweier Eltern werden zunächst unterschiedliche Ansätze verfolgt, die aber bei der Realisierung auf das unterschiedliche Verhalten des Programmzeigers zurückgeführt werden können. Dieser Unterschied wurde bereits für die Umsetzung der Ausführung eingeführt, sodass es hier keiner weiteren Differenzierung der Systeme bedarf.

Beim linearen GP-Referenzsystem findet eine *1-Punkt-Rekombination* statt, d. h., im Genom beider Eltern wird jeweils ein Rekombinationspunkt bestimmt und die vor diesem Punkt liegende Instruktionssequenz des einen Elters mit der hinteren Instruktionssequenz des anderen Elters durch Aneinanderhängen kombiniert. Der Grundgedanke bei der Rekombination zweier algorithmischer Chemien ist die zufällige Kombination ihrer Reaktionsmultimengen. Hierfür wird eine zufällige Reaktionsteilmultimenge des einen Elters mit jener eines zweiten Elters vereint.

Bei der Umsetzung wird auf die für die Ausführung der Individuen eingeführte Zuordnung zwischen Programmzeiger und Instruktion bzw. Reaktion zurückgegriffen. Zunächst wird in beiden Eltern jeweils ein Rekombinationspunkt c_1 und c_2 gleichverteilt in der Liste ihrer Instruktionen bzw. Reaktionen gewählt. Ähnlich der Ausführung wandert nun der Programmzeiger vom ersten Element der Liste an über den ersten Elter, bis der Zeiger die Position c_1 überschritten hat. Die hierbei zurückgegebenen Instruktionen/Reaktionen bilden den ersten Teil des Genoms im Nachkommen. Angefangen bei der Zeigerposition c_2 bewegt sich anschließend der

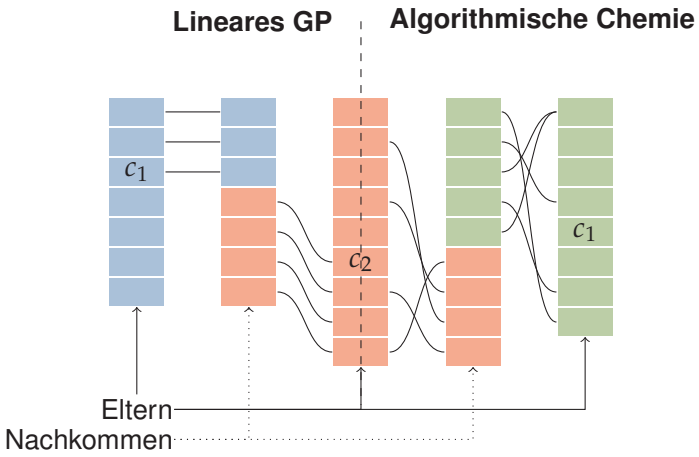


Abbildung 5.5.: Die Rekombination nutzt den für die Ausführung linearer Programme und algorithmischer Chemien eingeführten Unterschied in der Zuordnung einer Instruktion zur Position des Programmzeigers (vgl. hierzu Abb. 5.3). Den ersten Abschnitt des Nachkommen formen jene Einträge, die die Funktion auf dem Weg des Programmzeigers vom ersten Eintrag des ersten Elters bis zum Rekombinationspunkt c_1 zurückgibt. Der hintere Teil entspricht den Einträgen, die die Funktion auf dem Weg des Programmzeigers von c_2 zur letzten Instruktion zurückgibt. Während der zurückgegebene Eintrag beim linearen GP der Instruktion unter dem Programmzeiger entspricht, handelt es sich bei der algorithmischen Chemie um eine zufällige Reaktion.

Programmzeiger über den zweiten Elter, bis dessen Ende erreicht ist. Die hierbei zurückgegebenen Reaktionen bilden den zweiten Teil des Nachkommengenoms. Die wählbare Anzahl wiederholter Ziehung an einer Zeigerposition wird dabei auf 1 gesetzt.

Für die Individuen ist eine maximale Länge des Genoms definiert. Um bei der Rekombination die Einhaltung dieser Grenze sicherzustellen, werden ggf. die Rekombinationspunkte c_1 und c_2 so verschoben, dass die Nachkommen weniger Instruktionen von den Eltern erben. Auch bei den algorithmischen Chemien liegen die von den beiden Eltern stammenden Teile des Genoms in einem zusammenhängenden Speicherbereich, es sei aber daran erinnert, dass diese Reihenfolge bei der Ausführung keine Beachtung findet. Abbildung 5.5 stellt die Rekombination in beiden Systeme noch einmal einander gegenüber.

Neben der Reaktionsmultimenge werden im Individuum weitere Informationen evolviert, die es in den Nachkommen zu kombinieren gilt. Die in den Individuen evolvierten Werte zustandskonstanter Moleküle werden für jedes Molekül zufällig aus einem der beiden Eltern entnommen, ebenso die Wahl des Moleküls, dessen Zustand nach der Ausführung als Ergebnis interpretiert wird.

Unabhängig von der Produktion des Nachkommen per Rekombination oder Reproduktion unterläuft jedes Individuum anschließend die *Mutation*. Hierfür wird jede evolvierte Information mit der Mutationswahrscheinlichkeit p_m geändert. In jeder codierten Reaktion können die verwendeten Moleküle durch andere Moleküle ersetzt werden, dabei ist die Wahl des Produktmoleküls auf die für die

Interaktion bestimmten Molekülen beschränkt. Auch die verwendete Funktion kann aus der Funktionsmenge neu bestimmt werden. Mit derselben Wahrscheinlichkeit ändern sich die codierten Zustände für die zustandskonstanten Moleküle. Der mutierte Zustand ist dabei abhängig von dem aktuellen Wert γ_{alt} und wird aus der Normalverteilung $\mathrm{N}(\gamma_{\text{alt}}, \gamma_{\text{alt}}/10)$ bestimmt. Die Mutation der Wahl des Ergebnismoleküls geschieht durch zufällige Wahl eines der für die Interaktion der Reaktionen bestimmten Moleküls.

Die Tabelle 5.2 fasst abschließend noch einmal die Parameter des Systems aus den Abschnitten 4.2 und 5 zusammen.

5.4. Totalsynthese

Die Reaktionsmultimengen der evolvierten algorithmischen Chemien besitzen keine Informationen über die Ausführungsreihenfolge, die Instruktionen werden in zufälliger Reihenfolge im Reaktor ausgeführt. Die Assemblierung des Datenflusses geschieht im Verlauf der Zeit, da dieser nicht explizit, sondern implizit über die gemeinsame Verwendung von Molekülen in den Reaktionen codiert ist. Zeit wird hier an der Anzahl ausgeführter Reaktionen gemessen.

Während der Zeitbedarf zu einem gewissen Grad durch Parallelität von Reaktionen reduziert werden kann, ist es offensichtlich, dass die Ausführung algorithmischer Chemien mehr Rechenleistung benötigt als ein lineares Programm gleicher Datenflussgröße. Dabei wird sehr viel Rechenleistung auf die Ausführung von Reaktionen verwendet, deren Berechnungen keinen Eingang in den Zustand des Moleküls finden, welches als Ergebnis interpretiert wird.

Es lassen sich zwei Arten von überflüssigen Reaktionsausführungen unterscheiden. Die erste Art gleicht den Introns, sprich dem ineffektiven Code, in anderen Repräsentationen der genetischen Programmierung. Diese Reaktionen nehmen unter keinen Umständen am Datenfluss teil, weil sie z. B. den Zustand eines Moleküls ändern, das nicht in den Edukten anderer Reaktionen verwendet wird. Bei der zweiten Art handelt es sich um Reaktionen, bei denen der Zeitpunkt der Ausführung darüber entscheidet, ob sie „gewinnbringend“ Eingang in das Endergebnis finden. So kann auf die Ausführung einer Reaktion so lange verzichtet werden, wie die verwendeten Edukte noch nicht den letztendlichen Zustand erreicht haben. Für die erfolgreiche Ausführung ist wichtig, dass alle relevanten Reaktionen mindestens einmal zu einem Zeitpunkt stattfinden, an dem ihre Edukte ihren Zustand angenommen haben. Auch wenn mehrere Reaktionen dasselbe Produkt verwenden, welches Einfluss auf das Endergebnis nimmt, so kann letztendlich nur eine Reaktion mit ihrer Funktion den Zustand des Produkts bestimmen. Es ist überflüssig, die Zustände in den Edukten der anderen Reaktionen zu bestimmen.

Während eine hohe Anzahl ausgeführter Reaktionen die Wahrscheinlichkeit für die vollständige Assemblierung der Lösung erhöht, ist eine geringe Anzahl wünschenswert, um die erforderliche Rechenleistung zu reduzieren. Eine veränderte Ausführung zugunsten eine Aufwandsreduzierung ist hier für den Prozess der

Tabelle 5.2.: Parameter des AC- und LGP-Systems

Bezeichner	Beschreibung
Genetische Programmierung	
Populationsgröße μ	Anzahl der Individuen in der Elternpopulation
Selektionsdruck ν	Vielfaches der aus den Eltern zu erzeugenden Nachkommen
Rekombinationsrate p_c	Wahrscheinlichkeit, dass die Nachkommen durch Rekombination entstehen, mit der Wahrscheinlichkeit $(1 - p_c)$ entstehen die Nachkommen durch Reproduktion
Mutationswahrscheinlichkeit p_m	Wahrscheinlichkeit mit der jede Information in einem Nachkommen mutiert wird
Repräsentation	
Initialkardinalität	Reaktionen(AC) / Instruktionen(LGP) pro Individuum bei der Initialisierung
Maximalkardinalität	maximale Anzahl Reaktionen(AC) / Instruktionen(LGP) pro Individuum
Interaktion	Anzahl der zustandsveränderbaren Moleküle(AC) bzw. schreib-lese Register(LGP)
Konstanten	Anzahl zustandkonstanter Moleküle(AC) bzw. nur-lese Register(LGP), deren Wert im Individuum evolviert wird
Vielfache c (nur AC)	bestimmt als Vielfaches der im Individuum codierten Reaktionen die Anzahl der zufällig ausgeführten Reaktionen
Aufgabenstellung	
Eingabegröße	Anzahl der Parameter pro Probleminstanz
Funktionsmenge	in den Reaktionen(AC) bzw. Instruktionen(LGP) verwendete Funktionen
Trainingsmengengröße	Anzahl der Fallbeispiele die zur Bewertung eines Individuums herangezogen werden
Fitnessfunktion	Funktion zur Bewertung der Güte eines Individuums
Terminierungskriterium	Evolutionsdauer in Anzahl Individuenbewertungen

Input : Menge der Fallbeispiele F, Individuum $I = (\text{Reaktionsmultimenge } \mathcal{R}$, Konstanten, Wahl des Ergebnismoleküls $\alpha_r)$
Output : Güte von I bzgl. F

```
1 v⃗ ←<>,  v⃗' ←<>;                                  /* leere Listen */
2 S = (S_α1, ...) ←gruppiere(R);          /* anhand „Zielmolekül" */
3 foreach Beispiel ∈ F do
    // Beispiel = (Eingabe, Ausgabe)
4   T ←(Eingabe, Konstanten);                        /* Terminale */
5   Ergebnis ← recEval(α_r, S, T, ∅);
6   v⃗ ←append (v⃗, Ausgabe);
7   v⃗' ←append (v⃗', Ergebnis);
8 end
9 return f(v⃗, v⃗');                                 /* Fitnessfunktion */
```

Algorithmus 5.1 : Rahmen für die Ausführung einer algorithmischen Chemie unter Verwendung der Totalsynthese.

Parameteroptimierung von Interesse. Für diesen Prozess werden zum Teil über tausend Evolutionsläufe mit mehreren hunderttausend Bewertungen algorithmischer Chemien benötigt.

Die Einführung der Totalsynthese[4] zielt auf die vollständige Synthetisierung eines Datenflusses in kürzester Zeit, die ansonsten nur durch eine unendliche Anzahl ausgeführter Reaktionen sichergestellt werden kann (keine zyklische Verwendung von Molekülen vorausgesetzt). Dabei soll der Zufall in der Wahl der Reaktionen so wenig wie möglich eingeschränkt werden.

Die Totalsynthese entspricht dem rekursiven Aufbau des Datenflusses. Der in einem Individuum kodierte Datenfluss besitzt eine Mehrdeutigkeit, die erhalten bleiben soll, um die Vielfalt der Verhaltensvarianten bewerten zu können. Aus diesem Grund wird die Totalsynthese für jedes Fallbeipiel neu durchgeführt. Bei ihr handelt es sich nicht, wie etwa bei der Intronentfernung, um einen einmaligen Vorverarbeitungsschritt.

Algorithmus 5.1 stellt den Rahmen dar, in dem die Auswertung stattfindet. Die für die Interaktion vorgesehenen Moleküle stehen zunächst nicht den Reaktionen zum direkten Zugriff zur Verfügung. Ihr Zustand wird durch Aufruf der Funktion `recEval` (siehe Alg. 5.2) rekursiv ermittelt. Die Terminalmenge T enthält die Zustände der während der Ausführung nicht veränderbaren Moleküle. Diese enthalten die evolvierten Zustände und die Eingabewerte.

In einem Vorverarbeitungsschritt werden die Reaktionen eines Individuums anhand ihres Produkts α in Teilmultimengen S_α eingeteilt. Dieser Schritt ist für jedes Individuum nur einmal zu Beginn der Auswertung notwendig ist. Das Molekül α_r, dessen Zustand als Ergebnis interpretiert wird, wurde per Evolution bestimmt

[4] Die Totalsynthese beschreibt in der Chemie die vollständig künstliche Zusammensetzung organischer Moleküle aus einfachen Grundsubstanzen.

Input : Molekül α, dessen Zustand zu ermitteln ist, gruppierte Liste $\mathcal{S}$ von Reaktionsmultimengen, Menge T terminaler Moleküle, Menge B von Molekülen, deren Zustand im aktuellen Pfad bereits geändert wird
Output : möglicher Zustand des Moleküls α

```
1  if α ∈ T then
2      if α ∈ B then
3          Zustand ← 0 ;
4      else
5          Reaktion ← wähleZufällig(S_α);
6          C ← edukte(Reaktion);
7          Eduktezustände ← ∅;
8          foreach γ ∈ C do
9              Eduktezustände ← Eduktezustände ∪ recEval(γ, S, T, B ∪ {α});
10         end
11         Zustand ← ausführen(Reaktion, Eduktezustände);
12     end
13 else
14     Zustand ← ermittleZustand(α, T);
15 end
16 return Zustand
```

Algorithmus 5.2 : Algorithmus der Funktion `recEval`$(\alpha, \mathcal{S}, T, B)$ wie sie in Alg. 5.1 verwendet wird. Die Funktion ermittelt rekursiv den Zustand des Moleküls α unter Berücksichtigung des bisherigen Pfads B. $\mathcal{S}$ ist eine Liste von Reaktionsmultimengen, T ist die Menge der Terminale mit ihren Zuständen.

und ist daher bekannt. Da die Änderung seines Zustands die letzte relevante Zustandsänderung unter den Molekülen ist, beginnt der rekursive Aufbau des Datenflusses mit dem Molekül α_r (Alg. 5.1, Zeile 5).

Das erste Argument α der Funktion `recEval` (Alg. 5.2) gibt das Molekül an, dessen Zustand ermittelt werden soll. Damit lässt sich die Multimenge $\mathcal{S}_\alpha$ von Reaktionen ermitteln, die seinen Zustand verändern, aus ihnen wird eine Reaktion zufällig gewählt. Diese Reaktion benötigt Edukte, deren Zustände vor der Ausführung rekursiv ermittelt werden müssen, wenn diese nicht Element von T sind. Andere Systeme, wie das graphbasierte PDGP [90], arbeiten mit einer ähnlichen rekursiven Auswertung tatsächlich benötigter Komponenten.

Um Endlosschleifen zu vermeiden, speichert die Pfadmenge B jene Moleküle, die im aktuellen Datenflusspfad bereits verändert wurden. Wird der Zustand eines Moleküls dieser Menge ein weiteres Mal benötigt, so wird der Wert 0 zurück gegeben. Die Rückgabe des Werts 0 an dieser Stelle geschieht als Kompromiss, um unendliche Zyklen zu vermeiden, sie aber auch nicht durch eine letale Bewertung des Individuums gänzlich zu verbieten. Letzteres wird z. B. bei der grammatikalischen Evolution verwendet(siehe Kap. 1.2.3). Kann bei ihr ein Phänotyp nicht vollständig ausgebildet werden, wird dem Individuum der schlechteste mögliche

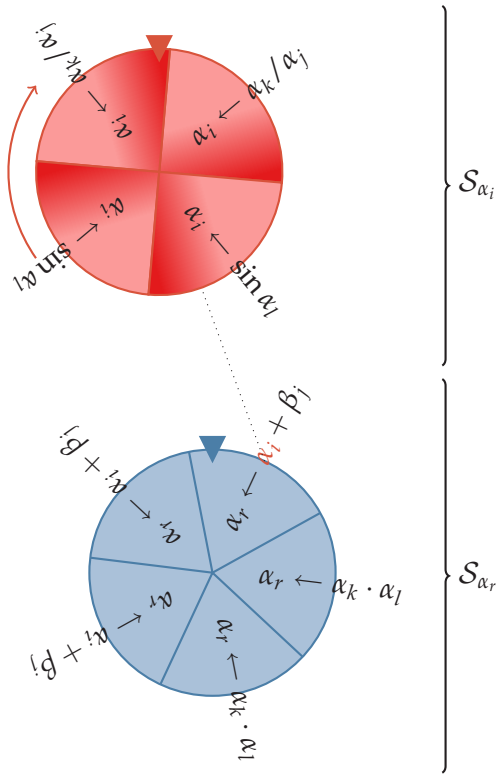


Abbildung 5.6.: Beispiel für den Ablauf bei der Totalsynthese. Ist das ergebnistragende Molekül α_r, so wird zunächst eine Reaktion, die seinen Zustand ändert, zufällig gewählt. Zur Belegung ihrer variablen Edukte (hier α_i, β_j sei eine Konstante) müssen in der Rekursion weitere Instruktionen gewählt werden.

Fitnesswert zugewiesen. In diesem Vorgehen liegt ein wesentlicher Unterschied zwischen der Totalsynthese und der klassischen Ausführung mit einer großen Anzahl wiederholter Ziehungen, bei der solche Zyklen nicht unterbunden werden, die Anzahl der „Schleifendurchläufe“ andererseits aber auch nicht definiert ist[5].

Die Messung der zu erwartenden Anzahl Zykleneintritte wird in Kapitel 6.3 besprochen. Ein erneuter Versuch, den Zustand zu ermitteln, wird in der Rekursion unterbunden, in parallelen Zweigen findet jedoch weiterhin eine neue Bestimmung des Zustands statt, da hier durchaus eine andere Reaktion den Zustand desselben Moleküls bestimmen kann.

Abbildung 5.6 stellt noch einmal die Grundidee der Totalsynthese grafisch dar. Angefangen bei dem per Evolution bestimmten Molekül α_r werden rekursiv die Reaktionen betrachtet, die zur Bestimmung seines Zustands notwendig sind. Können dabei Zustände von mehr als einer Reaktion beeinflusst werden, so wird eine Reaktion zufällig gewählt.

[5]Durch Ersetzung der Pfadmenge B mit einem Zugriffzähler wäre bei der Totalsynthese die Ausführung einer definierten Anzahl „Schleifendurchläufe“ möglich.

Um den Ausführungsaufwand in der linearen GP-Variante des Systems zu reduzieren, kommt die Entfernung von Introns in der Form zum Einsatz, wie sie von Brameier und Banzhaf [25] vorgeschlagen wurde.

6. Eigenschaften

Die Anzahl möglicher Phänotypen, auf die eine Chemie abgebildet werden kann, hängt von zwei Faktoren ab. Zum einen von der Diversität in Reaktionen mit identischen Produktmolekül, und zum anderen von der Anzahl zyklischer Verwendungen der Moleküle. Für beide Faktoren werden im Folgenden Maße entwickelt, um später den Einfluss der Evolution auf die Eindeutigkeit der evolvierten Chemie bewerten zu können. Des Weiteren werden die Höhe und die Größe sowie die Visualisierung der evolvierten Lösungen betrachtet.

6.1. Moleküleinfluss

Nicht jedes Molekül findet gleichermaßen Verwendung in den möglichen Verhaltensvarianten eines Individuums und nicht alle Datenflusspfade sind gleichwahrscheinlich. Die vorgestellten Maße sollen dieses berücksichtigen. Im Folgenden wird zunächst die erwartete Häufigkeit für die Verwendung eines Moleküls definiert. Anschließend wird eine Datenstruktur präsentiert, mit der die Erwartungswerte für alle Moleküle ermittelt werden. Auf sie wird bei anderen Maßen zurückgegriffen.

Moleküleinfluss

Der Einfluss eines Moleküls entspricht der erwarteten Häufigkeit mit der sein Zustand in Berechnungen, die in das Endergebnis eingehen, verwendet wird. Es seien α und β Moleküle, die als Produkt verwendet werden dürfen. Der Zustand des Moleküls β wird als Ergebnis interpretiert. $U(\alpha)$ ist die Menge der unterscheidbaren Reaktionen, die den Zustand von α als Edukt zur Bestimmung des Zustands eines anderen Moleküls verwenden. $R(\alpha)$ ist auf der anderen Seite die Menge der unterscheidbaren Reaktionen, die α als Produkt verwenden. Ist r eine Reaktion, so ist $t(r)$ das Molekül, dessen Zustand von r geändert wird und $f(r)$ ist die Häufigkeit dieser Reaktion in der betrachteten Chemie. Ob der Zustand eines Moleküls α mit einer erwarteten Häufigkeit größer 0 in den Rückgabewert der Chemie eingeht, hängt davon ab, ob von α ausgehend ein Pfad von Reaktionen existiert, dessen letzte Reaktion den Zustand von β ändert. Der Erwartungswert für die Haufigkeit $F(\alpha, C)$ ist rekursiv definiert und beschreibt „die Suche nach β“. Um die Terminierung sicherzustellen, ist der bisherige Pfad in Form der Menge C der veränderten Molekülzustände ein Parameter dieser rekursiven Funktion. Eine Rekursion erfolgt nur dann, wenn das aktuell betrachtete Molekül ungleich β ist und noch nicht

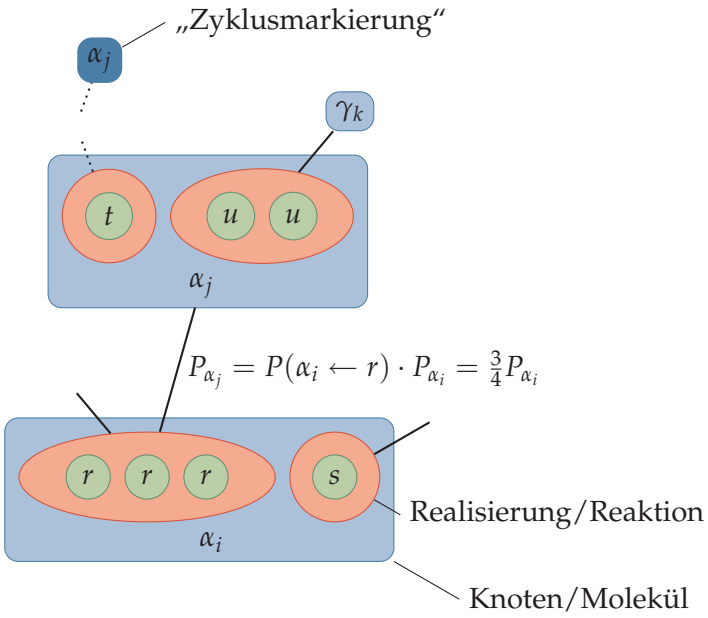


Abbildung 6.1.: Baum möglicher Reaktionen

betrachtet wurde ($\notin C$):

$$F(\alpha, C) = \begin{cases} 0 & \text{wenn } \alpha \in C, \\ 1 & \text{wenn } \alpha = \beta, \\ \sum\limits_{\forall r \in U(\alpha)} \frac{f(r)}{\sum\limits_{\forall s \in R(t(r))} f(s)} \cdot F(t(r), C \cup \alpha) & \text{sonst.} \end{cases}$$

Für die Ermittlung der zu erwartenden Häufigkeit $F(\alpha, \emptyset)$, mit der das Molekül α am Datenfluss teil hat, beginnt die Suche nach β bei α. Der bisher betrachtete Pfad $C = \emptyset$ ist somit zunächst leer. $F(\alpha, \emptyset)$ wird im Folgenden $F(\alpha)$ abgekürzt.

Reaktionsbaum

Während die rekursive Funktion oben von jedem einzelnen Molekül ausgehend versucht, über eine Kette von Reaktionen, das Ergebnismolekül β zu suchen, wird der Reaktionsbaum[1], dargestellt in Abb. 6.1, ausgehend von β als Wurzel aufgebaut. Jeder Knoten im Baum steht für die Verwendung eines Moleküls α. Dieses kann durch eine Menge unterschiedlicher Reaktionen $R(\alpha)$ beschrieben werden, wobei die Wahrscheinlichkeit für die Verwendung einer Reaktion $r \in R(\alpha)$

$$P(\alpha \leftarrow r) = \frac{f(r)}{\sum\limits_{\forall s \in R(\alpha)} f(s)} \tag{6.1}$$

[1]Der Graph als Struktur ist hier ungeeignet, da die Terminierung des Aufbaus in einem Zweig von den Knoten auf dem Pfad zur Wurzel abhängt.

von ihrer relativen Häufigkeit $f(r)$ in der Chemie abhängt. Jede Reaktion stellt eine mögliche Realisierung des Knotens dar, so kann jeder Knoten als Gruppe von Knoten für die möglichen Realisierungen betrachtet werden. Die Reaktionen verwenden in ihren Edukten weitere Moleküle, die Söhne der Realisierung darstellen. Die Kante zwischen Vater und Sohn wird mit der Wahrscheinlichkeit beschriftet, mit der der Molekülzustand des Sohnes, über den aktuell betrachteten Pfad, Eingang in das Endergebnis findet. Sie setzt sich zusammen aus dem Produkt der Wahrscheinlichkeit, mit der der Vater verwendet wird, und der Wahrscheinlichkeit, dass die Reaktion r den Zustand des Vaters bestimmt, d. h. die betrachtete Realisierung stattfindet.

Der Aufbau des Baumes wird in einem Knoten dann beendet, wenn:

- das Molekül, das er repräsentiert, konstant ist,
- der Zustand des Moleküls von der Eingabe an das System bestimmt wird oder
- das Molekül auf dem Pfad zur Wurzel bereits einmal verwendet wurde.

Im letzten Fall wird der Knoten als potenzieller Zykluseintritt markiert.

6.2. Unbestimmtheit

Der Zustand jedes Moleküls $\alpha \in A$ kann durch eine Menge unterschiedlicher Reaktionen $R(\alpha)$ bestimmt werden. Die Wahrscheinlichkeit für die Verwendung der Reaktion r beträgt $P(\alpha \leftarrow r)$ (siehe Gl. 6.1). Basierend auf dem Entropiemaß ergibt sich die Unbestimmtheit $E'(\alpha)$ des Moleküls α zu

$$E'(\alpha) = -\sum_{\forall r \in R(\alpha)} P(\alpha \leftarrow r) \log_{|R(\alpha)|} P(\alpha \leftarrow r) \quad .$$

Da nicht jedes Molekül gleichbedeutend für das Ergebnis der Chemie ist, berechnet sich die Unbestimmtheit $E(\mathcal{R})$ der algorithmischen Chemie aus der gewichteten und normierten Unbestimmtheit aller im Zustand änderbaren Moleküle A,

$$E(\mathcal{R}) = \frac{1}{\sum_{\forall \alpha \in A} F(\alpha)} \sum_{\forall \alpha \in A} F(\alpha) \cdot E'(\alpha) \quad .$$

Die Unbestimmtheit nimmt Werte aus dem Bereich $[0, 1]$ an. Der Wert 0 wird angenommen, wenn alle Moleküle jeweils von maximal einer Menge identischer Reaktionen in ihrem Zustand verändert werden. Ist die Chemie zyklusfrei, nehmen die Moleküle dann bei gleichem Initialzustand nach Ausführung unendlich vieler Reaktionen immer denselben Endzustand an. Die maximale Unbestimmtheit von 1 wird angenommen, wenn alle potentiell in das Endergebnis eingehenden Molekülzustände durch mindestens zwei unterschiedliche Reaktionen, die sich mit gleicher Häufigkeit in der Chemie befinden, geändert werden.

6.3. Zyklenanzahl

Zyklenanzahl

Die Zyklenanzahl entspricht der erwarteten Anzahl Zykleneintritte in einer evolvierten Lösung. Dieser Wert ist bei Verwendung der Totalsynthese in sofern hypothetischer Natur, als dass der Zyklus nicht betreten wird. Zur Ermittlung der Zyklenanzahl wird über alle als Zykluseintritt markierten Blätter im Reaktionsbaum iteriert und die Wahrscheinlichkeit der zu ihnen führenden Kanten aufsummiert.

6.4. Größe und Höhe

effektive Größe

Höhe

absolute Größe

Zwei Maße sollen den Aufwand für die Ausführung evolvierter Programme bewerten. *Die effektive Größe der Programme gibt die Anzahl der Instruktionen/Reaktionen an, die zur Berechnung des Ergebnisses benötigt werden.* Damit ist die effektive Größe ein Maß für den Berechnungsaufwand, der durch das Programm erzeugt wird. *Die Höhe des evolvierten Programms entspricht dem größten Niveau seines Datenflussgraphen/-baums, d. h. der Anzahl Knoten im längsten enthaltenen Pfad.* Die Sequenz nacheinander auszuführender Reaktionen bzw. Instruktionen gibt, auch bei beliebiger Anzahl Prozessoren, Aufschluss über die mindestens benötigte Ausführungsdauer[2]. *Die absolute Größe entspricht der Gesamtzahl der im Individuum codierten Instruktionen oder Reaktionen.* Hierzu zählen auch Introns, d. h. Instruktionen oder Reaktionen die nicht am Datenfluss beteiligt sind. Die Ermittlung der absoluten Größe vollzieht sich sowohl beim linearen GP als auch bei der algorithmischen Chemie durch Abzählen der codierten Instruktionen und Reaktionen im Genom des Individuums, dessen Länge variabel aber beschränkt ist.

Bei linearen GP Individuen entspricht die effektive Größe der Anzahl der im Individuum codierten Instruktionen nach Entfernung der Introns. Bei der algorithmischen Chemie entspricht sie der erwarteten Anzahl auszuführender Reaktionen bei der Totalsynthese. Ermittelt wird der Erwartungswert durch Aufsummieren der Wahrscheinlichkeiten, mit denen die inneren Knoten im Reaktionsbaum zur Ausführung gelangen. Die Wahrscheinlichkeit dafür, dass ein Knoten Teil des Datenflusses ist, ist in seiner zur Wurzel führenden Kante codiert. An der Wurzel beträgt diese Wahrscheinlichkeit 1.

Die für eine AC ermittelte effektive Größe kann die Individuengröße übersteigen, wenn Moleküle in unterschiedlichen Zweigen Verwendung finden. Da in beiden Zweigen die an der Zustandsbestimmung für dieses Molekül beteiligten Reaktionen unterschiedlich sein können, wird der zugehörige Datenfluss zweimal ermittelt. Mit zunehmender Bestimmtheit steigt die Wahrscheinlichkeit, dass die Berechnungen in beiden Zweigen identisch sind, und bei deterministischer Ausführung nur einmal erfolgen müsste.

[2] Es wird vereinfachend davon ausgegangen, dass jede Instruktion den selben Aufwand erzeugt.

Erwartungswertberechnung für die Höhe einer Algorithmischen Chemie

Da die verwendete Funktionsmenge keine Verzweigungsoperationen enthält, ist die Höhe des Datenflussgraphen linearer GP-Individuen eindeutig. Algorithmische Chemien hingegen können bei der Ausführung unterschiedliche Verhaltensvarianten zeigen. Die erwartete Höhe ergibt sich damit aus den Höhen der unterschiedlichen Varianten und ihren Wahrscheinlichkeiten. Für die Berechnung des Erwartungswerts wird angenommen, dass die Varianten zyklenfrei sind und vollständig ausgeführt werden. Für die Totalsynthese sind beide Annahmen erfüllt.

Die Berechnung der erwarteten Höhe basiert auf dem Reaktionsbaum ohne Betrachtung der Blätter, da diese keine Berechnung benötigen[3] (Abb. 6.1). Jeder Knotengruppe x entspricht der Zustandsbestimmung eines Moleküls $\alpha = m(x)$. Der Zustand kann durch eine Menge unterschiedlicher Reaktionen $R(\alpha)$ verändert werden, die dieses Molekül als Produkt verwenden. Das heißt sollte der Teilbaum bei der Ausführung ausgebildet werden, so wird eine der möglichen Reaktion $r \in R(\alpha)$ verwendet. Jede mit einer Wahrscheinlichkeit $P(\alpha \leftarrow r)$. Jeder Knoten x_r in der Knotengruppe x steht für eine mögliche Reaktion.

Die Wahrscheinlichkeitsverteilung für die Höhe $h(x)$ eines Teilbaums, mit dem inneren Knoten x als Wurzel, wird bestimmt durch die Wahrscheinlichkeitsverteilungen der Höhe $h'(x_r)$ in den möglichen Realisierungen:

$$P\big(h(x) = l\big) = \sum_{\forall r \in U(m(x))} P(\alpha \leftarrow r) \cdot P\big(h'(x_r) = l\big).$$

Die Wahrscheinlichkeitsverteilung für die Knoten des Baumes, deren Söhne nur noch aus Blättern bestehen, wird beschrieben durch

$$P\big(h(x) = l\big) = \begin{cases} 1 & \text{wenn } l = 1, \\ 0 & \text{sonst.} \end{cases}$$

Die Verteilung der Höhe einer einzelnen Realisierung x_r hängt ab von den Verteilungen in den Knoten, die zu den Edukten der entsprechenden Reaktion gehören. Ist K die Menge der Knoten, die beim Aufbau des Reaktionsbaums als Söhne der betrachteten Realisierung erzeugt werden, so berechnet sich die diskrete Wahrscheinlichkeitsverteilung der Höhe für diese Realisierung zu

$$P\big(h'(x_r) = l\big) = P\Bigg(\cup_{y \in K}\Big(\big(h(y) = l-1\big) \cap \big(\cup_{z \in K \setminus y}\big(h(z) \leq l-1\big)\big)\Big)\Bigg). \quad (6.2)$$

Das heißt das Ereignis, dass die Höhe des Teilbaums mit Wurzel x_r gleich l ist, besitzt die gleiche Wahrscheinlichkeit wie das Ereignis, dass mindestens einer seiner Teilbäume die Höhe $l-1$ hat und die anderen Teilbäume nicht höher sind.

Ist der Knoten w die Wurzel des Reaktionsbaums, da $\beta = m(w)$ das Molekül ist, dessen Zustand als Ergebnis interpretiert wird, so entspricht der Erwartungswert

[3] Es handelt sich bei ihnen um Eingaben, Konstanten oder den Eintritt in einen Zyklus.

Tabelle 6.1.: Beispielhafte Wahrscheinlichkeitsverteilung der Höhe $h'(x_r)$ von der möglichen Realisierung x_r im Knoten x. Sie ergibt sich aus den Verteilungen in beiden Operanden x_r^1 und x_r^2.

l	$P(h(x_r^1) = l)$	$P(h(x_r^1) \leq l)$	$P(h(x_r^2) = l)$	$P(h(x_r^2) \leq l)$	$P(h'(x_r) = l)$
1	0,18	0,18	0,08	0,08	0,00
2	0,27	0,45	0,23	0,31	0,01
3	0,45	0,91	0,15	0,46	0,13
4	0,00	0,91	0,00	0,46	0,28
5	0,09	1,00	0,23	0,69	0,00
6	0,00	1,00	0,31	1,00	0,27
7	0,00	1,00	0,00	1,00	0,31

der Höhe

$$E\big[h(w)\big] = \sum_{l=1}^{n} l \cdot P\big(h(w) = l\big).$$

Zyklen werden im Reaktionsbaum nicht ausgebildet. Aus diesem Grund ist die Höhe beschränkt durch die Anzahl der in ihrem Zustand veränderbaren Moleküle.

Da ausschließlich Funktionen der Arität eins oder zwei verwendet werden, besitzen die Realisierungen innerer Knoten maximal zwei Söhne. Die Gleichung (6.2) lässt sich somit vereinfachen, wobei zunächst der Fall mit zwei Söhnen $K = \{x_r^1, x_r^2\}$ betrachtet wird:

$$\begin{aligned} P\big(h'(x_r) = l\big) &= P\Big(\big((h(x_r^1) = l-1) \cap (h(x_r^2) \leq l-1)\big) \\ &\quad \cup \big((h(x_r^2) = l-1) \cap (h(x_r^1) \leq l-1)\big)\Big) \\ &= P\big(h(x_r^1) = l-1\big) \cdot P\big(h(x_r^2) \leq l-1\big) \\ &\quad + P\big(h(x_r^2) = l-1\big) \cdot P\big(h(x_r^1) \leq l-1\big) \\ &\quad - P\big(h(x_r^1) = l-1\big) \cdot P\big(h(x_r^2) = l-1\big). \end{aligned}$$

Im Fall von nur einem Sohn x' entspricht die Wahrscheinlichkeitsverteilung der Höhe des Teilbaums mit Wurzel x_r der um 1 verschobenen Verteilung des Teilbaums mit Wurzel x':

$$P\big(h'(x_r) = l\big) = P\big(h(x') = l-1\big) \quad .$$

Zur Verdeutlichung enthalten Tab. 6.1 und Abb. 6.2 die Wahrscheinlichkeitsverteilung für eine fiktive Realisierung x_r mit zwei Söhnen x_r^1 und x_r^2, bei denen die Wahrscheinlichkeitsverteilung gegeben ist. Das Beispiel verdeutlicht einige Spezialfälle. So ist für die Knoten x_r^1 und x_r^2 die Wahrscheinlichkeit 0, dass die bei

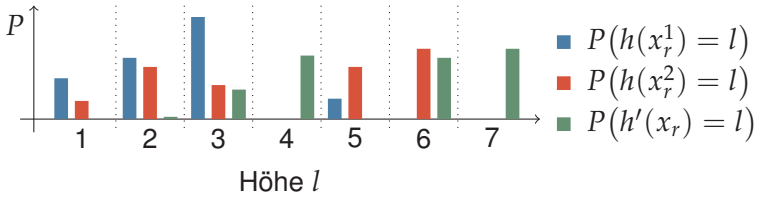


Abbildung 6.2.: Wahrscheinlichkeitsverteilung der maximalen Höhe einer in x_r wurzelnden Realisierung eines Knotens x. Die Verteilung ist abhängig von den Verteilungen in den Söhnen x_r^1 und x_r^2 der Realisierung x_r.

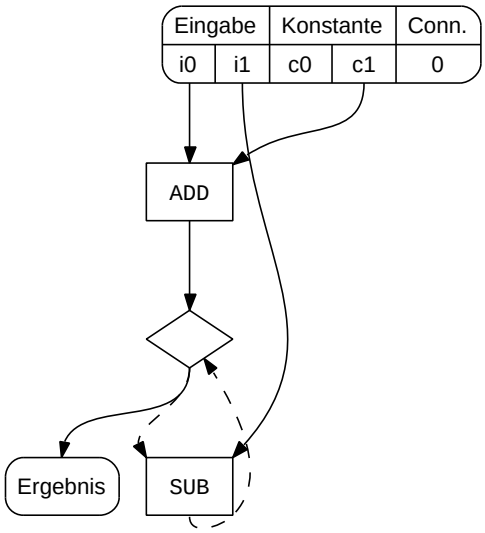


Abbildung 6.3.: Elemente für die Visualisierung von algorithmischen Chemien und linearen Individuen.

ihnen beginnenden Teilbäume die Höhe 4 haben. Hieraus folgt, dass die weiterverarbeitende Realisierung x_r nicht die Höhe 5 haben kann. Zudem hat von beiden Operanden nur der Knoten x_r^2 eine Wahrscheinlichkeit größer 0 für die maximal mögliche Höhe 6. Wird sie realisiert, ist die Auswirkung auf die Höhe von x_r unabhängig von Ereignissen im Knoten x_r^1, der diese Höhe nicht mehr „überbieten“ kann. Die Wahrscheinlichkeit, dass x_r die Höhe 7 hat, ist damit identisch mit der Wahrscheinlichkeit, dass x_r^2 die Höhe 6 hat.

6.5. Visualisierung

Der Datenfluss einer evolvierten Lösung lässt sich als gerichteter Graph darstellen (Abb. 6.3). *Rechteckige Knoten* entsprechen den Operationen aus den Reaktionen bzw. Instruktionen der Individuen und sind mit der enthaltenen Operation beschriftet.

Die eingehenden Kanten bestimmten die Herkunft der Operanden. Ist das Ergebnis der Operation abhängig von der Reihenfolge der Operanden, so entspricht die Reihenfolge der eingehende Kanten (von links nach rechts) der Reihenfolge der Operanden. Senken und Quellen des Graphen sind durch *Rechtecke mit abgerundeten Kanten* dargestellt. Bei der Senke handelt es sich um das Register oder Moleküle, dem zum Abschluss der Berechnung das Ergebnis entnommen wird. Die Quelle enthält alle Werte die von den einzelnen Operationen während der Berechnung gelesen aber nicht verändert werden. Zum einen handelt es sich dabei um die Eingabe, wie sie z. B. während des Trainings aus den betrachteten Fallbeispielen stammt. Zum anderen handelt es sich um die konstanten Werte, die zusammen mit der Lösung im Individuum evolviert werden. Die letzte der möglichen Quellen stammt von Registern oder Molekülen, die zwar in den Operationen Verwendung finden, aber nie durch eine andere Operation mit einem Wert belegt werden. Sie tragen den initialen Wert 0 in sich.

Graphen, die Lösungen aus algorithmischen Chemie darstellen, enthalten darüber hinaus Rauten als mögliche Knotenform und gestrichelte Kanten. Eine *Raute* steht für ein Molekül, dessen Zustand von unterschiedlichen Reaktionen bestimmt wird. Sie stellt damit eine durch den Zufall getroffene Entscheidung dar. Die ausgehenden Kanten übermitteln einen zufällig aus den eingehenden Kanten gewählten Wert. Die Wahl kann für jede ausgehende Kante neu getroffen werden. *Gestrichelte Linien* weisen darauf hin, dass der Zustand eines Moleküls über eine Kette von Reaktionen aus seinem eigenen Zustand bestimmt wird. Gestrichelte Linien sind damit Teil eines Zyklus.

Die Berechnung von Eigenschaften, wie der Größe der Lösung, geht von der pessimistischen Annahme aus, dass ein zu unterschiedlichen Zeitpunkten verwendetes Molekül bei jeder Verwendung einen unterschiedlichen Zustand besitzt. Dieses kann aufgrund unterschiedlich fortgeschrittener Assemblierung der Fall sein. Bei der Überführung einer Verhaltensvariante in eine deterministische Lösung, wird jede Verwendung eines Molekül unabhängig voneinander betrachtet. Während die Berechnungen von Größe und Höhe daher auf einer Betrachtung als Baum basieren, wird für die Visualisierung von algorithmischen Chemien die kompaktere Darstellung als Graph verwendet.

Teil III.

Design der Experimente

7. Parameter: Bestimmung und Effektermittlung

Ein fairer Vergleich unterschiedlicher Algorithmen setzt die adäquate Parametrisierung all jener Algorithmen voraus, die im Rahmen dieses Vergleichs zur Ausführung gebracht werden. Bei neuen Algorithmen sind solche Parameter meist nicht gegeben, für andere, am Vergleich teilnehmenden Verfahren, ist die Ermittlung in der Literatur angegebener Parameter meist unbekannt oder eine erfolgreiche Übertragung auf eine abweichende Problemstellung nicht garantiert.

Zur Bestimmung der Parameter wird hier die sequenziellen Parameteroptimierung (SPO) von Bartz-Beielstein [18] verwendet. Dabei handelt es sich um ein Verfahren zur Parametrisierung von nichtdeterministischen Algorithmen im Hinblick auf eine zu definierende Reaktion und um Techniken zur Analyse der dabei gewonnenen Daten. Die SPO setzt auf Verfahren zum Design und Analyse von (deterministischen) Computerexperimenten (DACE) [100] auf und erweitert sie um klassische, statistische Methoden und einfache Heuristiken, um dem Nichtdeterminismus des betrachteten Algorithmus Rechnung zu tragen. Das Design und die Analyse mittels SPO ermöglicht es, schnell und systematisch einen Eindruck vom Potenzial und den relevanten Stellgrößen eines Algorithmus zu bekommen.

Die SPO betrachtet zu Beginn einige zufällige Einstellungen des Algorithmus. Auf der Grundlage der Resultate mit allen bisher betrachteten Einstellungen werden dann in sequentiellen Schritten weitere vielversprechende Einstellungen verwendet und die Konfidenz an bereits betrachteten Einstellungen durch weitere Experimente erhöht. In jedem Schritt sieht die SPO ursprünglich nur die Erhöhung der Anzahl Experimente für die Beste der bisher betrachteten Einstellungen vor. Haben die Experimente innerhalb einer Einstellung eine hohe Varianz, besteht die Gefahr, dass die beste Einstellung nicht selektiert wird, wenn sie zu Beginn viele schlechte Ergebnisse erzeugt. Zudem benötigt es viele Iterationen um die vermeintlich guten Einstellungen als ungeeignet zu erkennen. Dieses erwies sich bei den hier betrachteten Algorithmen zum Teil als problematisch. Daher wurde die SPO in dieser Schrift um das von Chen u. a. [30] vorgestellte Verfahren zur „optimal computing budget allocation“ (OCBA) ergänzt, welches in jedem Schritt weitere Experimente auf alle bekannten Einstellungen verteilt, mit dem Ziel die Wahrscheinlichkeit für die korrekte Wahl der besten Einstellung am Schluss zu erhöhen.

Bei der Anwendung der SPO lassen sich fünf Phasen unterscheiden, die in den folgenden Abschnitten beschrieben sind:

1. Die Festlegung der problemspezifischen Einstellung und des betrachteten Rahmens der Optimierung (Abschnitt 7.1),

2. die Bestimmung eines initialen Designs und die Bewertung der hierzu gehörenden Designpunkte (Abschnitt 7.2),
3. die Modellbildung (Abschnitt 7.3),
4. die Auswahl sequenzieller Designs und ihre Evaluierung (Abschnitt 7.5) und
5. die Analyse der erhobenen Daten (Abschnitt 7.6).

Die Punkte 3 und 4 bilden die Grundlage der sequenziellen Schritte, die wiederholt werden, bis ein Experimentbudget aufgebraucht ist.

7.1. Algorithmen- und Problemdesign

Designpunkt

Design

Problemdesign

Zentrales Element der SPO ist das Design von Experimenten durch eine Menge von Designpunkten. *Ein Designpunkt bestimmt genau eine Einstellung aller Faktoren, die die Reaktion des Algorithmus beeinflussen.* Die Menge aller Faktoren spannt den Raum $\mathcal{R}$ auf. Dies ist der Suchraum der SPO. *Ein Design* $\mathcal{D} = \left\{x^{(1)}, \ldots, x^{(n)}\right\}$ *eines Experiments beschreibt eine Menge von Designpunkten* $x^{(i)} \in \mathcal{R}$ *mit denen der Algorithmus ausgeführt werden soll.* In Computerexperimenten kann der Zustand sämtlicher Faktoren festgelegt werden. Die Faktoren lassen sich anhand der Fragestellung unterscheiden, ob sie während der SPO variiert oder fixiert werden sollen. *Die fixierten Faktoren eines Algorithmus bilden das Problemdesign.* Das Problemdesign besteht während eines SPO-Laufs aus genau einem Punkt $x_P \in \mathcal{R}_\mathrm{P}$. In Abhängigkeit davon ist nun eine Einstellung $x = (x_1, \ldots, x_d)^\mathrm{T} \in \mathcal{R}_\mathrm{A}$ der verbleibenden d Faktoren gesucht, sodass die Leistung des Systems bzgl. einer Reaktion $Y(x_P, x)$ möglichst optimal ist. Diese Faktoren werden im Folgenden auch als *Parameter des Systems* bezeichnet, da sie die Schnittstelle zwischen der SPO und dem Algorithmus darstellen.

$\mathcal{R}_\mathrm{A}$ beschreibt den betrachteten Definitionsbereich der Parameter. Bei der Betrachtung nichtdeterministischer Algorithmen ist die *Reaktion* Y eine stochastische Variable, die aufgrund der Festlegung von x_P im Folgenden verkürzt mit $Y(x)$ bezeichnet wird. Die *Güte* eines Designpunkts $x^{(i)}$ wird aus allen mit ihm durchgeführten Wiederholungen geschätzt, zumeist wird hier die mittlere Reaktion $\overline{Y}(x^{(i)})$ verwendet. Im Folgenden wird die Minimierung dieses Werts als Ziel der SPO angenommen.

Einige Faktoren sind bei Anwendung der SPO offensichtlich Teil des Problemdesigns, wie die Wahl der vom Algorithmus bearbeiteten Aufgabenstellungen oder der maximale Aufwand[1]. Bei anderen Faktoren, die die Reaktion des Algorithmus beeinflussen, ist die Zuordnung abhängig vom aktuellen Fokus der Betrachtung. Ist zum Beispiel die Populationsgröße μ innerhalb definierter Grenzen veränderlich

[1] Hier ist bei Suchheuristiken darauf zu achten, dass für einen fairen Vergleich, bei im Rahmen der Optimierung variierten Faktoren, u. U. unterschiedliche Terminierungszeitpunkte (z. B. in Generationen) gewählt werden müssen, wenn diese ansonsten einen unterschiedlichen Aufwand (z. B. in Evaluationen) erzeugen würden. Der erlaubte Aufwand ist ein festgelegter Faktor im Problemdesign.

und soll eine Einstellung während der SPO bestimmt werden, so wird μ von der SPO als Parameter betrachtet. Findet hingegen eine Festlegung statt, so ist die Populationsgröße Teil des Problemdesigns. Im weitesten Sinne kann auch die Realisierung des Algorithmus als Faktor betrachtet werden, der dem Problemdesign zuzuordnen ist, denn die Umsetzung hat natürlich Einfluss auf die Reaktion des Systems, subsummiert sie doch alle nicht bewusst wahrgenommenen Faktoren, z. B. Wahl der Rekombinationsoperation bei einem evolutionären Algorithmus.

Eine vollständige und nachvollziehbare Beschreibung des Problemdesigns entzieht sich der Einflussnahme eines Verfahrens wie der SPO und es bleibt in der Verantwortung des Benutzers diese zu dokumentieren. Vorgaben macht die SPO jedoch bei der Beschreibung der Parameter, durch die Notwendigkeit zur Festlegung von $\mathcal{R}_\mathrm{A}$, aus dem das Algorithmendesign stammt. *Während der SPO bildet das Algorithmendesign die Menge möglicher Parametrisierungen, die in einem Schritt betrachtet werden. Nach der SPO bezeichnet das Algorithmendesign die von der SPO gefundenen und in den Experimenten verwendete Parametrisierung des Algorithmus.* Die Dokumentation von $\mathcal{R}_\mathrm{A}$ ist für die Anwendung der SPO notwendig, da das Verfahren in seinen Grenzen agiert. Der durch die Bestimmung von $\mathcal{R}_\mathrm{A}$ gesetzte Fokus wird im Folgenden auch „*region of interest*“ (ROI) genannt. Eine vollständige Beschreibung besteht aus der Liste der d Parameter p_i, deren zugehöriger Definitionsmenge $D_i \in \{\mathbb{R}, \mathbb{N}\}$ sowie den Grenzen $[\min_i, \max_i]$ in denen er betrachtet werden soll (Tab. 8.1 enthält ein erstes Beispiel). Die Definitionsmenge ist für die SPO nur von untergeordneter Bedeutung, die entsprechenden Parameter werden intern als reellwertig betrachtet und lediglich bei der Bestimmung von Designpunkten durch Rundung korrigiert.

Algorithmendesign

Im Folgenden werden nur quantitative Faktoren betrachtet. Für die Betrachtung ordinaler oder gar nominaler Faktoren eignen sich unter anderem Regressionsbäume, wie von Bartz-Beielstein und Markon [19] für den Einsatz bei der Parametrisierung von Suchalgorithmen beschrieben. Außer den Grenzen der Definitionsbereiche sind keine weiteren Beschränkungen vorgesehen. Einfache Beschränkungen, die durch Abhängigkeiten der Parameter untereinander entstehen, können häufig extern auflöst werden. So arbeitet die Evolution einer algorithmischen Chemie mit einer (μ, λ)-Strategie, die nur für $\lambda \geq \mu$ definiert ist. Diese Beschränkung lässt sich auflösen, indem die Nachkommenzahl $\lambda = \mu \cdot \nu$ aus der Elternzahl μ und dem Selektionsdruck ν berechnet wird.

7.2. Initiales Design

Bei der Bestimmung eines Algorithmendesigns $\left\{x^{(1)}, \ldots, x^{(n)}\right\} \in \mathcal{R}_\mathrm{A}$ bei SPO wird die Wahl des initialen Designs von der Designwahl in den sequenziellen Schritten unterschieden. Die Menge der *initialen Designpunkte* muss zwei Anforderungen genügen. Zum einen muss die Kardinalität des Designs groß genug sein, um aus den Punkten die Parameter des in den sequenziellen Schritten verwendeten Modells schätzen zu können. Zum anderen sollte sie eine gute Abdeckung von $\mathcal{R}_\mathrm{A}$ darstellen.

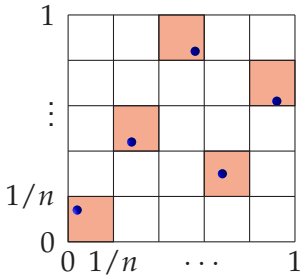


Abbildung 7.1.: Ein mögliches Design der Größe $n = 5$ durch Latin Hypercube Sampling für den zweidimensionalen Fall.

Die Größe des initialen Designs wird in Abhängigkeit der Parameteranzahl d gewählt zu

$$\max\left(11d, 2 + 1\frac{1}{2}d + \frac{1}{2}d^2\right).$$

Der rechte Term erklärt sich aus den Notwendigkeiten, die für die Modellbildung und -validierung entstehen (siehe Kap. 7.4). Der linke Term stammt aus den von Schonlau u. a. [104] beschriebenen Erfahrungen bei der globalen Optimierung. Von ihnen wird eine „$10d$-Regel“ verwendet und für den hier betrachteten vergleichsweise niedrigdimensionalen Raum so angepasst, dass sie am ehesten einer „$11d$-Regel“ entspricht.

Die initialen Designpunkte für den Algorithmus werden experimentell ausgewertet, indem der Algorithmus, unter Einhaltung des Problemdesigns, mit den entsprechenden Einstellungen gestartet wird. Wie bereits eingangs erwähnt, gelangt jeder Designpunkt mehrfach zur Auswertung, unter Verwendung variierender Seeds für den Zufallszahlengenerator. Die *initiale Anzahl Wiederholungen* n_0 ist für alle Punkte des initialen Designs identisch und sollte zwischen 5 und 20 liegen, da aus den Ergebnissen in einem späteren Schritt der Mittelwert und die Varianz geschätzt werden.

Latin Hypercube Sampling

Zu Sicherstellung einer guten Abdeckung des Suchraums bedient sich SPO eines raumfüllenden Designs und seiner Ermittlung durch Latin Hypercube Sampling (LHS) [75]. Das LHS der Größe n teilt für jede Dimension den betrachteten Definitionsbereich (hier $[0, 1]$) in n gleichgroße Intervalle $[0, 1/n], [1/n, 2/n], \ldots, [(n-1)/n, 1]$ auf und ordnet jedes Intervall zufällig einem Designpunkt zu. Aus dem Intervall wird dann gleichverteilt der Wert des Parameters gewählt. Diese Zuordnung geschieht für jede Dimension unabhängig voneinander. Abbildung 7.1 enthält ein Beispiel für die Verteilung von fünf Designpunkten in zwei Dimensionen. Diese Art der Zuordnung stellt bereits eine nahezu Gleichverteilung in jeder Dimension sicher. Um die Verteilung im Raum weiter zu verbessern, werden zumeist mehrere Stichproben erzeugt und eine bzgl. eines Optimierungskriteriums ausgewählt. Bei

SPO ist dieses Kriterium die Maximierung des minimalen, euklidischen Abstands zwischen zwei Punkten einer Stichprobe, ein anderes Kriterium wäre z. B. die Minimierung der Korrelation.

Für eine von den Größen der Definitionsbereiche unbeeinflussten Abstandsbestimmung arbeitet SPO zunächst in jeder Dimension mit dem Einheitsintervall $[0, 1]$ und transformiert erst abschließend die gewählte Punktemenge zur Bestimmung der Parameter in den Raum $\mathcal{R}_{\mathrm{A}}$.

7.3. Modellbildung

Diese initial betrachteten Designpunkte ermöglichen einen ersten Eindruck auf das Verhalten des Algorithmus. Die Suche nach einer guten Einstellung erfolgt nun in sequenziellen Schritten. Da die Auswertung eines Designpunktes je nach Algorithmus, verwendetem Problemdesign und der Anzahl Wiederholungen sehr aufwendig sein kann, sollen die weiter betrachteten Designpunkte sowohl in ihrer Anzahl gering, als auch in ihrer Güte vielversprechend sein. Die Wahl weiterer Designpunkte findet daher auf der Grundlage eines Modells statt, welches das Verhalten des Algorithmus approximiert. Es handelt sich dabei um die sogenannte Kriging[2] Approximation, die durch Anpassung eines stochastischen Prozessmodells geschieht.

Jones u. a. [54] zählen mit der Geologie, der globalen Optimierung und der Statistik die drei großen Gebiete auf, in denen stochastische Prozessmodelle eine lange Geschichte haben (Literaturhinweise siehe dort). Auf dem Gebiet der Approximation von Reaktionen aus Computerprogrammen hat sie ihren Ursprung bei Sacks u. a. [100] in der Verwendung für DACE. Sie wurde seitdem für weitere Anwendungsgebiete erschlossen. Trosset und Padula [115] nutzen die Modelle um Klassen schnell berechenbarer, pseudo zufälliger Zielfunktionen für Suchheuristiken zu erzeugen, Emmerich u. a. [41, 42] setzen sie in ihrem evolutionären Algorithmus zur Reduktion der Zielfunktionsauswertungen ein.

Die stochastischen Prozessmodelle aus DACE erwarten für jeden ausgewerteten Designpunkt $x^{(i)}$ genau eine Reaktion, die sie exakt interpolieren. Aufgrund der Betrachtung nichtdeterministischer Algorithmen gelangt hier jeder Designpunkt mehrfach zur Auswertung. Er wird, wenn nicht anders beschrieben, durch den Mittelwert $\overline{Y}(x^{(i)})$ aus der Menge seiner Auswertungen repräsentiert. In Anlehnung an die klassische DACE-Literatur und um zu verdeutlichen, dass die Reaktion für das Modell als deterministisch interpretiert wird, wird dieser Wert im Folgenden mit $y(x^{(i)})$ bezeichnet.

Gegeben sind n unterschiedliche Designpunkte $x^{(i)} = \left(x_1^{(i)}, \ldots, x_d^{(i)}\right) \in \mathcal{R}_{\mathrm{A}}$, mit $i = 1, \ldots, n$ und die zugehörigen Repräsentanten $y\left(x^{(i)}\right)$ der Reaktionen. Die Verwendung einer linearen Regression impliziert die Annahme, das Modell,

[2] Benannt nach dem südafrikanischen Ingenieur Daniel G. Krige.

welches die Beobachtung in Form der Repräsentanten generiert hat, habe die Form:

$$\sum_{j=1}^{q} \beta_j f_j(x) + \epsilon^{(i)},$$

wobei die f_j beliebige Funktionen von x und β_j die q zu schätzende Regressionskoeffizienten sind. Die $\epsilon^{(i)}$ sind normalverteilte, unabhängige Fehlerterme. Im einfachsten Fall besteht dieser Ausdruck aus dem konstanten Regressionsterm μ. Lineare Regression hat den Nachteil, dass zunächst die funktionale Struktur in Form der f_j spezifiziert werden muss. Für den Anwendungsbereich der SPO gibt es keinen Grund zu der Annahme, dass vor der Anwendung diese Struktur bekannt ist. Sehr variable Modelle scheitern daran, dass zur Schätzung der Koeffizienten sehr viele Designpunkte ausgewertet werden müssen, der Aufwand steigt linear mit der Anzahl der Terme. SPO verwendet daher die lineare Regression nur zur Ermittlung eines (hier: quadratischen) Trends, die Regressionsterme werden mithilfe der Methode der kleinsten (Fehler-)Quadrate geschätzt.

Von dem nicht erklärten Anteil[3] wird angenommen, dass er durch fehlende Regressionsterme zustande kommt und nicht durch ein Rauschen $\epsilon^{(i)}$. Er wird als *räumlich geordneter stochastischer Prozess* $Z(x)$ mit dem Mittelwert null und der Prozessvarianz σ^2 modelliert. Es ergibt dann als Gesamtmodell:

$$Y(x) = \sum_j \beta_j f_j\left(x^{(i)}\right) + Z(x).$$

Die Varianz an einem Designpunkt $x^{(i)}$ ist bekannt und entspricht der ortsabhängigen Abweichung $\epsilon\left(x^{(i)}\right)$ der linearen Regression von $y(x^{(i)})$.

SPO geht davon aus, dass der Einfluss der Parameter auf die Reaktion des Algorithmus kontinuierlich ist, womit auch der durch die Regression nicht erklärte Anteil einen kontinuierlichen Verlauf hat. Das heißt ein Punkt x der nahe bei einem bekannten Designpunkt $x^{(i)}$ liegt hat eine mit $\epsilon\left(x^{(i)}\right)$ vergleichbare Abweichung $\epsilon(x)$. Die Punkte sind korreliert. Die Korrelation nimmt mit der Distanz ab. Dementsprechend ergibt sich im Prozessmodell die *Kovarianz* zweier Punkte x und $x^{(i)}$ aus der Prozessvarianz σ^2 und einem noch näher zu spezifizierenden Korrelationsterm $\operatorname{corr}[\epsilon(x), \epsilon\left(x^{(i)}\right)]$ zu

$$\operatorname{Cov}\left(x, x^{(i)}\right) = \sigma^2 \operatorname{corr}\left[\epsilon(x), \epsilon\left(x^{(i)}\right)\right].$$

Die Korrelation berechnet sich über die gewichtete Distanz

$$\delta\left(x, x^{(i)}\right) = \sum_{k=1}^{d} \theta_k \left(x_k - x_k^{(i)}\right)^2$$

[3] Der Terminus Regressionsfehler wird hier vermieden, da der Fehlerbegriff ansonsten zu überfrachtet ist.

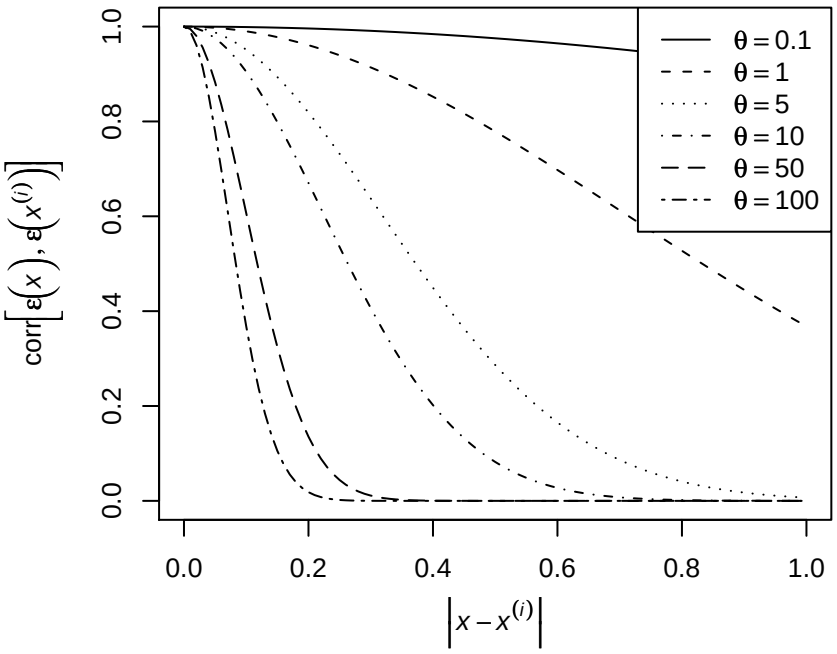


Abbildung 7.2.: Angenommene Korrelation der Abweichung vom zugrunde liegenden Trend an den Punkten x und $x^{(i)}$ in Abhängigkeit ihrer Distanz im Einheitsintervall und dem Korrelationsparameter θ (eindimensionaler Fall).

der beiden Punkte. Diese Gewichtung der d Parameter des Algorithmus über die Korrelationsparameter $\theta_k > 0$ mit $1 \leq k \leq d$ findet statt um ihren unterschiedlichen Einflüssen Rechnung zu tragen[4]. Mit dieser Distanz wird die Korrelation der Abweichung vom Regressionsmodell angenommen zu

$$\mathrm{corr}\left[\epsilon(x), \epsilon\left(x^{(i)}\right)\right] = e^{-\delta\left(x, x^{(i)}\right)}.$$

Die hier verwendete gaußsche Korrelation ist nur eine von vielen möglichen Korrelationsfunktionen im allgemeinen Fall. Sie wurde bereits von Sacks u. a. [100] als flexibel genug für die meisten Fälle beschrieben. Die Verwendung unterschiedlicher Korrelationsparameter in jeder Dimension ist eine weitere Konkretisierung des allgemeinen Falls. Abbildung 7.2 zeigt den Einfluss der Korrelationsparameterwahl auf die angenommene Korrelation.

Die unbekannten Parameter dieses Gaußprozesses sind die Parameter der Korrelationsfunktion $\theta_1, \ldots, \theta_d$ und die Prozessvarianz σ^2. Für sie lassen sich, wie in Sacks u. a. [100] beschrieben, Maximumlikelihoodschätzer angegeben. Jones u. a. [54] enthält eine Beschreibung für den analytisch leichter zu betrachtenden Fall mit einer Regression vom Grad 0. Dabei besteht der Trend in der Schätzung des Mittelwerts. Letzteres wird häufig in der klassischen DACE-Literatur verwendet, Welch

[4] Um die unterschiedlichen Größenordnungen der Parameter auszugleichen, werden zuvor alle Dimensionen auf das Einheitsintervall abgebildet.

u. a. [117] sprechen von einem kleinen Vorteil bei Verwendung eines komplexeren linearen Regressionsterms. Hinter den klassischen DACE-Anwendungen steht häufig ein physikalischer Prozess. Dessen Kenntnis ermöglicht es zumeist die Faktoren soweit einzuschränken, dass keine starken globalen Trends zu vermuten sind. Ein Anwendungsszenario der SPO ist die Verwendung auf vollkommen unbekannten Algorithmen. Selten lassen sich hier fundierte Annahmen über die Begrenzung von $\mathcal{R}_\mathrm{A}$ machen, sodass diese globalen Trends hier durchaus beobachtet werden können und die Verwendung eines Regressionsmodells vom Grad 2 motivieren.

Sind die Regressionskoeffizienten β und die Korrelationsparameter θ ermittelt, so kann mit ihrer Hilfe das Verhalten des Algorithmus an unbekannten Punkten geschätzt werden. Der durch die Regression beschriebene Anteil der Approximation beträgt dabei

$$\hat{y}_\beta(x) = \sum_j \beta_j f_j(x)$$

und kann getrennt berechnet werden. Damit ergibt sich der Schätzer für einen unbekannten Punkt x zu

$$\hat{y}(x) = \hat{y}_\beta(x) + \mathbf{r}^T(x) R^{-1}(\mathbf{y} - \mathbf{1}\hat{y}_\beta(x)).$$

Dabei ist $\mathbf{r}$ ein Vektor der Länge n, dessen Komponenten

$$r_i(x) = \mathrm{corr}\left[\epsilon(x), \epsilon\left(x^{(i)}\right)\right]$$

die Korrelationen von x mit den bereits bekannten Designpunkten $x^{(i)}$ beschreiben. Die $n \times n$-Korrelationsmatrix R berechnet sich aus den Designpunkten zu

$$R_{i,j} = \mathrm{corr}\left[\epsilon\left(x^{(i)}\right), \epsilon\left(x^{(j)}\right)\right].$$

Die an den bekannten Punkten $x^{(i)}$ erzielten Resultate $y(x^{(i)})$ bilden den Vektor $\mathbf{y}$, $\mathbf{1}$ ist ein Vektor mit Einsen der Länge n.

Neben einer Schätzung für die unbekannten Punkte, bietet das Modell auch die Möglichkeit, den hierbei gemachten, mittleren quadratischen Fehler $s^2(x)$ abzuschätzen (Herleitung und Berechnung finden sich bei Sacks u. a. [100] und Lophaven u. a. [72]). Dieses findet sowohl bei der Validierung als auch bei der Wahl neuer Designpunkte Verwendung. Es lässt sich zeigen, dass an allen bekannten Designpunkten $x^{(i)}$ gilt:

$$\hat{y}\left(x^{(i)}\right) = y\left(x^{(i)}\right) \quad \text{und} \quad s^2\left(x^{(i)}\right) = 0.$$

Die Abb. 7.3 zeigt am Beispiel des funktionalen Zusammenhangs

$$y(x) = 40\left(\tanh(x/4) - 0.6\right)^2 + 2\sin(x) \tag{7.1}$$

den Graphen des Modellschätzers $\hat{y}(x)$ und des dabei geschätzten Fehlers. Sämtliche Berechnungen wurden mit dem „DACE-MATLAB“ Paket von Lophaven, Nielsen und Søndergaard [72] durchgeführt.

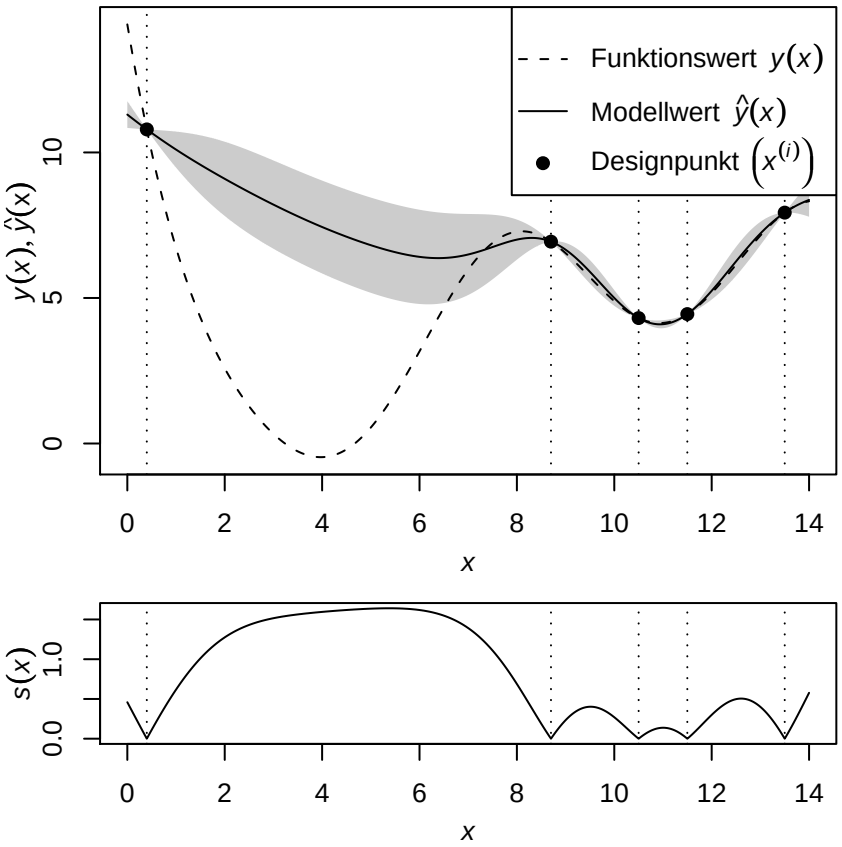


Abbildung 7.3.: Modellschätzung und Fehler. Oben dargestellt ist der Graph von $y(x)$ aus Gl. 7.1. Die Funktion wurden an den fünf markierten Punkten $x^{(i)}$ ausgewertet und auf dieser Grundlage ein Modell erzeugt. Der Verlauf des Modellschätzers $\hat{y}(x)$ ist ebenfalls dargestellt. Unten abgebildet ist die Schätzung des Modellfehlers. Diese wurde oben grau dargestellt, zu beiden Seiten des Modellschätzers aufgetragen, der Bereich bildet somit das 68,3%-Konfidenzintervall des Modellschätzers.

7.4. Modellvalidierung

Die Bewertung der Güte des Modells sollte zumindest einmal auf der Grundlage der initial ausgewerteten Designpunkte erfolgen. Jones u. a. [54] schlagen einige einfache Tests für die oben beschriebenen DACE-Modelle vor, die behilflich sind, offensichtliche Fehler zu vermeiden. Die Tests werden im Folgenden kurz beschrieben und später angewendet, um eine hinreichende Güte des Modells sicherzustellen, das der SPO zugrunde liegt. Bei der Interpretation wird berücksichtigt, dass das Verhalten des Algorithmus nichtdeterministisch ist und die Modellbildung auf fehlerbehafteten $y(x^{(i)})$ $(= \overline{Y}(x^{(i)}))$ beruht.

Die Tests basieren auf dem Prinzip der Leave-One-Out Kreuzvalidierung, dabei wird jeder der n Designpunkte, an denen das Verhalten des Algorithmus bekannt

ist, einmal aus der Menge der bekannten Designpunkte entfernt und das Verhalten an dieser Stelle durch das Modell vorhergesagt, dessen Parameter auf der Grundlage der restlichen Punkten geschätzt wurde. Im Folgenden sei $\hat{y}_{-i}(x)$ die Vorhersage von $y(x)$ in einem Modell, dessen Parameter ohne die Beobachtung $x^{(i)}$ geschätzt wurden. Der im Modell geschätzte Fehler dieser Vorhersage wird mit $s_{-i}(x)$ bezeichnet. Damit lassen sich nun drei Tests für die Modellgüte beschreiben:

1. Im Idealfall entspricht die Vorhersage aus der Kreuzvalidierung dem tatsächlich beobachteten Wert. Die $y(x^{(i)})$ für alle $x^{(i)}$ aufgetragen gegen $\hat{y}_{-i}(x^{(i)})$ liegen dann auf der 1. Winkelhalbierende.

2. Aus der Vorhersage und dem geschätzten Fehler ergibt sich das 99.7% Konfidenzintervall zu $\hat{y}_{-i}(x^{(i)}) \pm 3s_{-i}(x^{(i)})$. Der gegen die Vorhersage aufgetragene standardisierte Kreuzvalidierungsfehler

   $$\frac{y(x^{(i)}) - \hat{y}_{-i}(x^{(i)})}{s_{-i}(x^{(i)})}$$

   sollte sich damit möglichst im Intervall $[-3, +3]$ bewegen. Diese Art der Darstellung ermöglicht es, auch systematische Vorhersagefehler durch Korrelation der Werte zu entdecken.

3. Die Vorhersagefehler sollten normalverteilt sein. Die standardisierten Kreuzvalidierungsfehler aufgetragen in einem Q-Q-Plot gegen die Standardnormalverteilung sollten möglichst die 1. Winkelhalbierende ergeben.

Abbildung 8.1 zeigt diese drei Tests für eine SPO der GP-Parameter zur Evolution einer algorithmischen Chemie bzw. eines linearen Programms. Sollten die Tests negativ ausfallen, so wird in [54] vorgeschlagen, y vor der Modellbildung zu transformieren. Bei den vorgeschlagenen Transformationen handelt es sich um $\ln(y)$, $-1/y$ bzw. $-\ln(-y)$.

Das oben beschriebene Vorgehen bedarf zur Schätzung der Modellparameter und zu seiner Validierung (Leave-One-Out: +1) die experimentelle Auswertung an mindestens

$$\underbrace{1 + d + d(d-1)/2}_{\beta} + \underbrace{d}_{\theta} + 1 = 2 + 1\frac{1}{2}d + \frac{1}{2}d^2$$

Designpunkten. Dieser Bedarf muss bereits durch das initiale Design abgedeckt werden.

7.5. Sequenzielle Designs

Nachdem das initiale Design abgearbeitet ist, d. h. die zugehörigen Experimente ausgeführt wurden, sollen nun sequenziell weitere Designs erzeugt und ausgewertet werden. Die sequenziellen Designs verfolgen dabei zwei Ziele:

1. Weitere Experimente sollen so auf die bereits bekannten Designpunkte verteilt werden, dass die Wahrscheinlichkeit für die Wahl des besten Designpunkts maximiert wird.
2. Es soll ein zusätzlicher, vielversprechender Designpunkt gewählt und getestet werden, seine Bestimmung findet auf der Grundlage des Modells statt.

7.5.1. Wahl eines zusätzlichen Designpunkts

Die bisher beste Einstellung $x^{(b)}$ bestimmt sich aus den Repräsentanten $y(x^{(b)})$ der n bereits betrachteten Designpunkte:

$$x^{(b)} = x^{(i)}, \text{ mit } i = \arg\min_{1 \le i \le n} y(x^{(i)}) \quad .$$

Für die Wahl neuer Punkte bedienen wir uns der „erwarteten Verbesserung“ gegenüber $y(x^{(b)})$, in der von Schonlau u. a. [104] generalisierten Form. Dieses Kriterium ermöglicht es, sowohl den eigentlichen Vorhersagewert (lokale Suche), als auch die Unsicherheit der Vorhersage (globale Suche) zu berücksichtigen und unterschiedlich zu gewichten.

Die an einer ausgewerteten Einstellung x erzielte Verbesserung I gegenüber $x^{(b)}$ beträgt nach Potenzierung mit einem Exponenten g

$$\mathrm{I}^g(x) = \begin{cases} \left(y(x^{(b)}) - y(x)\right)^g & \text{wenn } y(x) < y(x^{(b)}), \\ 0 & \text{sonst.} \end{cases}$$

Von einem unbekannten Punkt x ist zunächst nur der Schätzwert $\hat{y}(x)$ und die Varianz $s^2(x)$ bekannt, mit ihnen lässt sich mit $g = 0$ die Wahrscheinlichkeit für eine Verbesserung angeben zu

$$\mathrm{E}\left(\mathrm{I}^0(x)\right) = \Phi\left(\frac{y(x^{(b)}) - \hat{y}(x)}{s(x)}\right).$$

Dabei ist Φ die kumulative Verteilungsfunktion von $N(0,1)$. Als generalisierte Form für $g = 1, 2, \ldots$ leiten Schonlau, Welch und Jones die erwartete Verbesserung her zu

$$\mathrm{E}\left(\mathrm{I}^g(x)\right) = s(x)^g \sum_{k=0}^{g} (-1)^k \binom{g}{k} u(x)^{g-k} T_k \tag{7.2}$$

mit

$$u(x) = \frac{y(x^{(b)}) - \hat{y}(x)}{s(x)}.$$

T_k ist rekursiv definiert als

$$T_k = -u(x)^{k-1}\phi\big(u(x)\big) + (k-1)T_{k-2}, \text{ mit } T_0 = \Phi\big(u(x)\big) \text{ und } T_1 = -\phi\big(u(x)\big),$$

bei ϕ handelt es sich um die Dichtefunktion von $N(0,1)$. Da im Modell $s(x)$ an bereits ausgewerteten Designpunkten den Wert null hat, nimmt hier die erwartete Verbesserung ihr Minimum an. Der Fehler der Schätzung steigt mit der Distanz zu bekannten Punkten. Je größer der Exponent g, desto bedeutender der Einfluss des Fehlers auf die zu erwartende Verbesserung. Die Suche wird globaler. Abbildung 7.4 zeigt den Einfluss unterschiedlicher Werte für g auf die ersten beiden Suchschritte in dem oben eingeführten Beispiel. In der linken Spalte befinden sich die Grafiken für $g = 1$ und in der rechten Spalte die für $g = 2$. Die erste Zeile zeigt die erwartete Verbesserung, wie sie sich für die beiden Parameter aus den in Abb. 7.3 dargestellten fünf Messpunkten ergibt. In beiden Fällen erfolgt eine Messung an den Stellen höchster Erwartung. Mit diesem zusätzlichen Messpunkt kann ein neues Modell ermittelt werden. Die beiden mittleren Grafiken zeigen den zusätzlichen Messpunkt und den Graphen des Schätzers. Die aus dem angepassten Modell hervorgehenden erwarteten Verbesserungen sind für beide Werte von g in der letzten Zeile abgebildet.

Im Folgenden wird SPO mit $g = 2$ für die Schätzung der erwarteten Verbesserung verwendet. Mit diesem Wert für g lässt sich Gl. 7.2 auch direkt interpretieren als:

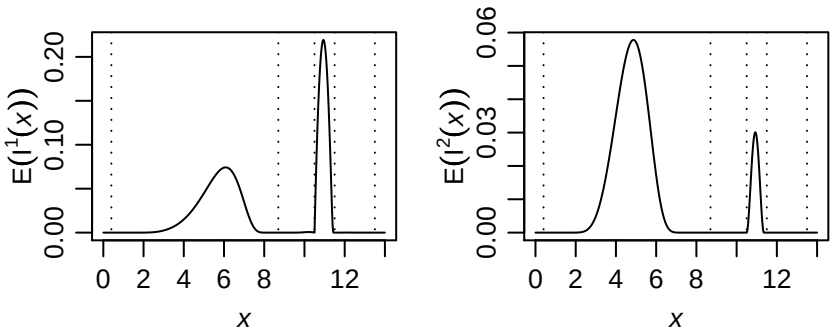
$$\mathrm{E}\big(\mathrm{I}^2(x)\big) = \mathrm{E}\big(\mathrm{I}^0(x)\big)^2 + \mathrm{Var}\big(\mathrm{I}^0(x)\big).$$

Zur Bestimmung eines Punktes mit einer hohen zu erwartenden Verbesserungswahrscheinlichkeit verwendet SPO ein großes LHS ($10^4 - 10^5$ Punkte) und schätzt an seinen Punkten x den Wert von $\mathrm{E}\big(\mathrm{I}^2(x)\big)$. Der Punkt mit dem größten Erwartungswert wird in das neue Design übernommen und n_0-mal experimentell ausgewertet.

Prinzipiell lässt sich anstelle dieses rein sequenziellen Ansatzes auch ein Ansatz mit mehr als einem neuen Designpunkt pro Runde verfolgen. Schonlau u. a. [104] macht für diesen Fall einen Vorschlag für eine entsprechend angepasste Schätzung der erwarteten Verbesserung an weiteren Designpunkten. Die Anpassung berücksichtigt, dass an den vorherigen hinzugenommenen Designpunkten der geschätzte Fehler der Vorhersage Null wird und somit auch die Schätzung über die erwartete Verbesserung in seiner Umgebung abnimmt.

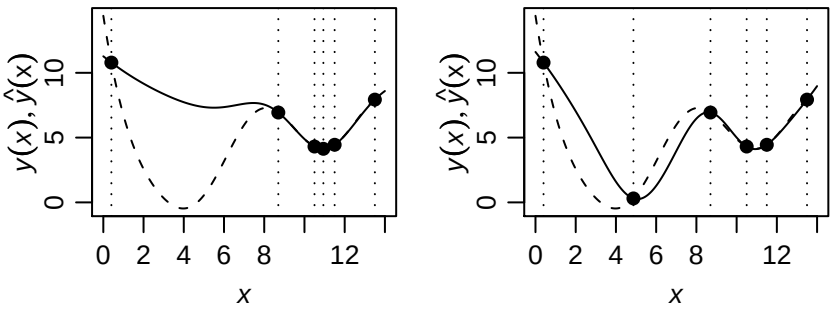
7.5.2. OCBA basierte Budgetverteilung

Neben der oben beschriebenen Bestimmung eines neuen, möglichst vielversprechenden Designpunkts dient das sequenzielle Design auch der Verteilung weiterer Experimente auf die bekannten Designpunkte. Dabei werden zwei gleichläufige Ziele verfolgt: (i) Die Wahrscheinlichkeit für die korrekte abschließende Bestimmung des besten Designpunkts soll maximiert werden. Dieses ist das vorrangige Ziel. (ii) Die Konfidenz über die geschätzte Güte der Designpunkte soll vor allem in dem Bereich der ROI erhöht werden, die das Potenzial besitzt, den besten Designpunkt zu stellen. Damit soll auch die Aussagegenauigkeit des Modells in diesen Bereichen erhöht und die Bestimmung neuer Designpunkte verbessert werden.



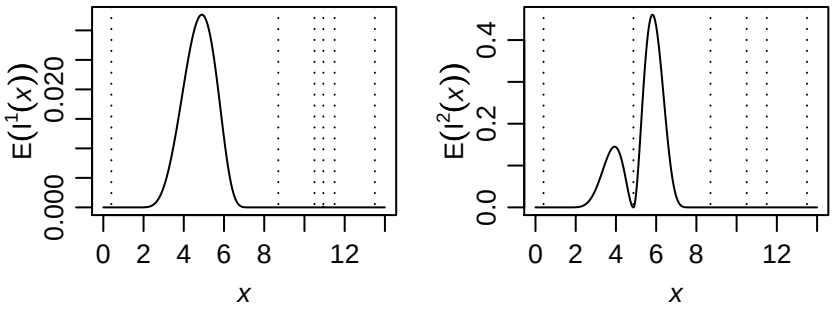
(a) $\mathrm{E}\big(\mathrm{I}^1(x)\big)$, $t = 0, g = 1$

(b) $\mathrm{E}\big(\mathrm{I}^2(x)\big)$, $t = 0, g = 2$



(c) $\hat{y}(x)$, $t = 1, g = 1$

(d) $\hat{y}(x)$, $t = 1, g = 2$



(e) $\mathrm{E}\big(\mathrm{I}^1(x)\big)$, $t = 1, g = 1$

(f) $\mathrm{E}\big(\mathrm{I}^2(x)\big)$, $t = 1, g = 2$

Abbildung 7.4.: Einfluss des Exponenten g auf die erwartete Verbesserung. Zu sehen ist zunächst die erwartete Verbesserung aufgrund der bekannten Punkte (Zeitpunkt $t = 0$). Der Punkt mit dem höchsten Erwartungswert wird hinzugenommen und ausgewertet. Mit dem zusätzlichen Punkt kann ein neues Modell berechnet werden. Die erwartete Verbesserungswahrscheinlichkeit in diesem Modell ist in der unteren Zeile dargestellt. Links dargestellt sind Erwartung und Wahl für $g = 1$, rechts abgebildet ist die Wahl für $g = 2$, hier erzielen zunächst unbekannte Bereiche einen hohen Erwartungswert, die Suche ist globaler.

Um das vorrangige Ziel zu erreichen, muss ggf. die Konfidenz über die Güte anderer bereits betrachteter Designpunkte erhöht werden. Es wird eine Variante der „optimal computing budget allocation“ (OCBA) von Chen u. a. [30] verwendet, um die Verteilung weiterer Experimente auf die bereits bekannten Designpunkte zu bestimmen.

OCBA gehört zur Klasse der „Ranking and Selection“-Prozeduren (R&S), siehe Bechhofer u. a. [20] für Selektionsprozeduren im Allgemeinen und Goldsman und Nelson [49] für einen Überblick über R&S basierte Ansätze im Speziellen. Die Rang basierten Selektionsverfahren machen sich zunutze, dass während die Schätzer mit $O(1/\sqrt{n})$ konvergieren[5], ihr Rang zumeist mit exponentieller Geschwindigkeit konvergiert. Des Weiteren handelt es sich bei der OCBA um einen sequenziellen Ansatz der, anders als z. B. zweistufige Ansätze, in jedem Schritt der SPO direkt zur Anwendung kommen kann.

Den meisten Selektionsverfahren gemein ist, dass sie entweder versuchen, eine vorgegebene Mindestwahrscheinlichkeit $P(\mathrm{CS})$[6] für die richtige Wahl der besten Alternative zu erreichen oder, dass sie mit einer Budgetvorgabe arbeiten, unter der sie versuchen, $P(\mathrm{CS})$ zu maximieren. Eine Übersicht und Diskussion von Verfahren, die mit Vorgabe von $P(\mathrm{CS})$ arbeiten, findet sich bei Kim und Nelson [60]. Durch die Hinzunahme eines weiteren Punktes in jedem Schritt, der beliebig nah an dem bisher besten Designpunkt liegen kann, ist es nicht ohne weiteres möglich die Terminierung in endlicher Zeit über das Erreichen der Mindestwahrscheinlichkeit, sicherzustellen. Die hier verwendete OCBA arbeitet mit einer Budgetvorgabe. Durch das Budget kann zudem sichergestellt werden, dass für zu vergleichende Systeme der gleiche Optimierungsaufwand betrieben wird.

Die Wahrscheinlichkeit $P(\mathrm{CS})$ bezieht sich hier auf die Wahl des Designs $x^{(b)}$ mit dem zuletzt besten beobachteten Wert, N_i ist die Anzahl der für den Designpunkt $x^{(i)}$ durchgeführten Experimente. Ist T das zur Verfügung stehende Budget, für die Gesamtzahl der Experimente, dann verfolgt OCBA das Ziel, dieses so auf die Designpunkte $x^{(i)}, i = 1, \ldots, k$ zu verteilen, das $P(\mathrm{CS})$ maximal ist:

$$\max_{N_1,\ldots,N_k} P(\mathrm{CS}), \text{ mit } \sum_{i=1}^{k} N_i = T \text{ und } N_i \in \mathbb{N}^+.$$

Basierend auf einem bayes'schen Modell approximieren Chen u. a. [30] die untere Schranke für $P(\mathrm{CS})$ durch

$$P(CS) \geq 1 - \sum_{i=1, i\neq b}^{k} P\left[\overline{Y}(x^{(b)}) > \overline{Y}(x^{(i)})\right]$$

und zeigen, dass diese asymptotisch maximiert wird, wenn für die Verteilung der

[5] n ist die Anzahl der Wiederholungen, die in die Schätzung eingehen

[6] CS steht hier für den englischen Term „correct selection“.

Experimente gilt

$$\frac{N_i}{N_j} = \left(\frac{\sigma_i / \left(\overline{Y}(x^{(i)}) - \overline{Y}(x^{(b)}) \right)}{\sigma_j / \left(\overline{Y}(x^{(j)}) - \overline{Y}(x^{(b)}) \right)} \right)^2 \text{ und } N_b = \sigma_b \sqrt{\sum_{i=1, i \neq b}^{k} \frac{N_i^2}{\sigma_i^2}} \quad .$$

Dabei ist σ_i die Standardabweichung der Resultate am Designpunkt $x^{(i)}$. Da sie unbekannt ist, wird sie aus den vergangenen Experimenten geschätzt. Die Herleitung geschieht unter der Annahme, dass die Experimente normal verteilt sind. Chen u. a. zeigen aber, dass ihr Verfahren robust bzgl. dieser Annahme ist und erzielen bei Anwendung auf normal- wie nicht normal verteilten Daten vergleichbar gute Resultate in Relation zu anderen Selektionsverfahren.

Chen u. a. [31] erweitern die Herleitung der Budgetverteilung um die Möglichkeit, den kleinsten noch signifikanten Unterschied $d^\star$ zu definieren. Dieser Unterschied wird in der Literatur auch als „indifference-zone“ bezeichnet. Durch die Definition eines kleinsten signifikanten Unterschieds wird verhindert, dass weitere Experimente auf die genauere Bestimmung der Rangordnung von Designpunkten verwendet werden, deren Unterschied als nicht signifikant erachtet wird. Ist der kleinste signifikante Unterschied nicht bekannt, so verwendet das Verfahren den Unterschied der zwei besten Designpunkte. Chen u. a. [31] zeigen experimentell, dass OCBA und OCBAIZ dann eine vergleichbare Wahrscheinlichkeit $P(\mathrm{CS})$ für die richtige Selektion des besten Designs erzielen.

Für die Bestimmung der Anzahl weiterer Experimente auf die bekannten Designpunkte wird hier die von Chen u. a. [31], Anhang A gegebene Implementierung von OCBAIZ verwendet. Falls vorab sinnvolle Annahmen für $d^\star$ gemacht werden können, wird dies angegeben.

Die Verteilung des Restbudgets T auf die Designpunkte geschieht sequenziell und parallel zur Bestimmung eines weiteren Designpunkts. Dabei wird in jeder Runde l ein Teil Δ_l des restlichen Budgets auf die Designpunkte verteilt. Bei der Verteilung weiterer Teilbudgets gehen die Ergebnisse aller vorherigen Experimente mit ein. Dieses sequenzielle Vorgehen, anstelle der einmaligen Verteilung des Budgets, ermöglicht die frühzeitige Nutzung der gewonnenen Erkenntnisse.

Für die Wahl von Δ_l geben Chen u. a. [30] einen Wert zwischen 5 Experimenten und 10% der betrachteten Designpunkte an, in [31] wird dieser Wert dynamisch aus dem restlichen Budget zu $\Delta_l = \min\left(T, \max\left(k, \lceil T/2 \rceil\right)\right)$ bestimmt. Für die SPO gilt es zum einen die Budgetverteilung nicht zu defensiv vorzunehmen, da eine Erhöhung der Experimentanzahl nicht nur die Bestimmung des Rangs sondern auch die eigentliche Vorhersage verbessert und damit der Bestimmung neuer Punkte zugute kommt. Ein zu großzügiger Wert für Δ_l andererseits reduziert die Anzahl der Runden und damit die zusätzlichen Designpunkte. Im Folgenden findet

$$\Delta_l = \min(T, k + (l - 1)),$$

Verwendung, k ist die Anzahl initialer Designpunkte. Damit entspricht die Anzahl zusätzlicher Experimente der Anzahl bereits betrachteter Designpunkte.

1. Initial: Führe für jeden der k initialen Designpunkte n_0 Experimente durch. Setze $l = 0$, $N_1^l = N_2^l = \ldots = N_k^l = n_0$ und $T = T - k \cdot n_0$.
2. Sequenziell:
   a) Setze $l = l + 1$, berechne nach OCBAIZ die Anzahl $N_1^l, N_2^l, \ldots N_{k+l-1}^l$ der Experimente für die die Approximation von $P(\text{CS})$ nach einer Erhöhung der Gesamtzahl um $\Delta_l = \min(T, k + l - 1)$ maximal ist und führe für jeden Designpunkt $x^{(i)}$ die fehlende Zahl von Experimenten durch. Setze $T = T - \Delta_l$.
   b) Falls $T \geq n_0$: bestimme einen weiteren Designpunkt $x^{(x+l)}$ und führe n_0 Wiederholungen durch, setze $T = T - n_0$.
3. Falls $T > 0$ gehe zu Schritt 2.
4. Gib das Design $x^{(b)}$ zurück, sodass $\overline{Y}(x^{(b)})$ minimal ist.

Abbildung 7.5.: Ablauf eines um OCBAIZ erweiterten SPO Laufs. Dabei ist k die Anzahl der initialen Designpunkte, n_0 die initiale Anzahl Wiederholungen, N_i^l die Gesamtzahl der für den Designpunkt $x^{(i)}$ am Ende des sequenziellen Schritts l durchgeführten Wiederholungen. T beinhaltet zu Beginn das Gesamtbudget. Die Bestimmung eines weiteren Designpunkts in 2.b) findet ohne die zusätzlichen Experimente in 2.a) statt und kann somit parallel geschehen.

Abbildung 7.5 stellt noch einmal den Ablauf der hier verwendeten Kombination von OCBAIZ und SPO dar. Es bleibt abschließend darauf hinzuweisen, dass die OCBA-Verfahren für gewöhnlich mit einem Design fester Größe arbeiten. Durch die Vergrößerung des Designs in den sequenziellen Schritten kann sich die Maximierung von $P(\text{CS})$ nur auf den aktuellen Zustand beziehen und nicht auf die Menge der letztendlich betrachteten Designpunkte. Es ist daher ratsam, nach dem letzten Schritt einen Blick auf die Anzahl der Experimente für den besten Designpunkt und die Verteilung seiner Ergebnisse zu werfen.

7.6. Effekte und Interaktionen

Neben der Suche nach einer guten Belegung der Parameter lassen sich die gewonnenen Daten und das erzeugte Modell auch für weitere Einsichten über den generellen Einfluss dieser Parameter auf die Reaktion des Algorithmus nutzen.

Die Reaktion des Algorithmus, beschrieben durch $y(x)$, lässt sich in mehrere Komponenten unterteilen: die durchschnittliche Reaktion, Reaktionsänderungen, die auf Veränderung eines Faktors zurückzuführen sind (Haupteffekte), und Ände-

rungen, die durch Interaktion mehrerer (im Folgenden zweier) Faktoren erzeugt werden. Der Durchschnitt von $y(x)$ in der Region $\mathcal{R}_\mathrm{A}$ beträgt

$$\mu_0 = \int_{\mathcal{R}_\mathrm{A}} y(x) \prod_{h=1}^{d} \mathrm{d}x_h.$$

Der durch die Festlegung von Parameter i auf x_i verursachte Effekt beträgt, als Abweichung vom Durchschnitt und integriert über alle anderen Parameter,

$$\mu_i(x_i) = \int_{\mathcal{R}_\mathrm{A}} y(x) \prod_{h\neq i}^{d} \mathrm{d}x_h - \mu_0.$$

Entsprechend lässt sich der Effekt, der durch die Interaktion zweier Parameter i und j an der Stelle x_i und x_j verursacht wird, berechnen durch

$$\mu_{i,j}(x_i, x_j) = \int_{\mathcal{R}_\mathrm{A}} y(x) \prod_{h\neq i,j}^{d} \mathrm{d}x_h - \mu_0 - \mu_i(x_i) - \mu_j(x_j).$$

Um die Effekte zu schätzen, wird $y(x)$ durch den Schätzer des Modells $\hat{y}(x)$ ersetzt. Des Weiteren wird die Integralbildung diskretisiert. Hierfür wird zunächst ein großes Design (hier 2000 Designpunkte) mittels LHS bestimmt. Der durchschnittliche Wert des Schätzers an diesen Punkten bestimmt $\hat{\mu}_0$, um den Effekt des Parameters p_i an der Stelle x_i zu bestimmen, wird dieser Parameter nun bei allen Designpunkten auf x_i gesetzt. Das so entstandene neue Design wird im Modell ausgewertet, die Abweichung des mittleren Schätzwerts dieses Designs von $\hat{\mu}_0$ ergibt den Effekt $\hat{\mu}_i(x_i)$. Entsprecht werden für die Schätzung der Interaktion zwei Parameter fixiert.

Andere hier nicht weiter diskutierte Möglichkeiten zur statistischen Aufbereitung der gewonnenen Daten bestehen etwa in der Verwendung von Klassifikations- und Regressionsbäumen.

8. Experimentdesign

In diesem Kapitel werden die Problemstellungen beschrieben und Parametrisierungen der verwendeten Algorithmen mittels der sequenziellen Parameteroptimierung aus Kap. 7 bestimmt. Stellvertretend für die Systemvarianten zur Evolution einer algorithmischen Chemie findet die Parameteroptimierung unter Verwendung der Totalsynthese statt, da diese Variante schneller ausgewertet werden kann. Auch die lineare GP-Variante des Systems wird bzgl. ihrer Parametrisierung optimiert, da sie später als Referenzsystem herangezogen wird. Als Probleme werden die symbolische Regression einer trigonometrischen Funktion, die Evolution einer booleschen Funktion und ein Klassifikationsproblem aus der Medizin betrachtet.

Das System zur genetischen Programmierung algorithmischer Chemien wird im folgenden AC-System genannt, die Variante für die genetische Programmierung linearer Programme als LGP-System bezeichnet. Es sei an dieser Stelle noch einmal darauf hingewiesen, dass es sich dabei um größtenteils identische Systeme handelt, da versucht wurde, die Unterschiede auf das notwendige Minimum zu reduzieren. Nachdem in diesem Kapitel die Parameter der Systeme mittels SPO eingestellt wurden, findet im folgenden Kapitel der Vergleich der Systeme statt. Dort wird sich zeigen, wie sich die unter Verwendung der Totalsynthese gewonnenen Parameter für die Evolution einer algorithmischen Chemie ohne Totalsynthese eignen und wie der Evolutionsverlauf von der Anzahl ausgeführter Reaktionen abhängt.

Tabelle 8.1 zeigt die im Rahmen der Optimierung betrachteten Parameter und die Intervallgrenzen („region of interest“), in denen die Optimierung stattfindet. Betrachtet werden dabei die Anzahl der Eltern, der Selektionsdruck, die Mutations- und Rekombinationsrate sowie die Anzahl der im LGP-System vorhandenen Re-

Tabelle 8.1.: Die Grenzen der Parameteroptimierung („region of interest“) und die besten gefundenen Einstellungen für die GP-Systeme zur Evolution einer algorithmischen Chemie bzw. eines linearen Programms für die unterschiedlichen Problemstellungen. Bei den Problemstellungen handelt es sich um das Parity- (Par4), das Thyroid- (Thy) und das Sinusproblem (Sin). Die beiden markierten Werte (*) stammen aus einer späteren manuellen Anpassung (siehe Abschnitt 9.3).

	„region of interest“			AC-System			LGP-System		
Parameter	min	max	$\mathbb{N}/\mathbb{R}$	Par4	Thy	Sin	Par4	Thy	Sin
Eltern	100	4000	$\mathbb{N}$	150*	156	133	165	167	168
Selektionsdruck	1.01	7	$\mathbb{R}$	6.8*	6.92	6.64	2.93	3.88	5.37
Mutationsrate (in %)	0	4	$\mathbb{R}$	2.27	3.68	3.14	0.54	3.57	0.75
Rekomb.rate (in %)	0	30,100	$\mathbb{R}$	0.12	0.31	2.7	43.8	42.2	28.5
Moleküle/Register	15	30	$\mathbb{N}$	24	21	27	15	19	26

Tabelle 8.2.: Potenzreihe der Sinusfunktion und ihre Fitness auf der Testmenge in Abhängigkeit der Anzahl verwendeter Glieder. Die Anzahl wird durch die Obergrenze der Summe $n_{\max}$ bestimmt.

$n_{\max}$	Potenzreihe	Fitness
0	x	1.918
1	$x - \frac{x^3}{3!}$	0.426
2	$x - \frac{x^3}{3!} + \frac{x^5}{5!}$	2.014×10^{-2}
3	$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$	3.264×10^{-4}
4	$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$	2.303×10^{-6}

gister bzw. im AC-System vorhandenen zustandsveränderlichen Moleküle. Die Parameter und ihre Intervalle werden für beide Systemvarianten und für alle drei betrachteten Problemstellungen nahezu identisch gewählt. Die Ausnahme bildet die Rekombinationsrate. In einem ersten Screening wurde festgestellt, dass hohe Rekombinationsraten bei der Evolution algorithmischer Chemien nur zu sehr schlechten Resultaten führen. Die Obergrenze für den Anteil der durch Rekombination erzeugten Nachkommen wird auf 30% für das AC-System und auf 100% für das LGP-System festgelegt.

Die Tabelle 8.1 enthält bereits die im Rahmen der Parameteroptimierung ermittelten Einstellungen. Die folgenden Unterabschnitte gehen auf diese und auf problemspezifische Aspekte des Optimierungsprozesses genauer ein. Da sich der Prozess für jede Problemstellung wiederholt, wird auf diesen bei der symbolischen Regression der Sinusfunktion genauer eingegangen, für die anderen Problemstellungen finden sich Detailinformationen aus dem Optimierungsprozess im Anhang A.

8.1. Sinus: Symbolische Regression

Bei diesem Problem soll mittels symbolischer Regression eine Funktion gefunden werden, welche die Sinusfunktion im Bereich $[-\pi, \pi]$ möglichst gut approximiert. Dabei dürfen keine trigonometrischen Funktionen sondern ausschließlich die Addition, Subtraktion, Multiplikation und Division verwendet werden. Die Potenzreihe

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}. \tag{8.1}$$

stellt eine mögliche Lösung dar. Die Güte einer evolvierten Lösung kann mit der Güte der Potenzreihe bei unterschiedlicher Anzahl von Gliedern verglichen werden. Dafür wird die ursprüngliche Reihe nach oben durch $n_{\max}$ in der Anzahl ihrer Gieder beschränkt. Tabelle 8.2 zeigt die Potenzreihe und ihre Fitness auf der Testmenge für unterschiedliche Werte von $n_{\max}$.

Tabelle 8.3 gibt einen Überblick über weitere problemspezifische Einstellungen. Der Ausdruck muss aus den Operationen Addition, Subtraktion, Multiplikation,

Tabelle 8.3.: Problemdesign der Suche nach einem symbolischen Ausdruck für die Sinusfunktion und spezifische Einstellungen für die SPO.

Parameter	Einstellung
Problemdesign	
Funktionsmenge	$+, -, *, /$
Terminale	x, 5 konstante Werte
Fitnessfunktion	$\frac{1}{n}\sum_{i=1}^{n}(\sin(x_i) - I(x_i))^2$, MSE
Trainingsgesamtmenge	1000 Werte gleichverteilt aus $[-\pi, \pi]$
Trainingsmenge	100 Fallbeispiele per SSS
Terminierung	300 000 Evaluationen
initale Länge	25 Instruktionen/Reaktionen
maximale Länge	500 Instruktionen/Reaktionen
Validierungsmenge	400 Werte gleichverteilt aus $[-\pi, \pi]$
Testmenge	400 Werte gleichverteilt aus $[-\pi, \pi]$
Sequenzielle Parameteroptimierung	
Budget	1000 Experimente
Indifference-Zone	–
Anmerkung	Fitness nach oben auf 0.2 begrenzt

Division sowie der Variablen x für die Eingabe und fünf Konstanten gebildet werden. Die Werte der Konstanten werden im Individuum evolviert, sind aber während der Ausführung unveränderlich. Die Individuen werden anhand von 100 Fallbeispielen über den mittleren quadratischen Fehler (MSE) bewertet. Diese 100 Fallbeispiele werden für alle Individuen einer Generation zufällig aus 1000 gleichverteilten Punkten im Bereich $[-\pi, \pi]$ gewählt. Der Größe der Trainingsmenge kommt besondere Bedeutung zu. Sie wird im Abschnitt 8.2.1 genauer betrachtet. Für die Evolution einer Lösung stehen den Systemen je 300 000 Evaluationen zur Verfügung. Die initiale Länge der Individuen beträgt 25, die maximale Länge ist auf 500 Instruktionen bzw. Reaktionen beschränkt. Test- und Validierungsmenge bestehen aus jeweils 400 Fallbeispielen. Die Auswertung der Individuen auf diesen Mengen beeinflusst nicht den Evolutionsverlauf, bestimmt aber das beste Individuum, welches vom System zurückgegeben wird.

Tabelle 8.3 listet des Weiteren Einstellungen für die sequenzielle Parameteroptimierung auf. Für die Optimierung jedes Systems stehen 1000 Experimente zur Verfügung, Annahmen über eine Indifference-Zone wurden nicht gemacht. Die Bewertung eines einzelnen Laufs entspricht der Fitness des Individuums, das bei Erreichen des Terminierungskriteriums als Bestes aus der Population selektiert wird. Die Güte vergangener Individuen wird weder vermerkt, noch wird eine elitäre Strategie verfolgt. Die Fitness dieses Individuums kann, aufgrund des mittleren quadratischen Fehlers als Fitnessmaß und der nichtdeterministischen Auswertung bei einzelnen Fallbeispielen zu großen Ausreißern führen. Dieses lässt das Indivi-

duum und im Rahmen der SPO die gesamte Einstellung sehr schlecht erscheinen. Die fehlende Robustheit des Mittelwerts gegenüber Ausreißern führte hier dazu, dass keine vernünftige Optimierung mehr möglich war. Die Problemstellung weist zudem ein sehr dominantes lokales Optimum auf. Die Güte des lokalen Optimums liegt im Bereich zwischen 0.18 und 0.20, es handelt sich dabei um triviale Lösungen, die zumeist aus einer Geraden bestehen. So erzeugt z. B. die Gerade durch den Ursprung mit Steigung 0.3 einen mittleren quadratischen Fehler von 0.186 auf der Testmenge. Nahezu alle Einstellungen des initialen Designs beim AC-System haben ihren Median im lokalen Optimum. Daher stellte auch die Verwendung des Median keine Alternative dar, um die Experimente einer Einstellung zu aggregieren. Um dieses Problem zu beheben, wurden die Ausreißer auf einen Wert von 0.2 beschränkt. Die Güte der Einstellung berechnet sich anschließend aus der mittleren Fitness der zugehörigen Experimente.

Die Tabellen 8.4 (a&b) geben für beide Systeme die 10 besten während der Parameteroptimierung betrachteten Einstellungen wieder. Zeile eins entspricht dem Eintrag in Tabelle 8.1. Die ersten Spalten beschreiben die Parameter des Designpunkts gefolgt von Mittelwert und Varianz aller in ihm durchgeführten Experimente. Die hinteren Spalten geben die Anzahl der Experimente an, die in den einzelnen Schritten der SPO durchgeführt wurden. Beim Schritt 0 handelt es sich um das initiale Design. Da der Suchraum 5-dimensional ist, werden anfänglich $5 \times 11 = 55$ Designpunkte mithilfe eines raumfüllenden LHS betrachtet. Für jeden dieser Punkte werden 10 Experimente durchgeführt. Die weiteren Schritte zeigen zum einen die Aufteilung weitere Experimente mittels OCBA auf bereits bekannte Einstellungen, zum anderen wird in jedem Schritt ein weiterer Designpunkt auf der Grundlage des Modells bestimmt, dem dann erstmalig Experimente zugeordnet sind.

Der beste Designpunkt für das AC-System wird im fünften sequenziellen Schritt erstmals betrachtet. Aus den 70 Experimente der SPO für diese Einstellung errechnet sich eine durchschnittliche Fitness von 0.109. Für das LGP-System ist die beste gefundene Einstellung bereits unter den initial getesteten Einstellungen vorhanden, die SPO ermittelt im Schritt 4 noch eine vielversprechende Einstellung, die sich durch eine größere Population und eine höhere Rekombinationsrate auszeichnet. Die 114 Experimente, die im Laufe der SPO mit der besten gefundenen Einstellung für das LGP-System durchgeführt werden, ergeben eine durchschnittliche Fitness von 0.037. Dieser Wert liegt deutlich unter dem Wert, der bei der Evolution einer algorithmischen Chemie erreicht wird. Ein detaillierter Vergleich findet in Kap. 9 statt.

Abbildung 8.1 dient der in Kap. 7.4 beschriebenen Validierung der Modelle, die während der SPO erzeugt werden. Für beide Systeme ist eine Häufung der Einstellungen bei einer durchschnittlichen Fitness (in den Graphen mit y bezeichnet) im Bereich des oben beschriebenen lokalen Optimums zu erkennen. Die Einstellungen werden in den Modellen nur schlecht abgebildet und streuen um den Wert des lokalen Optimums. Dieses wird beim AC-System besonders deutlich. Abbildung 8.1 (b) zeigt, basierend auf allen während der SPO betrachteten Einstellungen für dieses

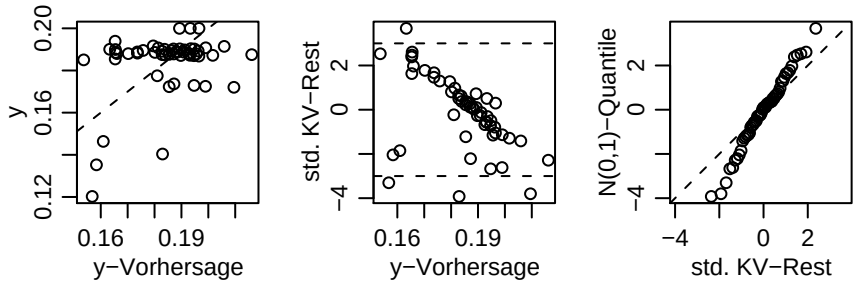
Tabelle 8.4.: Verlauf der sequenziellen Parameteroptimierung für das (a) AC-System bzw. (b) LGP-System. Dargestellt sind jeweils die 10 besten Einstellungen, die hiermit durchschnittlich erreichte Fitness, die Standardabweichung sowie die Anzahl der mit einer Einstellung durchgeführten Experimente in den einzelnen Runde der SPO.

(a) AC-System

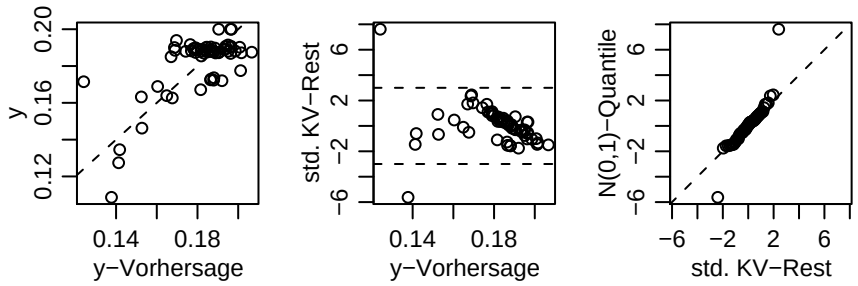
	Eltern	Sel.d.	Rekomb.	Mut.	Mol.	mean	stddev	0	1	2	3	4	5	6	7
1	133	6.64	0.027	0.0314	27	0.109	0.0872						10	60	0
2	119	6.5	0.0187	0.0302	30	0.127	0.0864			10	50	44	20	0	0
3	131	6.77	0.00495	0.0163	30	0.135	0.0788							10	45
4	467	6.77	0.0928	0.029	25	0.146	0.0747	10	30	0	0	4	39	0	0
5	370	5.37	0.0469	0.032	18	0.163	0.0612	10	21	0	0	10	0	0	0
6	230	6.8	0.055	0.0251	15	0.163	0.0586		10	23	0	0	0	0	0
7	725	5.09	0.075	0.0209	30	0.164	0.0506	10	0	33	0	0	0	0	0
8	433	3.99	0.0016	0.00595	18	0.167	0.0589	10	4	0	7	0	0	0	0
9	279	6.64	0.289	0.0397	30	0.169	0.0471				10	0	0	0	0
10	201	6.21	0.0134	0.0394	30	0.171	0.0481					10	0	0	0

(b) LGP-System

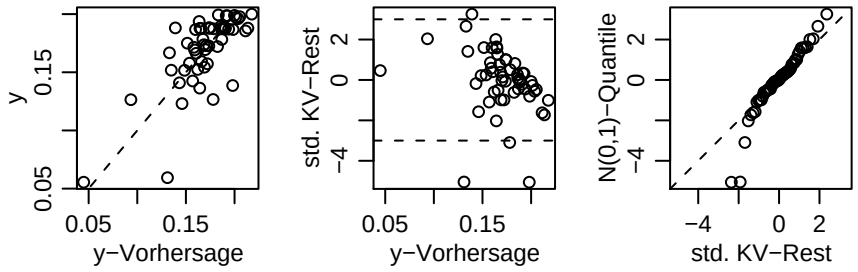
	Eltern	Sel.d.	Rekomb.	Mut.	Reg.	mean	stddev	0	1	2	3	4	5	6	7
1	168	5.37	0.285	0.00748	26	0.0373	0.069	10	30	15	26	22	10	1	0
2	283	6.37	0.602	0.00182	15	0.0491	0.0755					10	49	59	0
3	109	6.46	0.696	0.003	30	0.0613	0.0836		10	41	31	36	0	0	0
4	248	6.86	0.144	0.00516	15	0.0689	0.08						10	0	0
5	287	6.99	0.575	0.000413	17	0.0826	0.0833	10	25	0	0	0	0	0	0
6	139	6.96	0.938	0.00277	15	0.0855	0.0883							10	45
7	394	5.72	0.836	0.0212	22	0.123	0.0819	10	0	0	0	0	0	0	0
8	469	4.28	0.155	0.00518	16	0.126	0.0775	10	0	0	0	0	0	0	0
9	1490	3.69	0.0416	0.0293	16	0.127	0.0756	10	0	0	0	0	0	0	0
10	326	6.48	0.928	0.0000329	30	0.132	0.078			10	0	0	0	0	0



(a) AC-System, initiales Modell



(b) AC-System, finales Modell



(c) LGP-System, initiales Modell

Abbildung 8.1.: Graphen zur visuellen Validierung der im Rahmen der SPO erzeugten Modelle. Betrachtet wird die Optimierung der Systeme für das Sinusproblem. Zu sehen ist die Validierung der Modelle für das AC-System auf der Grundlage der Ergebnisse (a) des initialen raumfüllenden Designs, (b) am Ende der SPO und (c) auf der Grundlage des initialen Designs für das LGP-System.

System, dass im Laufe der SPO der vielversprechendere Bereich besser im Modell erfasst wird, wenngleich zwei Punkte in den Modellen der Kreuzvalidierung nur sehr schlecht rekonstruiert werden.

Die Abbildungen 8.2 und 8.3 zeigen die aus den Daten der SPO-Läufe gewonnenen Effekt- und Interaktionsplots für das AC-System und das LGP-System. Berechnet auf der Grundlage des Modells stellen die Effektplots auf der Diagonalen dar, wie hoch im Durchschnitt über den betrachteten Parameterraum die Effektänderung wäre, würde der betrachtete Parameter entsprechend festgelegt werden. Die Interaktionsplots zeigen zusätzliche Abweichungen vom Durchschnitt bei Änderung zweier Parameter.

Bei Betrachtung der Effektplots des AC-Systems werden drei Einflüsse besonders deutlich. Die ersten beiden betreffen die Anzahl der Eltern und den Selektionsdruck. Die Einstellungen profitieren im Durchschnitt von einer kleinen Populationsgröße und einem hohen Selektionsdruck. Die Kombination aus kleiner Population und hohem Selektionsdruck führt auch beim linearen GP-System im Schnitt zu besseren Ergebnissen. Aufgrund der vorherigen Beschränkung des Suchraums ist die Präferenz des AC-Systems für eine kleine Rekombinationsrate nur leicht zu erkennen.

Deutliche Interaktionen der Parameter existieren zwischen dem Selektionsdruck und der Mutationsrate bzw. der Populationsgröße. Die Auswirkung einer zu großen Mutationsrate kann durch einen höheren Selektionsdruck kompensiert werden. Während eine kleine Population zunächst einmal eine längere Evolutionsdauer ermöglicht, da pro Generation weniger der limitierten Evaluationen zur Auswertung der Individuen benötigt werden, reduziert ein hoher Selektionsdruck die Evolutionsdauer wieder zugunsten einer schnelleren Konvergenz. Der Interaktionsplot deutet darauf hin, dass das AC-System nur dann einen hohen Selektionsdruck bevorzugt, wenn eine angemessene Evolutionsdauer durch eine geringe Populationsgröße garantiert ist. Bei größeren Populationen ist ein geringer Selektionsdruck erfolgreicher.

Bei Betrachtung der Skalen von Parametereffekten und -interaktionen wird zunächst deutlich, dass der Einfluss der Parametrisierung im LGP-System ca. um den Faktor drei größer ist. Eine kleine Population und ein hoher Selektionsdruck werden auch vom LGP-System bevorzugt, im Unterschied zum AC-System hingegen ist die durchschnittlich optimale Rekombinationsrate deutlich höher. Das LGP-System bevorzugt eine geringere Mutationsrate. Bei hohen Mutationsraten weisen die Interaktionsplots auf bessere Ergebnisse mit einer größeren Population hin, die das Überleben von genug angemessen mutierten Individuen ermöglicht.

8.2. Parity: Boolesche Paritätsfunktion

In diesem Abschnitt werden beide Systeme für die Evolution der booleschen Paritätsfunktion parametrisiert. Das Parityproblem ist bezüglich der Eingabegröße beliebig skalierbar. Hier wird die Eingabe auf 4 Bits $(x_1, \ldots, x_4) \in \{0,1\}^4$ festgelegt.

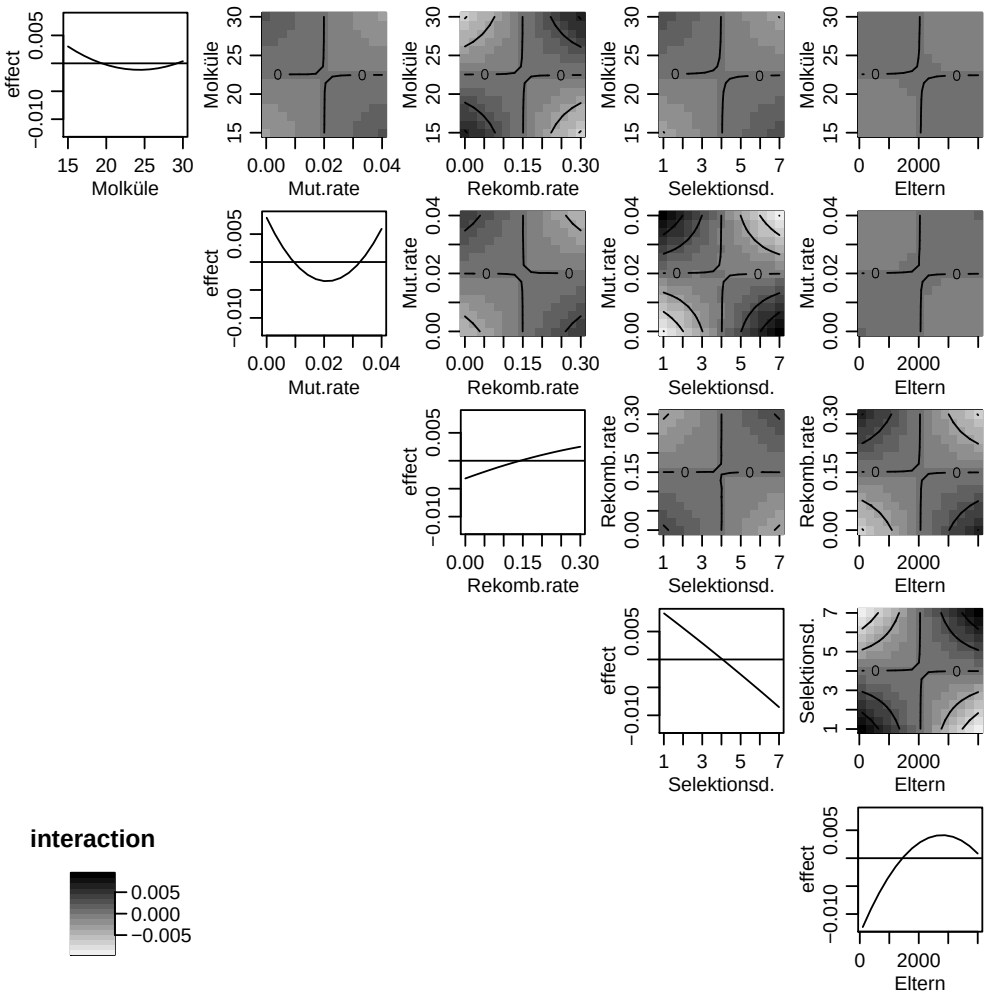


Abbildung 8.2.: Effekt- und Interaktionsplots für die Parameter des AC-Systems auf dem Sinusproblem. Die Effektplots auf der Diagonalen zeigen den Einfluss einzelner Parametereinstellungen auf die erzielte Güte im Durchschnitt aller verbleibenden Parameter. Kleinere Werte stellen eine Verbesserung dar. Beispielsweise sind eine geringe Größe der Elernpopulation, eine kleine Rekombinationsrate sowie eine hoher Selektionsdruck von Vorteil. Die Plots oberhalb der Diagonalen zeigen den Einfluss an, der sich aus der Interaktion der Parameter ergibt. Sollte sich z. B. entgegen den Empfehlungen der Effektplots für eine große Elternpopulation entschieden werden, so sollte der Selektionsdruck gering gewählt werden.

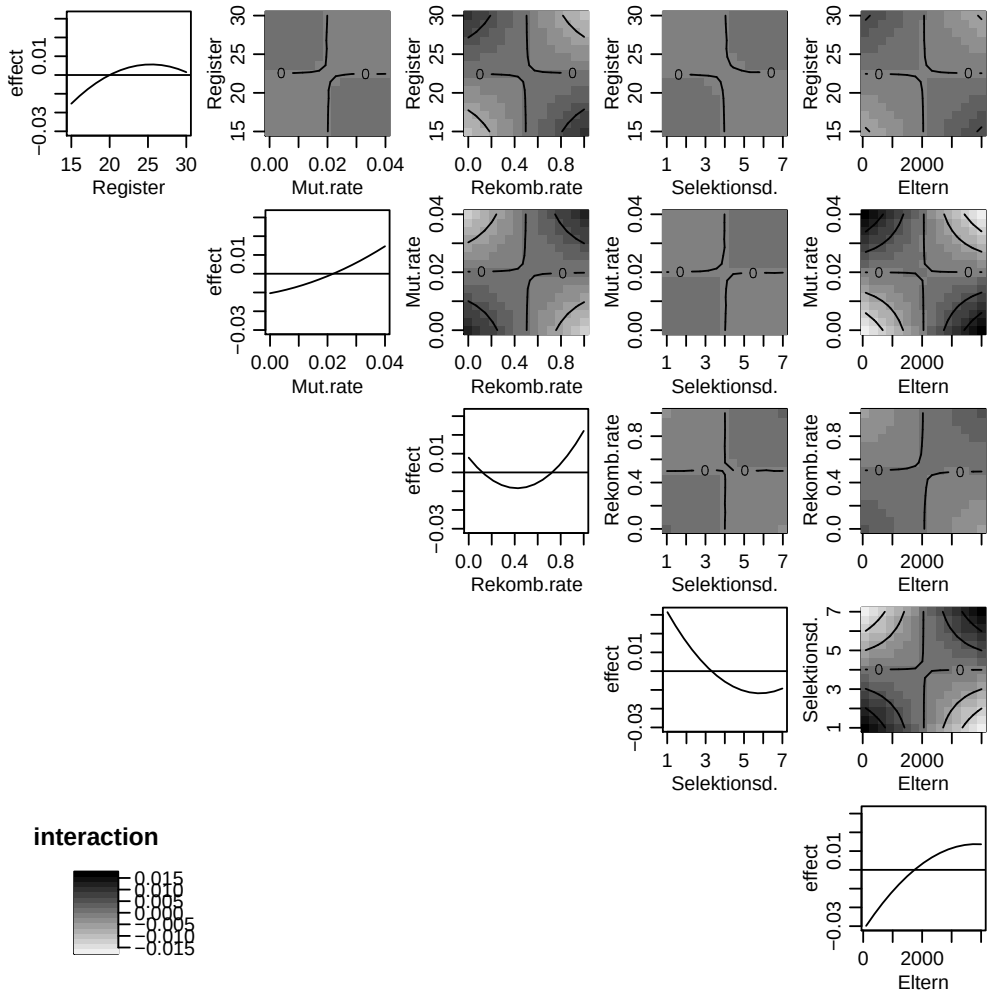


Abbildung 8.3.: Effekt- und Interaktionsplots für die Parameter des LGP-Systems auf dem Sinusproblem. Die Bildunterschrift zu Abb. 8.2 liefert eine Interpretationshilfe. Im Durchschnitt aller Parameter ist beispielsweise eine Rekombinationsrate von 40% ein guter Wert, die Mutationsrate sollte geringer gewählte werden, als beim AC-System.

Die Werte 0 und 1 entsprechen den Zuständen „false“ und „true“. Gesucht wird eine Funktion, die bei gerader Parität den Wert 1, bei ungerader Parität den Wert 0 zurückgibt:

$$p(x_1, \ldots, x_4) = 1 - \sum_{i=1}^{4} x_i \mod 2 \quad .$$

In der GP-Literatur ist dieses Problem als „Even-4-Parity“ bekannt. Verwendet werden dabei klassischerweise nur die Operationen AND, OR, NAND und NOR. Paritätsfunktionen werden unter anderem für die Fehlererkennung in übertragenen oder gespeicherten Daten verwendet, da mithilfe eines Paritätsbits jeder 1 Bit Fehler erkannt wird.

Die ersten Versuche, diese Paritätsfunktion mittels GP zu erzeugen, finden sich bereits bei Koza [62]. Koza untersucht die Fähigkeit zur Evolution boolescher Funktionen im Allgemeinen und zieht die Paritätsfunktion später heran, um die Nützlichkeit von ADFs zu demonstrieren. Er stellt fest:

> „[...] the parity functions are the hardest Boolean functions to find via blind random search of the space of S-expressions composed using the function set [{AND, OR, NAND, NOR}] and they are the hardest to learn via genetic programming.“

Die Schwierigkeit des Parityproblems erklären Langdon und Poli[64, 65] mit der unter Umständen fehlenden Gradienteninformation. So ist die logische Operation EQ hinreichend, um jede Instanz des beschriebenen Problems mit einer geraden Eingabegröße zu lösen. Prinzipiell wäre damit die EQ-Operation ein geeigneter Building Block, allerdings lösen alle hiermit erzeugten Funktionen, mit Ausnahme der perfekten Lösung, genau die Hälfte aller Fallbeispiele, sodass keine Gradienteninformation existiert. Anderen Funktionsmengen, die in der Lage sind, hiervon abweichende Fitnesswerte einzunehmen und somit unter Umständen geeignete Gradienteninformationen zu erzeugen, kann die Entstehung der EQ-Operation als Building Block damit zum Nachteil gereichen.

Koza [62] berichtet von einem Aufwand von 1 276 000 Evaluationen, um mit 99% Erfolgswahrscheinlichkeit eine Lösung zu evolvieren. Die Evolution einer Lösung für eine Eingabe der Größe 5 gelang ihm nur einmalig unter Verwendung einer großen Population. Die Lösung größerer Instanzen des Problems benötigte ADFs. Bei späteren Betrachtungen des Problems ohne ADFs verwendet Koza [63] dann generell eine Populationsgröße von 16 000 Individuen (vormals 4 000). Der ermittelte Aufwand für die 4 Bit Eingabe beträgt hier 384 000 Evaluationen, um eine 99% Erfolgswahrscheinlichkeit zu erreichen, für die Problemstellung mit 5 Eingabe-Bits beträgt der Aufwand 6 538 000 Evaluationen. Größere Problemdimensionen erwiesen sich auch mit dieser Populationsgröße als ohne ADFs unlösbar. Lones [71] ermittelt für sein „Enzym Gentic Programming“-Ansatz einen Aufwand von 2 703 000 Evaluationen bzw. ohne Verwendung der Rekombinationsoperation einen Aufwand von 1 830 000 Evaluationen zur Evolution einer Lösung für die 4 Bit

Tabelle 8.5.: Problemdesign und SPO spezifische Einstellungen für die Evolution einer Paritätsfunktion.

Parameter	Einstellung
Problemdesign	
Funktionsmenge	AND, OR, NAND, NOR
Terminale	4 boolsche Werte
Fitnessfunktion	Anteil Beispiele, für die die Parität falsch erkannt wurde
Trainingsmenge	LGP: 16 / AC: $8 \times 16 = 128$ Fallbeispiele
Terminierung	1 000 000 Evaluationen
initale Länge	25 Instruktionen/Reaktionen
maximale Länge	500 Instruktionen/Reaktionen
Validierungsmenge	LGP: 16 / AC: $8 \times 16 = 128$ Fallbeispiele
Testmenge	LGP: 16 / AC: $8 \times 16 = 128$ Fallbeispiele
Sequenzielle Parameteroptimierung	
Budget	2000 Experimente
Indifference-Zone	1/32

Instanz des Problems. Miller u. a. [77] ermitteln leider nicht den Aufwand zum Erreichen der 99% Erfolgswahrscheinlichkeit mit ihrem Cartesian GP-System, geben aber an, dass aus 100 Läufen, mit je 4×10^6 Evaluationen, 15 Läufe erfolgreich waren.

Tabelle 8.5 gibt einen Überblick über das Problemdesign und die SPO spezifischen Einstellungen. Wie bereits oben beschrieben, besteht die Terminalmenge aus 4 booleschen Werten, für die Wahl der Operationen kann auf die Funktionsmenge {AND, OR, NAND, NOR} zurückgegriffen werden. Für das LGP-System wurden, wie für dieses Problem üblich, alle der 16 möglichen Eingaben Teil der Trainings- sowie der Validierungs- und Testmenge[1]. Die Wahl von Trainings-, Validierungs- und Testmenge für das AC-System wird im Abschnitt 8.2.1 beschrieben.

Beim Parityproblem ist eine guten Annäherung uninteressant. Es interessieren ausschließlich Läufe, bei denen die perfekte Lösung gefunden wurde. Dieses ist dann eine boolesche Funktion, die für alle Eingaben die richtige Parität ermittelt. Somit besteht die eigentliche Aufgabe der Parameteroptimierung darin, eine Einstellung zu finden, die eine möglichst hohe Erfolgsrate aufweist. Die Optimierung der Parametrisierung bzgl. dieses Kriteriums erwies sich jedoch als schwierig. In den initialen Läufen mit ihrer geringen Anzahl Wiederholungen waren zu wenige Erfolge zu verzeichnen, um mit ihnen geeignete und ungeeignete Parameterbereiche zu differenzieren. Die erwartete Verbesserung, auf deren Grundlage weitere Punkte gewählt werden, basiert dann hauptsächlich auf der Distanz zu bereits

[1] Da alle möglichen Fallbeispiele bereits beim Training verwendet werden müssen, sind Validierung und Test bei der deterministischen Auswertung überflüssig.

betrachteten Einstellungen. In der Hoffnung auf verwertbare Gradienteninformationen wird als zu minimierendes Optimierungskriterium während der SPO der Anteil an Fallbeispielen bewertet, der vom besten Individuum der Population falsch gelöst wird. Bei späteren Evolutionsläufen wird dann ausschließlich wieder die Erfolgsrate, d. h. die Häufigkeit, mit der alle Fallbeispiele gelöst werden, betrachtet.

Da bei einer Eingabegröße von 4 lediglich $2^4 = 16$ unterschiedliche Eingaben existieren, beträgt der kleinste mögliche Fitnessunterschied $1/16$. Die Indifference-Zone wurde auf die Hälfte dieses Werts festgelegt. Jedem Evolutionslauf standen 10^6 Evaluationen für die Suche zur Verfügung. Für die SPO jedes Systems wurde ein Budget von 2000 Experimenten verwendet.

8.2.1. Rauschen

Die nichtdeterministische Ausführung der Individuen führt bei der Auswertung zu Rauschen. Es wird an dieser Stelle diskutiert, da es sich bei einer kleinen Trainingsmenge, wie bei dieser Instanz der Parityproblems mit $2^4 = 16$ Fallbeispielen, besonders bemerkbar macht.

Das lineare Individuum

$$b_{\text{res}} = \overline{b_{i1} \wedge b_{i1}}$$
$$b_{\text{res}} = b_{i1} \wedge b_{i1}$$

aus zwei Instruktionen, die nacheinander in das Ergebnisregister b_{res} schreiben, gibt immer den booleschen Eingabewert b_{i1} zurück. Dies entspricht in 50% der betrachteten Fallbeispielen dem gesuchten Wert. Dem Individuum wird eine Fitness von 0.5 zugewiesen.

Die Auswertung als algorithmische Chemie führt jedoch dazu, dass zufällig eine der beiden Instruktionen das Ergebnisregister beschreibt, was mit einer Wahrscheinlichkeit von $p = 0.5$ dem gesuchten Wert entspricht und mit der Wahrscheinlichkeit $q = 1 - p = 0.5$ dazu führt, dass das Individuum die falsche Lösung berechnet. Die Anzahl k der richtig gelösten Fallbeispiele (n Stück) ist binomial verteilt

$$B(k|p,n) = \binom{n}{k} p^k (1-p)^{n-k}, \quad 0 \leq k \leq n \quad .$$

Der Mittelwert beträgt np. Die daraus berechnete Fitness $(np)/n = p = 0.5$ entspricht im Mittel der Fitness des linearen Individuums, der Fitnesswert ist annähernd normal verteilt (wenn $np = n(1-p) > 5$ [79]). Diese Normalverteilung kann als Rauschen interpretiert werden, welches vor allem bei hohem Selektionsdruck und damit hohem Nachkommenüberschuss die Evolution fehlleitet.

Mit Erhöhung der Anzahl betrachteter Fallbeispiele um den Faktor m durch Duplikation wird die denkbar einfachste Methode zur Rauschreduktion gewählt. Allerdings nimmt die Varianz der ermittelten Fitnesswerte dabei nur um den Faktor $\sqrt{m}$ ab. Andere Verfahren zur Erstellung einer Rangordnung und zur Selektion arbeiten hier effizienter (siehe Bechhofer u. a. [20]). Im Folgenden wird

der Faktor $m = 8$ verwendet, d. h., durch Duplikation werden Trainings-, Test- und Validierungsmenge auf 128 Fallbeispiele vergrößert.

Die Tabelle A.1 im Anhang enthält die besten Einstellungen aus den SPO-Läufen beider Systeme sowie Informationen über den eigentlichen Verlauf der SPO. Die Abbildungen A.1 und A.2 stellen die Effekt- und Interaktionsplots der Parameter beider Systeme dar. Die für das AC-System auf dem Paritätsproblem gefundene Einstellung wich bzgl. Populationsgröße und Selektionsdruck stark von den Einstellungen für die anderen Probleme ab. In Kap. 9.3 wird daher zum einen die für dieses Problem mittels SPO gefundene Einstellungen getestet, zum anderen aber auch eine angepasste Einstellung auf der Grundlage der anderen SPO-Läufe. Beide liefern bzgl. des verwendeten Optimierungskriteriums vergleichbare Werte, die angepasste Version schneidet aber bei der Erfolgsrate deutlich besser ab. Die Erfolgsrate ist das hier eigentlich interessierende Kriterium, welches sich aber als ungeeignet für die Optimierung mittels SPO erwies. Die Effektplots für diese Problem-Algorithmen-Kombination stützen die mittels SPO gefundene Einstellung, indem sie ihr eine Ergebnisverbesserung gegenüber dem Durchschnitt prognostizieren. Die Verwendung der mittleren Systemleistung anstelle der Erfolgsrate scheint, zumindest auf das AC-System bezogen, ein unzureichendes Ersatzkriterium für die Optimierung zu sein. Ein alternatives Vorgehen wäre die Erhöhung der Anzahl Experimente pro (initialem) Designpunkt gewesen, um so deutlicher Unterschiede der Designpunkte zu ermitteln und im Modell wiederzugeben. Allerdings hätte dann auch das verwendete Budget deutlich höher ausfallen müssen.

8.3. Thyroid: Klassifikation

Die Aufgabe beim Thyroidproblem[2] aus dem UCI-Repository [81] besteht in der Klassifikation der Schilddrüsenfunktion anhand von klinisch erhobenen Patientendaten. Jedes Patientendatum besteht dabei aus 15 binären und 6 reellwertigen Messwerten. Die Funktion der Schilddrüse wird zwei Klassen möglicher Fehlfunktionen sowie einer Klasse für die Normalfunktion zugeordnet. Die beiden Klassen möglicher Fehlfunktionen lassen sich durch eine einfache Funktion voneinander abgrenzen. Damit müssen nur Fehl- und Normalfunktion unterschieden werden (siehe Gathercole [48]). Der vorhandene Datensatz setzt sich aus 3772 Fallbeispielen für die Trainingsphase und weiteren 3428 Fallbeispielen für eine Testphase zusammen. Da 92% der erfassten Patienten eine normale Funktion der Schilddrüse aufweisen, muss der Klassifikationsfehler deutlich unter 8% betragen, um von praktischem Nutzen zu sein. Der Größe des Datensatzes dieses realen Klassifikationsproblems und seine häufige Betrachtung in der Literatur führen zur Auswahl des Thyroidproblems als Probleminstanz.

Gathercole [48] verwendet ein baumbasiertes GP-System und erzielt unter Einbeziehung sämtlicher Trainingsdaten bei jeder Auswertung und bei der Wahl einer 400 Fallbeispiele umfassenden Trainingsmenge mittels „Dynamic Subset Selection“

[2] Thyroid ist der englische Begriff für Schilddrüse.

Tabelle 8.6.: Problemdesign und SPO spezifische Einstellungen für die Suche nach einer Klassifikationsfunktion für die Schilddrüsenfunktion.

Parameter	Einstellung
Problemdesign	
Funktionsmenge	$+, -, *, \vee, \wedge, \neg, \sin, \cos, x^y$
Terminale	Patientendaten: 15 binäre/6 reellwertig, 5 Konstante
Fitnessfunktion	Anteil der falsch klassifizierten Schilddrüsenfunktionen
Trainingsgesamtmenge	2772 Fallbeispiele
Trainingsmenge	400 Fallbeispiele per SSS
Terminierung	500 000 Evaluationen
initale Länge	25 Instruktionen/Reaktionen
maximale Länge	500 Instruktionen/Reaktionen
Validierungsmenge	1000 Fallbeispiele
Testmenge	3428 Fallbeispiele
Sequenzielle Parameteroptimierung	
Budget	1200 Experimente
Indifference-Zone	–

eine Klassifikationsrate von 99%. Unter Verwendung von „Random Subset Selection“, welche dem hier verwendeten „Stochastic Subset Sampling“ (SSS) am nächsten kommt, erzielt er Klassifikationsraten bis 98.40%. Schiffmann u. a. [103] erreichten mittels neuronaler Netze, deren Topologie sie mit genetischen Algorithmen optimieren, Klassifikationsraten zwischen 96.9% und 97.5%. Durch eine zusätzliche Lernphase konnte die Leistung der neuronalen Netze auf bis zu 98.6% gesteigert werden. Sie betrachten das Problem in der ursprünglichen Form, d. h. mit drei zu unterscheidenden Klassen. Weiss und Kapouleas [116] wenden unterschiedliche Techniken des maschinellen Lernens an und erzielen mit Klassifikations- und Regressionsbäumen eine Erfolgrate von ca. 99.4%.

Tabelle 8.6 beschreibt das Problemdesign und listet SPO spezifische Einstellungen. In der Funktionsmenge kommen arithmetische, logische und trigonometrische Funktionen zum Einsatz. Die Terminale werden aus den Daten des betrachteten Fallbeispiels und fünf Konstanten, die im Individuum evolviert werden, gebildet. Die Klassifikation geschieht durch Interpretation der Ausgabe des Individuums. Ein Wert größer null wird als Zuordnung zu einer normalen Schilddrüsenfunktion, ein Werte kleiner oder gleich Null als Zuordnung zu einer der beiden Fehlfunktionen interpretiert. Die Fitness berechnet sich aus dem Anteil falsch klassifizierter Schilddrüsenfunktionen. Gathercole [48] verwendet für seine baumbasierten GP-Läufen zwischen 1.6×10^8 und 11.3×10^8 Baumevaluationen. Für die meisten Versuche erlaubt er dabei etwas über 2×10^8 Baumevaluationen und eine Trainingsmengen-

größe von 400 Fallbeispielen. Dieses führt hier zur Festlegung auf 400 Fallbeispiele, die mittels SSS aus der Menge aller Trainingsbeispiele in jeder Runde gewählt werden. Der Aufwand wird auf 5×10^5 Individuenevaluationen festgelegt, nach denen die Läufe terminiert werden. Der ursprünglichen Menge an Trainingsdaten wird eine 1000 Fallbeispiele umfassende Validierungsmenge entnommen, wobei der Relation der Klassen zueinander entsprochen wird. Die Testmenge bleibt unverändert, die Ergebnisse auf ihr sind somit mit denen in der Literatur vergleichbar. Allerdings bleiben ca. 27% der ursprünglichen Fallbeispiele aufgrund ihrer Verschiebung in eine Validierungsmenge für das Training unbekannt.

Die Tabelle A.2 enthält die Besten der betrachteten Einstellungen beider Systeme und Informationen über den Verlauf der SPO. Die Abbildungen A.3 und A.4 zeigen die aus den Daten der SPO generierten Effekt- und Interaktionsplots. Im Durchschnitt über alle Parametrisierungen übt die Mutationsrate den größten Einfluss auf die Güte einer Einstellung aus. Die Tabellen zeigen, dass beide Systeme eine geringe Populationsgröße bevorzugen. Während das LGP-System diese mit einem mittleren Selektionsdruck kombiniert, bevorzugt das AC-System einen höheren Selektionsdruck. Wieder dominieren geringe Rekombinationsraten die guten Einstellungen für das AC-System.

Zusammenfassung und Diskussion

In diesem Abschnitt wurden die Paritätsfunktion, die symbolische Regression der Sinusfunktion und die Klassifikation der Schilddrüsenfunktion als Problemstellungen und die Optimierung der beiden Systeme bezüglich der drei Probleme vorgestellt. Die unabhängige SPO dient als Grundlage für einen fairen Vergleich beider Systeme in den folgenden Abschnitten. Über die Grenzen der betrachteten Probleme hinweg lassen sich während der Optimierungsphase folgenden Beobachtungen machen:

- Bei zwei der drei betrachteten Probleme wurde aus den Experimenten an den initialen Designpunkten offensichtlich, dass eine Anpassung des Problemdesigns notwendig ist. So musste beim Parityproblem anstelle der Erfolgsrate mit einer Optimierung der durchschnittlichen Fitness gearbeitet werden. Das Sinusproblem erforderte eine Anpassung der Ausreißer in den Experimenten einer Einstellung. Dieses verdeutlicht eindringlich, dass auch die SPO nur semiautomatisch angewendet werden sollte und für die Optimierung des Algorithmendesigns unter Umständen ein angepasstes Problemdesign benötigt wird.

- Beim LGP-System befinden sich unter den besten Parametrisierungen deutlich mehr Einstellungen, die bereits aus dem initialen raumfüllenden Design stammen. Dieses lässt vermuten, dass das LGP-System generell stabiler bzgl. der Parametrisierung ist.

- Beide Systeme bevorzugen für alle drei Probleme vergleichbare Einstellungen, sodass zu hoffen ist, bei neuen Problemen in einem ersten Schritt mit diesen Parametern gute Ergebnisse zu erzielen.
- Die gefunden Einstellungen befinden sich für einzelne Parameter an den Grenzen der verwendeten „region of interest“. Für die Eltern hätte hier eine kleinere Untergrenze, für den Selektionsdruck eine höhere Obergrenze gewählt werden können.

Die bisherige Rekombination ist zunächst durch den Wunsch motiviert, den Unterschied der Systeme gering zu halten. Für die Realisierung ihrer Rekombinationsoperation wurde auf Unterschiede im Programmzeigerverhalten bei der Ausführung zurückgegriffen. Für alle drei Problemstellungen bevorzugt das AC-System eine äußerst geringe Rekombinationsrate von unter 3%. Diese ist ein erster Hinweis darauf, dass die hier gewählte Realisierung der Rekombination nicht für die Erzeugung guter Individuen geeignet ist. Kapitel 11 wird sich erneut mit dem Rekombinationsoperator beschäftigten.

Teil IV.

Empirische Ergebnisse

9. Evolutionsverlauf

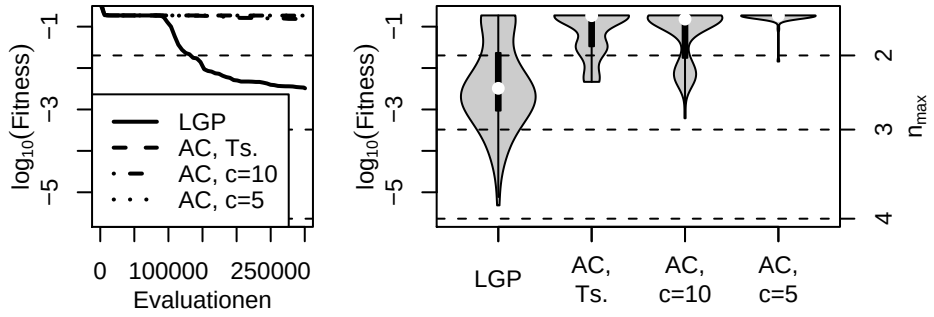
Nachdem im vorherigen Kapitel die Parameter der GP-Systeme zur Evolution eines linearen Programms bzw. einer algorithmischen Chemie mit identischem Aufwand optimiert wurden, wird hier die Evolution von Lösungen unter Verwendung der gefundenen Parameter untersucht. Das AC-System wurde unter Verwendung der Totalsynthese optimiert, welche zur Reduzierung des SPO-Aufwands eingeführt wurde. Im Folgenden werden algorithmische Chemien zudem ohne Verwendung der Totalsynthese evolviert. Ohne Totalsynthese wird ein Vielfaches c der im Genom codierten Reaktionen zufällig gewählt und zur Ausführung gebracht. Hierbei werden die Vielfachen $c = 5$ und $c = 10$ betrachtet. Es wird dabei Aufschluss darüber gewonnen werden, ob die zuvor unter Verwendung der Totalsynthese für die AC-Varianten gefundenen Parameter auch auf die Systemvarianten ohne Totalsynthese übertragbar sind.

Die vorrangige Frage ist zunächst, in wieweit sich die Verwaltung von Datenflussinformationen in der zeitkontextfreien mehrdeutigen Form der algorithmischen Chemie zur Evolution von Lösungen für die vorgestellten Probleme eignet. Durch den Vergleich mit LGP als Referenzsystem wird der Frage nachgegangen, in welcher Form sich die Evolution einer Lösung bzgl. Geschwindigkeit und den Eigenschaften der Lösungen verändert, wenn auf eine explizite Codierung des Datenflusses verzichtet wird.

9.1. Sinus

Mit den gegebenen Operationen und Terminalen ist eine Approximation der Sinusfunktion ähnlich der Potenzreihenentwicklung möglich. Abbildung 9.1(a) stellt den Evolutionsverlauf der einzelnen Systeme im Median aus 100 Experimenten dar. Die horizontalen, gestrichelten Linien markieren die Güte der Potenzreihenentwicklung aus Gl. (8.1) unter Verwendung unterschiedlicher $n_{\max}$ Werte. Die Verteilung der in diesen Läufen erzielten Fitness bis zum Zeitpunkt der Terminierung ist in Abb. 9.1(b) dargestellt. Bei diesen *Violinenplots* handelt es sich um eine Kombination aus Boxplot und geschätzter Dichtefunktion. Die Boxplots zeigen den Bereich zwischen dem ersten und dritten Quartil an, der weiße Punkt entspricht dem Median. Grau, zu beiden Seiten dargestellt, ist die Schätzung der Dichtefunktion. Die Fitness wurde in beiden Diagrammen logarithmisch aufgetragen, da sie bei Hinzunahme weiterer Glieder der Protenzreihe nur negativ exponentiell abnimmt.

Zunächst lässt sich das beim Algorithmendesign (Kap. 8.1) diskutierte Problem eines starken lokalen Optimums bei den Experimenten der AC-Varianten beob-



(a) Evolutionsverlauf im Median

(b) Verteilung der Einzelläufe nach 3×10^5 Evaluationen

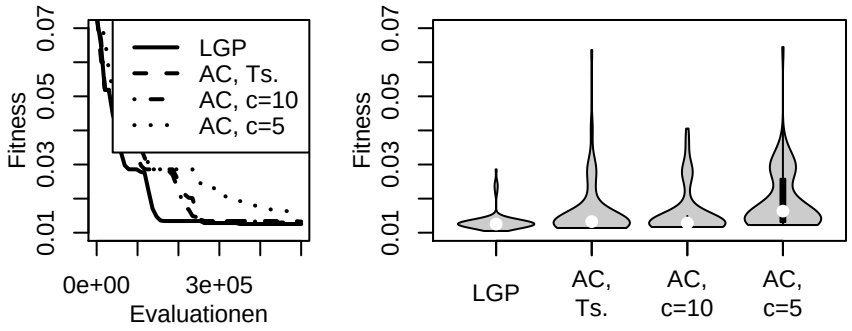
Abbildung 9.1.: (a) Verlauf der Evolution einer Approximation für die Sinusfunktion und (b) Verteilung der Resultate in den einzelnen Läufen. Die Fitness entspricht der mittleren quadratischen Abweichung für 400 Fallbeispiele der Testmenge, die das Individuum erzeugt welches als Bestes aus der Validierung hervorgeht.

Tabelle 9.1.: p-Werte der paarweisen Wilcoxon-Rangsummentests für das Sinusproblem. Die p-Werte wurden nach Holm [52] angepasst.

	LGP	AC, Ts.	AC, $c = 10$
AC, Ts.	< 0.001	–	–
AC, $c = 10$	< 0.001	0.103	–
AC, $c = 5$	< 0.001	0.103	< 0.001

achten. Die Lage des Medians zeigt, dass mehr als die Hälfte aller Läufe auch bei optimierter Parametrisierung in das lokale Optimum fallen, weswegen der Median bei der SPO als Optimierungskriterium nicht geeignet erschien.

Ein Vergleich der Systeme zeigt, dass das LGP-System bessere Approximationen der Sinusfunktion evolviert, als dieses bei den AC-Varianten der Fall ist. Die Nullhypothese über die Gleichheit der Systeme wird vom Kruskal-Wallis-Test zum Signifikanzniveau $\alpha = 0.05$ verworfen, $p < 0.001$. Tabelle 9.1 enthält die Resultate der paarweisen Wilcoxon-Rangsummentests. Die enthaltenen p-Werte wurden nach Holm [52] angepasst. Die Tabelle zeigt, dass die Hypothese über die Gleichheit mit dem LGP-System für alle drei Varianten des AC-Systems verworfen werden kann. Die Unterschied beider Systeme in der erzielten Lösungsgüte, bezogen auf die Komplexität der hierfür benötigten Potenzreihe, entspricht einem Reihenglied, welches vom LGP-System zumeist mehr evolviert wird. Als Ursachen für die frühere Stagnation der AC-Varianten wird der steigende Einfluss des Rauschens mit sinkender Verbesserungsmöglichkeit vermutet und die mit der Individuenhöhe sinkende Wahrscheinlichkeit, dass der im Individuum codierte Datenfluss vollständig



(a) Evolutionsverlauf im Median

(b) Verteilung der Einzelläufe nach 5×10^5 Evaluationen

Abbildung 9.2.: (a) Verlauf des Medians bei der Evolution einer Klassifikation für die Schilddrüsenfunktion und (b) Verteilung der Resultate in den einzelnen Läufen. Die Fitness entspricht dem Anteil falsch klassifizierter Patientendaten der Testmenge, gemessen beim besten Individuum auf der Validierungsmenge.

assembliert wird.

Die Abhängigkeit der erreichten Lösungsgüte von der Anzahl ausgeführter Reaktionen zeigt sich in den AC-Evolutionsläufen ohne Totalsynthese. Während die Variante, die ein Vielfaches von $c = 10$ der codierten Reaktionen ausführt, in nahezu 25% der Experimente die Potenzreihe mit $n_{\max} = 2$ in der Eignung als Approximation der Sinusfunktion übertrifft, erreicht die Variante mit $c = 5$ diese Resultate nur einmalig. Der Wilcoxon-Rangsummentest (Tab. 9.1) zeigt, dass die Hypothese auf Gleichheit der bei $c = 5$ und $c = 10$ erreichten Fitnesswerte verworfen werden kann. Es zeigt sich, dass die unter Verwendung der Totalsynthese bei der SPO gefundenen Parameter auf die Varianten ohne Totalsynthese übertragen werden können. Diese trifft aber keine Aussage darüber, ob dieses eine besonders geeignete Einstellung für die Läufe ohne Totalsynthese ist.

9.2. Thyroid

Abbildung 9.2 stellt den Verlauf der Evolution einer Funktion zur Klassifikation der Schilddrüsenfunktion im Median (a) und die Verteilung (b) der erzielten Resultate dar. Der zeitliche Verlauf zeigt eine etwas frühere Evolution guter Lösungen für das lineare GP-System. Zudem existiert ein lokales Optimum bei einem Klassifikationsfehler von etwa 3%, in dem ein kleiner Teil der Läufe stagniert.

Mittelwert, Median und die beste jeweils erzielte Klassifikationsrate sind in Tab. 9.2 aufgelistet. Sowohl das LGP-System als auch die AC-Varianten erzielen Klassifikationsraten, die mit denen aus der Literatur (siehe Kap. 8.3) vergleichbar sind. Dieses gilt auch für die zuvor vergleichsweise schlechte AC-Variante, die das Vielfache $c = 5$ der im Individuum codierten Reaktionen ausführt. Um die

Tabelle 9.2.: Klassifikationsraten der betrachteten Systeme (in %) auf dem Thyroidproblem. Dargestellt sind Mittelwert, Median und das beste Ergebnis aus jeweils 100 Experimenten.

Verfahren	Mittel.	Median	Max.
LGP	98.68	98.75	98.95
AC, ts.	98.40	98.67	98.86
AC, $c = 10$	98.41	98.72	98.83
AC, $c = 5$	98.10	98.37	98.77

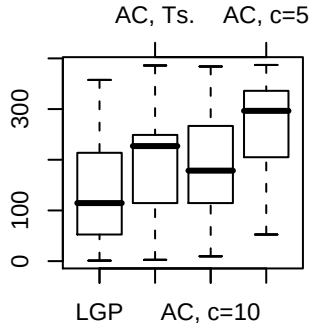


Abbildung 9.3.: Boxplots der Ränge der erzielten Ergebnisse

Tabelle 9.3.: p-Werte der paarweisen Wilcoxon-Rangsummentests für das Thyroidproblem. Die p-Werte wurden nach Holm [52] angepasst.

	LGP	AC, Ts.	AC, $c = 10$
AC, Ts.	< 0.001	–	–
AC, $c = 10$	0.001	0.447	–
AC, $c = 5$	< 0.001	< 0.001	< 0.001

Unterschiede in der Verteilung der Resultate zu verdeutlichen, zeigt die Abb. 9.3 den Boxplot der Ränge, die die einzelnen Resultate in der Menge aller erzielten Resultate einnehmen.

Während die Systeme in ihren absoluten Werten nur einen geringen Unterschied aufweisen, deuten die Boxplots auf einen Unterschied in der Verteilung der Ergebnisse hin, der Kruskal-Wallis-Test bestätigt mit $p < 0.001$, dass die Hypothese auf Gleichheit der Systeme zum verwendeten Signifikanzniveau verworfen werden kann. Paarweise Vergleiche mittels Wilcoxon-Rangsummentest (siehe Tab. 9.3) ergeben, dass die Nullhypothese von der Gleichheit zweier Systeme lediglich für den Vergleich zwischen der AC-Evolution mit Totalsynthese und ohne Totalsynthese mit $c = 10$ nicht zum erforderlichen Signifikanzniveau verworfen werden kann. Die Ähnlichkeit der Ergebnisverteilung beider Systeme zeigt sich auch in den Boxplots und bestätigt erneut, dass die in der SPO mit Totalsynthese ermittelten Parameter auch ohne Totalsynthese ihre Gültigkeit behalten.

9.3. Parity

Anders als beim Sinusproblem und der Klassifikation der Schilddrüsenfunktion (Thyroid) sind bei der Evolution der Paritätsfunktion nur Individuen mit einer vollständigen Lösung des Problems interessant. Eine zusätzliche Differenzierung der Individuen anhand der Anzahl gelöster Fallbeispiele dient ausschließlich der

Tabelle 9.4.: Mittels SPO ermittelte Einstellung und die angepasste Einstellung für das AC-System auf dem Parityproblem.

	Eltern	Selek.druck	Mut.rate	Rekomb.rate	Moleküle
SPO	807	4.08	2.27%	0.12%	24
angepasst	**150**	**6.8**	2.27%	0.12%	24

Steuerung des Evolutionsverlaufs durch Selektion der besseren Individuen.

Dem Umstand, dass nur vollständige Lösungen Verwendung finden, wird bei der Analyse über die kumulative Erfolgswahrscheinlichkeit $P_s(e)$ und der Berechnung des Mindestaufwands für eine vorgegebene Erfolgswahrscheinlichkeit z Rechnung getragen. Die kumulative Erfolgswahrscheinlichkeit $P_s(e)$ ist abhängig von der Evolutionsdauer, gemessen anhand der Anzahl ausgeführter Evaluationen e, und entspricht dem Anteil der Läufe, die mit einem Aufwand $\leq e$ eine vollständige Lösung gefunden haben. Die Wahrscheinlichkeit z, in R unabhängigen Läufen mit jeweils e Evaluationen eine vollständige Lösung zu finden beträgt:

$$z = 1 - (1 - P_s(e))^R .$$

Die Auflösung nach R ergibt die Anzahl der Läufe $R(z, e)$ die in Abhängigkeit der Evaluationen e nötig sind, um eine Erfolgwahrscheinlichkeit z zu erreichen:

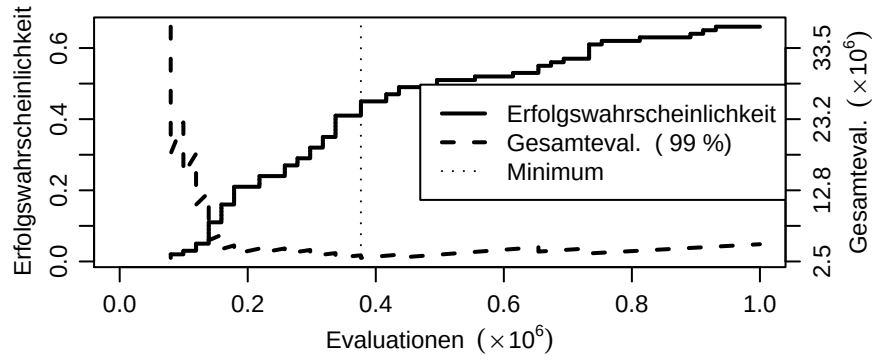
$$R(z, e) = \left\lceil \frac{\log(1 - z)}{\log(1 - P_s(e))} \right\rceil .$$

Der Gesamtaufwand

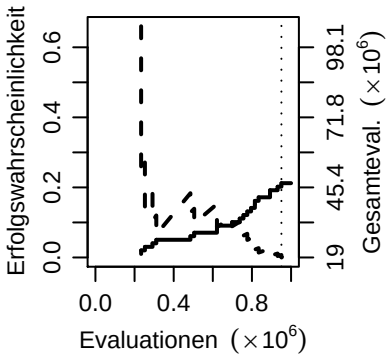
$$R(z, e) \cdot e \tag{9.1}$$

kann durch geeignete Wahl der Evaluationen e pro Einzellauf minimiert werden.

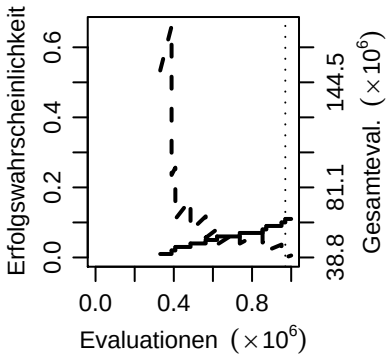
Tabelle 9.4 enthält die mittels SPO gefundene Parametrisierung des AC-Systems. Abbildung 9.4 (d&e) zeigt die hiermit erzielten Resultate für die AC-Systeme mit Totalsynthese bzw. unter Verwendung eines Vielfachen $c = 10$. Für $c = 5$ kann kein erfolgreicher Lauf beobachtet werden. Der Performanceunterschied zwischen dem LGP-System und den AC-Varianten führt zur genaueren Betrachtung der mittels SPO gefundenen Parameter. Ein Vergleich der Parameter aus Tab. 9.4 mit den für andere Problemstellungen gefundenen Einstellungen (siehe Tab. 8.1) zeigt große Unterschiede bei der Populationsgröße und dem Selektionsdruck. Basierend auf den optimierten Parametern für die anderen betrachteten Problemstellungen wird eine geeignete Populationsgröße von 150 Individuen und ein Selektionsdruck von 6.8 angenommen. Mit diesen Parametern werden weitere Experimente für die AC-Varianten durchgeführt. Die Verläufe sind in Abb. 9.4 (b&c) dargestellt. Beide AC-Varianten weisen mit den modifizierten Einstellungen am Ende der Laufzeit eine deutlich höhere Erfolgsrate auf. Die Variante mit $c = 5$ findet auch diesmal



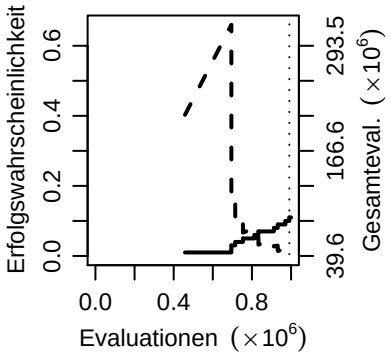
(a) Lineares GP



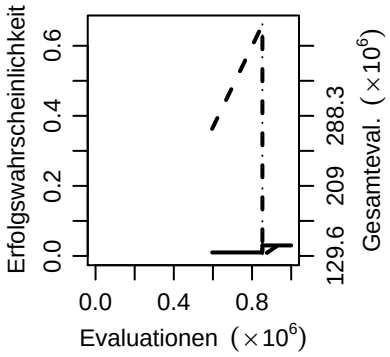
(b) AC, mit Totalsynthese, modifizierte Parameter



(c) AC, mit $c = 10$, modifizierte Parameter



(d) AC, mit Totalsynthese, SPO basierte Parameter



(e) AC, mit $c = 10$, SPO basierte Parameter

Abbildung 9.4.: Erfolgswahrscheinlichkeit in Abhängigkeit der Zeit und Gesamtaufwand $R(z, e) \cdot e$ für einen Erfolg mit der Wahrscheinlichkeit $z = 99\%$. Zusätzlich markiert ist die Anzahl Evaluationen e, für die der Gesamtaufwand minimal wird.

Tabelle 9.5.: Für die unterschiedlichen Systemvarianten ist angegeben: die maximal beobachtete Erfolgsrate $P_s^{\max}$, die ideale Anzahl Evaluationen $e^\star = \arg\min R(99\%, e) \cdot e$ pro Lauf um den Aufwand zum Erreichen einer Erfolgswahrscheinlichkeit von 99% zu minimieren, die dazugehörige Anzahl an Läufen $R(z, e^\star)$ sowie der daraus resultierende Gesamtaufwand $e^\star \cdot R(z, e^\star)$.

Verfahren	$P_s^{\max}$	$e^\star$	$R(z, e^\star)$	$e^\star \cdot R(z, e^\star)$
LGP, SPO	0.66	376740	8	3013920
AC, Totalsynthese, mod.	0.21	950640	20	19012800
AC, $c = 10$, mod.	0.11	970020	40	38800800
AC, Totalsynthese, SPO	0.11	991193	40	39647720
AC, $c = 10$, SPO	0.03	852887	152	129638824

nicht die gesuchte Paritätsfunktion. Vermutlich ist die Anzahl ausgeführter Reaktionen nicht ausreichend, um Funktionen der benötigten Höhe mit hinreichender Wahrscheinlichkeit zu assemblieren.

Wieso wurde die modifizierte Einstellung nicht während der SPO gefunden? In Kapitel 8.2 wurde bereits darauf eingegangen, dass nicht die hier betrachtete Erfolgswahrscheinlichkeit optimiert wurde, sondern die durchschnittliche Fitness. Diese beträgt für die mittels SPO gefundene Einstellung 0.147 und für die modifizierte Einstellung 0.165. Bezüglich des bei der SPO vorgegebenen Designkriteriums wurde folglich die bessere Einstellung gefunden, allerdings korreliert dieses zwangsweise verwendete Kriterium nicht hinreichend mit dem intendierten Kriterium. Es bleibt anzumerken, dass die Anpassung über die Parameter anderer Problemstellungen erfolgreich war. Das lässt hoffen, dass auch bei neuen Aufgaben vergleichbare Einstellungen in erster Annahme verwendet werden können. In den folgenden Abschnitten wird ausschließlich die modifizierte und in Tab. 8.1 angegebene Einstellung verwendet.

Ziel ist, mit einer Wahrscheinlichkeit $z = 99\%$ einen Erfolg zu erzielen. Tabelle 9.5 enthält die aus der Minimierung von Gl. (9.1) ermittelte optimale Evaluationszahl je Lauf, die Anzahl benötigter Läufe sowie den hieraus resultierenden Gesamtaufwand der verschiedenen Systeme. Daraus folgt für die AC-Varianten, dass die optimale Laufzeit am Ende des betrachteten Zeitraums liegt. Es besteht die Möglichkeit, dass der Gesamtaufwand mit längerer Laufzeit der Einzelläufe noch weiter sinkt. Dennoch ist nicht davon auszugehen, dass der Aufwand für die AC-Varianten in den Bereich des LGP-Systems gelangt. Ein Vergleich zwischen den Läufe mit modifizierten Einstellungen und den von der SPO eingestellten Läufen zeigt eine deutliche Aufwandsreduzierung. Die AC-Variante mit Totalsynthese benötigt nun nur noch die Hälfte, die Variante mit einem Vielfachen von $c = 10$ sogar nur noch ein Drittel der Evaluationen. Paarweise Anteilstests zeigen, dass sich die Erfolgswahrscheinlichkeit der damit parametrisierten AC-Varianten weiterhin signifikant ($\alpha = 0.05$) von der Erfolgsrate des LGP-Systems unterscheidet. Die

Hypothese auf Gleichheit der beiden AC-Varianten kann nicht zum erforderlichen Signifikanzniveau verworfen werden, $p = 0.077$.

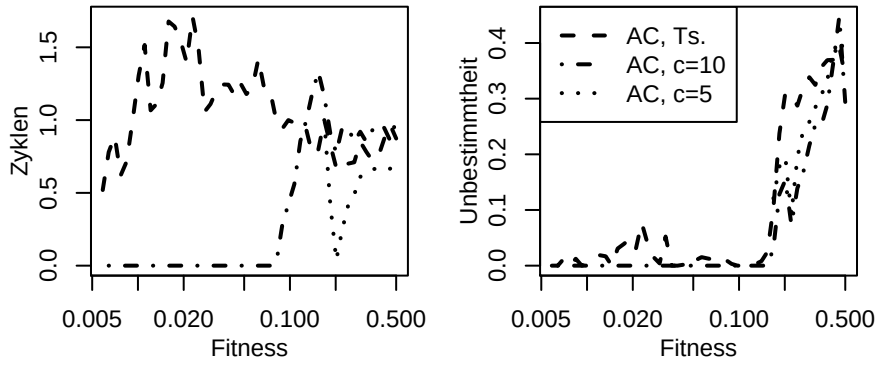
9.4. Ordnung: Unbestimmtheit und Zyklen

Im Kapitel 6 wurden zwei Maße eingeführt, mit deren Hilfe bewertet werden soll, in welchem Maß Ordnung in der algorithmischen Chemie des besten Individuums entsteht. Ordnung wird hier im Sinne einer Reproduzierbarkeit der Ergebnisse verstanden.

Mehrere Ausführung einer algorithmischen Chemie können aufgrund der nichtdeterministischen Auswahl der Reaktionen, auch bei hinreichender Anzahl ausgeführter Reaktionen und identischen Anfangszuständen, zu unterschiedlichen Endzuständen führen. Hierfür sind zwei unterschiedliche Eigenschaften verantwortlich. Erstens können unterschiedliche Reaktionen der Chemie den Zustand eines einzelnen Moleküls ändern. Dann hängt der Zustand zum Zeitpunkt der Verwendung von der letzten Reaktion ab, die dieses Molekül als Produkt genutzt hat. Eine Mehrdeutigkeit folgt auch für alle Moleküle, deren Zustand sich über eine Reihe von Reaktionen aus diesem Molekül ergibt. Existiert für alle Moleküle genau eine Art von Reaktion, die durchaus mehrfach im Individuum codiert sein darf, so können Schleifen bei wiederholter Ausführung zu unterschiedlichen Zuständen führen. Eine Schleife liegt dann vor, wenn sich ein Zustand über eine Reihe von Reaktionen aus sich selbst berechnet. Die unterschiedlichen Endzustände entstehen dadurch, dass die Anzahl vollzogener Schleifendurchläufe zum Zeitpunkt, in dem das Ergebnis der Chemie betrachtet wird, unbestimmt ist.

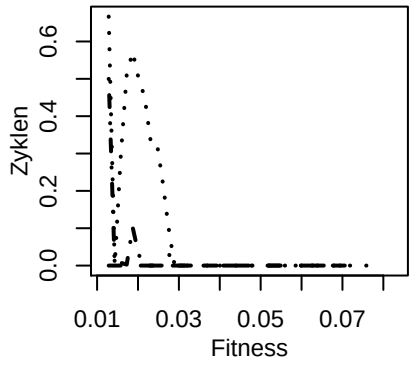
Die erste Eigenschaft wird durch die Unbestimmtheit, die zweite über die Anzahl der Zykleneintritte jeweils für das beste Individuum auf der Validierungsmenge gemessen und in Abb. 9.5 gegen seine Fitness auf der Testmenge aufgetragen. Dabei wurde für jedes System der Median aus allen Experimenten, die eine entsprechende Fitness aufweisen, ermittelt. Da nicht jeder Lauf jeden möglichen Fitnesswert annimmt, werden die Messwerte dazwischen linear interpoliert. Voraussetzung für die Darstellung eines Systems in einem bestimmten Fitnessbereich ist, dass mindestens zehn der zugehörigen Experimente eine Fitness kleiner oder gleich dem entsprechenden Wert erreicht haben.

Zunächst wird der Verlauf der Zykleneintritte und der Unbestimmtheit für die symbolische Regression der Sinusfunktion betrachtet. Bei der Anzahl potenzieller Zykleneintritte in Abb. 9.5(a) fällt auf, dass Lösungen aus Läufen mit Totalsynthese im Bereich guter Fitness zum Teil sehr viele Zyklen enthalten. Zyklen lassen sich bei der Totalsynthese gezielt als Konstanten mit definiertem Wert verwenden, da Zykleneintritte per Definition bei der Totalsynthese zu 0 ausgewertet werden. Die Ausnutzung dieser Eigenschaft könnte eine Erklärung für den beobachteten Anstieg sein. Diese Ausnutzung ist ein unerwünschter Effekt, schließlich soll die Totalsynthese das Verhalten der algorithmischen Chemie bei hinreichend langer Ausführung möglichst gut annähern, ohne den Berechnungsaufwand einer großen

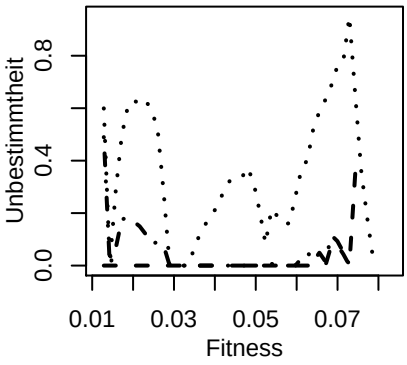


(a) Sinus, Zykleneintritte

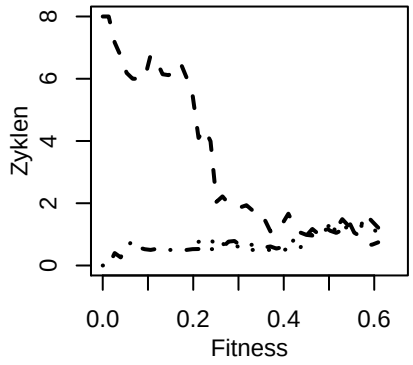
(b) Sinus, Unbestimmtheit



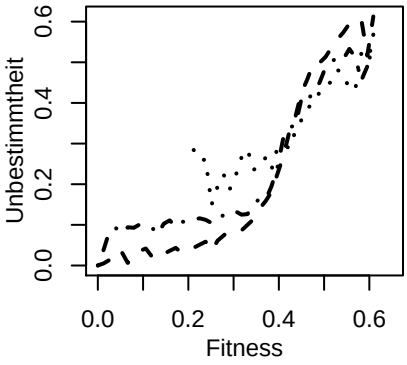
(c) Thyroid, Zykleneintritte



(d) Thyroid, Unbestimmtheit



(e) Parity, Zykleneintritte



(f) Parity, Unbestimmtheit

Abbildung 9.5.: Anzahl Zykleneintritte und Unbestimmtheit des besten Individuums in Abhängigkeit seiner Fitness. Geringe Fitnesswerte stehen für eine bessere Eignung des Individuums.

Anzahl von Reaktionen zu erzeugen. In der gezielten Auswertung zu 0 an den Stellen des Zykleneintritts unterscheidet sich die Auswertung mit und ohne Totalsynthese. Vermutlich hätte diese gezielte Ausnutzung vermieden werden können, wenn anstelle mit dem Wert 0 der Zustand zufällig gesetzt worden wäre. Alternativ hätte, bei etwas höherem Aufwand, eine kleine zufällige Anzahl Zyklen durchlaufen werden können.

Für die Variante mit dem Vielfachen von $c = 10$ wird eine starke Reduktion der Zykleneintritte für den Fitnessbereich gemessen, der eine Verbesserung gegenüber dem beobachteten lokalen Optimum (Fitness bei ca. 0.18) darstellt. Die Unbestimmtheit nimmt für die AC-Varianten mit Überschreiten des lokalen Optimums abrupt ab (siehe Abb. 9.5(b)). Experimente unter Verwendung von $c = 5$ lassen nur Beobachtungen bis in den Bereich des lokalen Optimums zu und wurden hier nicht weiter betrachtet.

Eine vergleichbare Abnahme der Unbestimmtheit lässt sich in Abb. 9.5(f) für das Parityproblem beobachten. Der Zeitpunkt der Abnahme beginnt hier nach den trivialen Individuen, deren Fitness bei 0.5 liegt. Triviale Individuen geben für alle betrachteten Fallbeispiele genau einen der beiden booleschen Werte zurück. Diese Fitness wird im Mittel aber auch dann erreicht, wenn die Lösung geraten wird, was bei Individuen mit hoher Unbestimmtheit der Fall sein kann. Die Unbestimmtheit der Lösungen wird im Laufe der Evolution auf 0 reduziert, d. h. es existiert pro verwendeten Molekül genau eine Art von Reaktion, die seinen Zustand verändert. Abbildung 9.5(e) lässt darauf schließen, dass sich auch hier die Varianten mit Totalsynthese wieder die Auswertung zu 0 bei Zykleneintritt zu Nutzen machen. Dieser Wert wird von den Funktionen der Funktionsmenge als „false“ interpretiert.

Bei Betrachtung der Experimente für das Thyroidproblem unter Verwendung eines Vielfachen von $c = 10$ fällt auf, dass sowohl die Zyklenanzahl in Abb. 9.5(c), als auch die Unbestimmtheit der Lösungen in Abb. 9.5(d) im Bereich des kleinsten beobachteten Klassifikationsfehlers sprunghaft ansteigen. Dieses wird bei den anderen Problemstellungen nicht beobachtet. Denkbar ist, dass hier durch ein Rauschen noch zufällige Verbesserungen erzielt werden. Die Anzahl der Zykleneintritte und die Unbestimmtheit ist für die Läufe mit Totalsynthese bei Lösungen mit einem Klassifikationsfehler von unter 6% im Median 0. Für die Resultate unter Verwendung von $c = 5$ weist vor allem die Unbestimmtheitsmessung starke Schwankungen auf. Die Stärke der Schwankungen ist bedingt durch die geringe effektive Größe der Individuen (vgl. Kap. 10). Die Unbestimmtheit wird normiert auf die Anzahl der in der Lösung involvierten Moleküle, wodurch bei kleinen Lösungen die Unbestimmtheit einzelner Moleküle einen großen Einfluss hat.

Zusammenfassung und Diskussion

Die Reduzierung von Zyklenanzahl und Unbestimmtheit zeigt, dass die genetische Programmierung in der Lage ist, eine algorithmische Chemie mit reproduzierbaren Eigenschaften zielgerichtet zu evolvieren. Die Evolution einer Klassifikation für die

Schilddrüsenfunktion gelang mit den algorithmischen Chemien im Vergleich zu anderen Verfahren gut, blieb aber hinter dem LGP-System zurück. Beim Parityproblem ist bei der Evolution einer algorithmischen Chemie eine deutlich geringere Erfolgsrate als bei der Evolution eines linearen Programms zu beobachten. Dieses führt dazu, dass teilweise ein mehr als zehnfacher Aufwand entsteht, um mit einer Wahrscheinlichkeit von 99% eine Lösung zu evolvieren. Für die Evolution eines Ausdrucks, der den Verlauf der Sinusfunktion annähert, lässt sich bezogen auf die Potenzreihenentwicklung festhalten, dass die evolvierten linearen Programme bis zu einem Summenglied besser sind als die entstehenden Lösungen in Form einer algorithmischen Chemie. Die Evolutionsläufe für die algorithmische Chemie stagnieren hier vergleichsweise häufig in lokalen Optima.

Das für die SPO der Systeme auf dem Parityproblem gewählte Optimierungskriterium (vgl. Kap. 8.2) hat beim Algorithmendesign des AC-Systems zu stark von den anderen Problemstellungen abweichenden Einstellungen geführt. Diese Einstellungen wiesen eine geringe Performance auf. Eine manuelle Anpassung anhand der Einstellungen, die für andere Problemstellungen ermittelt wurden, war möglich und hat deutlich bessere Ergebnisse gezeigt. Dieses lässt hoffen, dass sich dieses Algorithmendesign auch für neue Problemstellungen eignet.

Die Totalsynthese wurde eingeführt, um den Aufwand der SPO-Läufe für das AC-System zu reduzieren. Die für die AC-Varianten gefunden Parameter bezogen sich somit auf die Verwendung der Totalsynthese. Die Experimente haben gezeigt, dass die Verwendung dieser Parameter auch bei den AC-Varianten zum Erfolg führt, die auf die Verwendung der Totalsynthese verzichten.

Bei der Evolution künstlicher Chemien ohne Verwendung der Totalsynthese hängt der Erfolg maßgeblich davon ab, dass genügend Reaktionen ausgeführt wurden, um den Datenfluss der enthaltenen Lösung zu assemblieren. Die Anzahl der ausgeführten Reaktionen wird über das Vielfache c der im Individuum codierten Reaktionen bestimmt. Hier wurden zunächst die Vielfachen $c = 5$ und $c = 10$ gewählt. Die Wahl von $c = 5$ erwies sich für das Parity- und das Sinusproblem als deutlich zu gering. Für das Parityproblem wurde mit dieser Wahl keine Lösung gefunden, beim Sinusproblem stagnieren nahezu alle Läufe im lokalen Optimum. Lediglich beim Thyroidproblem gelang die Evolution guter Individuen. Kapitel 10.3 zeigt, dass der effektive Code der Individuen hier sehr klein ist, wodurch die Wahrscheinlichkeit erfolgreicher Assemblierung erhöht wird. Komplexe Lösungen enthalten eine höhere Anzahl von Abhängigkeiten, sprich Reaktionen, die in der richtigen Reihenfolge ausgeführt werden müssen, um nicht elastisch zu sein. Die Wahrscheinlichkeit, dass diese zumindest einmal in richtiger Reihenfolge ausgeführt wurden, erhöht sich mit der Anzahl ausgeführter Reaktionen. Die Reaktionszahl trägt damit zur „phänotypischen Stabilität“[1] bei. Der Zusammenhang zwischen Individuenhöhe, ausgeführter Reaktionen und der Assemblierungswahrscheinlichkeit wird in Kap. 12 noch genauer betrachtet.

[1] Dieser Begriff stammt von Lones [71], der die „phänotypische Stabilität“ bei Enzym-GP durch Anpassung der im Genom codierten Profile mittels Reinforcement Learning erhöht.

10. Bloat und effektiver Code

Die absolute und die effektive Größe der Individuen sowie ihre Höhe sind die drei Maße, die in diesem Abschnitt betrachtet werden. Die exponentielle Zunahme der absoluten Individuengröße ist ein unter der Bezeichnung Bloat bekannter Effekt in der genetischen Programmierung. Dieser tritt bei unterschiedlichen Repräsentationen auf. Bloat wird zunächst im Abschnitt 10.1 diskutiert und in Abschnitt 10.2 im Vergleich beider Systeme und ihrer Varianten betrachtet.

Es existieren unterschiedliche Hypothesen über die Ursache von Bloat, die im Folgenden noch genauer vorgestellt werden. Sollte es sich beim Bloat um einen Mechanismus zum Schutz vor destruktiven Veränderungen der Suchoperatoren handeln, so lässt das Modell von Nordin und Banzhaf [83] vermuten, dass nicht nur die absolute Größe der Individuen abnimmt, sondern auch die effektive Größe der Individuen zunimmt, sollte dieser Schutz überflüssig werden. Soule und Foster [108] beobachten die Zunahme effektiver Größe bei nicht destruktiven Suchoperatoren experimentell. Zhang und Mühlenbein [119] gehen davon aus, dass kleinere Lösungen tendenziell eine bessere Generalisierung darstellen. Daher kann eine Zunahme effektiver Größe unter Umständen als unerwünschter Effekt betrachtet werden, der ebenfalls gemessen wird.

Zusätzlich wird, als dritte Eigenschaft, die effektive Höhe des besten Individuums protokolliert. Teil der Motivation ist die Betrachtung von Sequenzialität als Dogma, welches seinen Ursprung in der von-Neumann-Architektur hat und welches nicht nur unser Verständnis von Algorithmen und viele Programmiersprachen beeinflusst hat, sondern auch die Repräsentation von Lösungen in der genetischen Programmierung. Die Ausbildung der Sequenzialität, die bei den meisten Repräsentationen explizit in der Struktur des Genoms codiert ist, wird durch das Fehlen einer solchen Struktur bei den algorithmischen Chemien erschwert und ist nur noch implizit über die Moleküle möglich.

Aus diesem Grund wird vermutet, dass die evolvierten Chemien eine zu starke Sequenzialität in der Datenverarbeitung meiden, da mit jeder Erhöhung der Anzahl sequenzieller Reaktionen die Wahrscheinlichkeit für die erfolgreiche Assemblierung des Datenflusses sinkt. Besitzt eine algorithmische Chemie n Reaktionen und wurde r_1 zum Zeitpunkt $t_1 \geq 1$ erfolgreich ausgeführt, so beträgt bei paralleler Ausführung von r_2 die Wahrscheinlichkeit, dass diese Reaktion zum Zeitpunkt $t_2 > t_1$ erfolgreich ausgeführt wurde:

$$1 - \left(\frac{n-1}{n}\right)^{t_2-1}.$$

Hängt eine Reaktion r_2 sequenziell von einer anderen Reaktion r_1 ab, weil r_1 den Zu-

stand eines der Edukte von r_2 bestimmt, so muss r_1 bereits ausgeführt worden sein, um r_2 erfolgreich (nicht elastisch) ausführen zu können. Die Wahrscheinlichkeit beträgt dann nur

$$1 - \left(\frac{n-1}{n}\right)^{t_2 - t_1}.$$

Je später r_1 ausgeführt wurde, desto geringer ist die Wahrscheinlichkeit, dass r_2 in der verbleibenden Zeit erfolgreich ausgeführt werden kann. Um das Ausmaß an Sequenzialität abzuschätzen, wird auch die effektive Höhe der besten Individuen in beiden Systemen verglichen.

10.1. Bloat – Grundlagen

Bei vielen Repräsentationen lässt sich im Laufe der Evolution eine Zunahme der Genomgröße beobachten, die aus Code resultiert, welcher das berechnete Ergebnis oder die Fitness der Individuen nicht beeinflusst. Dieser Code wird als *Introns* bezeichnet. Während codierenden Bereiche in der Biologie als *Exons* bezeichnet werden, stehen Introns für Bereiche im Genom eukaryontischer Zellen, die keine Proteine codieren und die vor der Translation entfernt werden. Banzhaf u. a. [17] definieren Introns in der genetischen Programmierung über zwei Eigenschaften:

- Introns entstehen durch die Evolution von Genotypen variabler Größe und
- ihre Existenz beeinflusst nicht direkt die Fitness des Individuums.

Bloat *Nimmt die Akkumulation von Introns exponentiell zu, dann wird dieser Prozess als Bloat bezeichnet.* Für diesen Prozess existiert in der Natur keine Analogie. Neben der Größenzunahme im Allgemeinen stellen Soule und Foster [108] für die Baumrepräsentation zudem fest, dass Bloat die Evolution hoher anstelle dichter Bäume begünstigt.

Für lineares GP unterscheiden Brameier und Banzhaf [25] zwischen strukturellen und semantischen Introns. Bei den *strukturellen Introns* handelt es sich um Instruktionen, die Variablen manipulieren, welche nicht in die Berechnung der Ausgabe eingehen. *Semantische Introns* sind Instruktionen, die den Wert einer Variablen nicht ändern, z. B. die Addition von 0 oder die Multiplikation mit 1. Da es beim linearen GP wesentlich einfacher ist, strukturelle Introns zu erzeugen, gehen sie davon aus, dass semantische Introns vergleichsweise selten vorkommen. Ihr Algorithmus ignoriert daher diese und ermittelt in einem Rückwärtslauf durch das Genom die strukturellen Introns. Für die lineare Variante des Systems wird dieser Algorithmus zur Intronbestimmung verwendet, bei der Evolution einer algorithmischen Chemie findet das in Kap. 6.4 beschriebene Gegenstück Verwendung. Auch dieses ignoriert die semantischen Introns. Abbildung 10.1 beschreibt ein solches Intron in einer algorithmischen Chemie, die eine Lösung für das 3 Bit Even-Parityproblem darstellt.

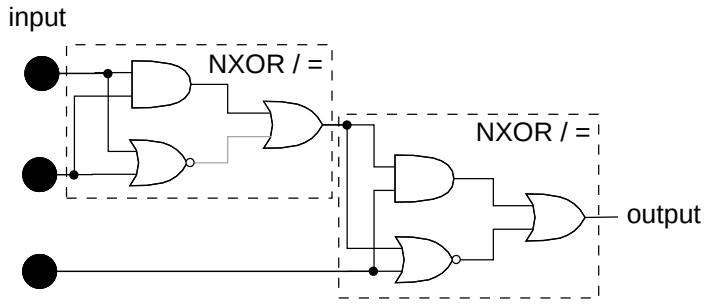


Abbildung 10.1.: Semantisches Intron in einer algorithmischen Chemie für das 3 Bit Even-Parity-Problem. Jede Verbindung zwischen 2 Gattern entspricht dem Datenaustausch über ein Molekül. Das Molekül der grau dargestellten Verbindung wird noch durch eine zweite hier nicht dargestellte Reaktion in seinem Zustand verändert. Die zweite Reaktion entspricht der Und-Verknüpfung dieses Zustands mit dem Wert „true“, dieses entspricht für jede Belegung dem vorherigen Zustand.

Im Folgenden werden die Begriffe der absoluten und der effektiven Größe verwendet. Bei der absoluten Größe handelt es sich um die Anzahl aller Instruktionen bzw. Reaktionen. Durch Entfernung der strukturellen Introns ergibt sich die Menge effektiver Instruktionen. Die Anzahl effektiver Instruktionen entspricht der effektiven Größe des Individuums (siehe auch Kap. 6.4).

Ursachen

Bzgl. der Ursache für die Akkumulation von Introns existieren unterschiedliche Hypothesen, die sich gegenseitig nicht ausschließen:

1. Die am häufigsten vertretene Hypothese vermutet die Ursachen in der fehlenden Homologität der Rekombinationsoperation, d. h. die Rekombinationsoperation ist zumeist nicht in der Lage, Codesegmente identischer Funktionalität zwischen den Eltern auszutauschen. Ausgetauschte Funktionalität wird im falschen Kontext platziert und ist dann zumeist wertlos. Den Nachkommen fehlt diese Funktionalität. Sie erhalten daher eine schlechtere Fitness. Lassen sich evolutionär nur schwer Verbesserungen erzielen, so erhöht Bloat die Wahrscheinlichkeit, dass Nachkommen wenigstens die Funktionalität und damit die Fitness ihrer Eltern aufweisen. Es haben jene Individuen einen Selektionsvorteil, die aus Eltern mit vielen Introns hervorgegangen sind. Bei der Rekombination dieser Eltern ist aufgrund der zumeist gleichverteilten Rekombinationspunkte die Wahrscheinlichkeit höher, dass Codesegmente innerhalb der Introns ausgetauscht werden und somit die Nachkommen in ihrer Funktion identisch mit den Eltern sind. Bloat erhöht die effektive Fitness. Hierdurch tendieren auch die Nachkommen zu einer größeren Anzahl Introns.

   Nordin und Banzhaf [83] zeigen empirisch, dass Rekombination zu Beginn der Evolution entweder neutrale oder eine negative Auswirkungen auf die

Fitness der Nachkommen hat und sich dieses in späteren Phasen zugunsten neutraler Auswirkungen ändert. Des Weiteren modellieren sie den Einfluss der Relation von absoluter und effektiver Größe auf den Erfolg der Rekombination. Diese Betrachtung erlaubt es ihnen, in dem Schutzmechanismus eine natürliche Tendenz zu einem kürzeren effektiven Code zu sehen, da auch eine Verkleinerung der effektiven Größe die Wahrscheinlichkeit für eine erfolgreiche Rekombination erhöht.

2. Soule und Foster [107, 109] stellen die Hypothese auf, dass die Entfernung kleiner Programmteile tendenziell eine geringere Wahrscheinlichkeit aufweist, effektiven Code zu tangieren, als die Entfernung großer Programmteile. Beim Hinzufügen von Programmteilen existiert eine solche Tendenz nicht. Diese einseitige Tendenz kann demnach zum Wachstum des Codes führen.

3. Langdon und Poli [68] machen die Verteilung der Lösungen für das Wachstum der absoluten Größe verantwortlich. Bei variabler Größe des Genoms und der Möglichkeit nicht effektiven Code zu erzeugen existieren für jede Lösung unendlich viele semantische gleiche Lösungen, die eine höhere Größe aufweisen, aber nur eine begrenzte Anzahl kleinerer semantisch gleicher Lösungen. Ist ein (lokales) Optimum erreicht, so wandelt sich die Suche in eine Zufallssuche nach semantisch gleichen Lösungen, die mit höherer Wahrscheinlichkeit in Form von größeren Lösungen gefunden werden. Vorhersagen auf der Grundlage dieser Hypothese sowie die Annahme, dass Introns zur Steigerung der effektiven Fitness dienen, bestätigten sich auch in einem repräsentationsfreien Modell von Banzhaf und Langdon [15].

Problematik

Das exponentielle Wachstum in den Phasen geringen oder fehlenden Fitnessfortschritts gilt es aus mehreren Gründen zu verhindern. Offensichtlich von Nachteil ist der gesteigerte Verbrauch von Speicher und CPU-Zeit. Trifft zudem die Schutzhypothese zu, so kann dieses auch zur Verhinderung von Veränderungen und evolutionären Fortschritt führen. Bloat gilt daher als einer der Gründe für die Stagnation von GP-Läufen.

Gegenmaßnahmen

Die häufigste Strategie, dem Bloat zu begegnen, ist die Begrenzung der Programmgröße. Bei der Evolution linearer Programme wird die Programmlänge, bei Baumstrukturen die Baumhöhe oder Knotenanzahl begrenzt. Dieses Vorgehen entspricht einer Begrenzung des Suchraums, wobei begründete Annahmen für eine solche Grenze zumeist nicht gemacht werden können.

Andere Ansätze ändern die Repräsentation und/oder den genetischen Operator. ADFs und andere Modularisierungstechniken schützen Teile des Genoms explizit durch Auslagerung in spezielle Bereiche. Die von Nordin u. a. [85] verwendeten

„explicit defined introns“ konzentrieren die Rekombination an definierten Punkten über hierfür eingeführte spezielle Knoten, welche gezielt als Rekombinationspunkt gewählt werden. Soule und Foster [108] verwenden die von Altenberg [3] vorgeschlagene nichtdestruktive Rekombination, um den Einfluss der destruktiven Eigenschaft dieser Operation auf das Codewachstum zu untersuchen. Bei ihrer Form der Rekombination werden Nachkommen nur in die nächste Generation übernommen, wenn sie mindestens die Fitness der Eltern erreichen. Andernfalls werden die Eltern wieder Teil der nächsten Generation. In ihren Experimenten findet zum einen kein exponentielles Wachstum statt, zum anderen ist die effektive Größe des Codes höher, was für die Annahme von Nordin und Banzhaf [83] spricht, dass der Schutz vor der destruktiven Wirkung der Rekombination sowohl zu einer höheren absoluten Größe, als auch zu einer kleineren effektiven Größe führt.

Blickle und Thiele [23] erlauben die Rekombination lediglich innerhalb des effektiven Codes und erhöhen so die Konvergenz in den meisten der von ihnen betrachteten Probleme. Hier, wie auch bei Nordin und Banzhaf [83], findet sich zudem eine Modellierung des Verhaltens von GP-Systemen in Abhängigkeit des Verhältnisses zwischen der effektiven und der absoluten Größe des Codes im Genom des Individuums.

Eine weitere Gruppe von Ansätzen betrachten die Evolution kleiner Programme. Diese Verfahren arbeiten zumeist einkriteriell, indem sie die Größe des Individuums als Strafterm in seine Fitness eingehen lassen. Dabei kann die Berechnung des Strafterms statisch oder dynamisch [119] sein. Ein zu hoher Strafterm birgt die Gefahr, frühzeitig in lokale Optima zu konvergieren, jene Optima, die mittels kleiner Individuen erreicht werden können (siehe Soule und Foster [110]).

Neuere, multikriterielle Ansätze [22, 37] betrachten die Evolution kleiner Individuen als von der Fitness unabhängige Zielsetzung und versuchen eine paretooptimale Menge von Individuen zu finden. Dies sind Individuen, bei denen kein kleineres Individuum existiert ohne die Fitness zu verschlechtern und umgekehrt. Es bleibt dann dem Benutzer in einem späteren Schritt überlassen, nach subjektiven Kriterien eine Lösung aus der Menge der Individuen zu wählen. Es hat sich dabei gezeigt, dass dieses Vorgehen auch die Erfolgsrate für die Lösung eines Problems erhöhen kann.

Bloatfreie Repräsentationen

Während Bloat für die beiden populärsten Repräsentationen, Baum-GP und lineares GP, ein häufig beobachtetes Problem darstellt, scheinen andere Repräsentationen davon nicht betroffen zu sein. Einige von ihnen wurden in Kap. 1 bereits vorgestellt. Lones und Tyrrell [69, 71] berichten für das Enzym-GP System (Kap. 1.2.2) von einem weitestgehend konstanten Verlauf der durchschnittlichen Individuengröße, bzw. einem moderaten Wachstum bei zu geringer Anfangsgröße. Auch Salustowicz und Schmidhuber heben für PIPE (Kap. 1.3.1) hervor, dass ihr Ansatz im Vergleich zu anderen GP-Varianten zu kleineren Individuen tendiert. Gleiches gilt für die eCGP-Variante (Kap. 1.3.2) dieses Ansatzes von Sastry und Goldberg [102], die den

Ansatz mit einer Heuristik koppeln, die in jeder Generation versucht, das Modell zu verkleinern, aus dem die Population entnommen wird. Es handelt sich bei diesen beiden Verfahren um EDAs. Der ihnen als Modell zugrundeliegende Prototypbaum kann zwar wachsen, eine Rekombination, die zu einer abweichenden Größe in den Nachkommen führt, findet aber nicht statt. Zudem ist Bloat bei ihnen nur in Form semantischer Introns möglich. Auch bei den probabilistischen Grammatiken (Kap. 1.3.3) von Ratle und Sebag [97] handelt es sich um einen EDA. Wachstum ist hier nur durch eine Transformation der Grammatik möglich.

Kontext einer Funktionalität

Der Kontext einer evolvierten Funktionalität ist ein wichtiger Teil der evolvierten Gesamtinformation. Lones und Tyrrell [69] äußern die Hypothese, dass der Verlust des Kontexts bei der Rekombination verantwortlich dafür ist, dass selten gute Nachkommen erzeugt werden. Zusammen mit der Hypothese, dass Bloat zum Schutz vor den genetischen Operatoren selektiert wird, lässt sich eine zusammengesetzte Hypothese aufstellen.

Hypothese 10.1 Eine Repräsentation in der genetischen Programmierung, deren Operationen den Kontext evolvierter Funktionalität bewahren, ermöglicht eine weitgehend bloatfreie Evolution.

Die bloatfreien Repräsentationen im vorherigen Abschnitt berücksichtigen den Kontext. Bei den EDA-Ansätzen werden allerdings die Operationen aller Individuen samt Kontext in einem gemeinsamen Modell gespeichert, Individuen werden nur aus diesem Modell heraus erzeugt. Beim Enzym-GP hingegen besitzt jedes Individuum ein eigenes Genom. Die darin codierten unterschiedlichen Enzymarten besitzen auch Informationen über die gewünschte Funktionalität, welche sie als Input verwenden. Eine Beschreibung der bereitgestellten Funktionalität wird errechnet. Diese beiden Informationen helfen, die Enzyme im richtigen Kontext zu verwenden.

Bei den algorithmischen Chemien beschreiben die verwendeten Moleküle den Kontext, in dem eine Reaktion stattfindet. Eine funktionelle Abhängigkeit muss durch die Verwendung entsprechender Moleküle als Edukte und Produkt aufgelöst werden. Eine Reaktion gelangt immer als Ganzes von dem einen Elter in das andere, d. h. mitsamt den Informationen über den ursprünglichen Kontext. Werden die Edukte einer Reaktion in dem einen Elter mit den gleichen Kontexten assoziiert wie im anderen Elter, so wird die Reaktion automatisch eingegliedert, ansonsten stellt das Reaktionsprodukt im Nachkommen eine neue Funktionalität zur Verfügung. Mit dieser Sichtweise und der oben genannten Hypothese 10.1 lässt sich vermuten, dass die Evolution einer algorithmischen Chemie nicht zu Bloat tendiert. Dieses wird im Folgenden betrachtet und mit der entsprechenden Evolution eines linearen Individuums verglichen.

Die von der SPO gering gewählte Rekombinationsrate kann unter dieser Hypothese eine Hinweis darauf sein, dass der Rekombinationsoperator nicht geeignet ist,

Tabelle 10.1.: Pro Lauf ausgeführte Instruktionen, Instruktionen pro Fallbeispiel und durchschnittliche Größe der Individuen am Ende der Evolution, jeweils gemittelt über alle Läufe.

Verfahren	$\sum$ Inst. $(\times 10^8)$	Inst./Fallbsp.	Größe
Sinus			
LGP	10.7	35.53	315.88
AC, ts.	1.7	5.51	24.44
AC, $c = 5$	51.9	172.99	37.72
AC, $c = 10$	90.4	301.08	34.33
Thyroid			
LGP	6.1	3.07	290.63
AC, ts.	4.7	2.35	23.39
AC, $c = 5$	337.6	168.81	40.62
AC, $c = 10$	508.7	254.31	25.03
Parity			
LGP	13.1	81.75	401.58
AC, ts.	25.1	19.56	24.49
AC, $c = 5$	187.0	146.04	36.00
AC, $c = 10$	360.5	281.50	32.51

die erforderlichen Kontexte in den Nachkommen herzustellen. Diese Vermutung führt u.a. in Kap. 11 zu einer Überarbeitung des Rekombinationsoperators.

10.2. Absolute Individuengröße

Zunächst wird der Verlauf der absoluten Größe der Individuen im Populationsdurchschnitt über die Zeit betrachtet. Dieser ist in Abb. 10.2 für die drei Problemstellungen dargestellt. Die letzte Spalte in Tab. 10.1 listet die durchschnittliche absolute Größe am Ende der Evolution auf. Alle Individuen wurden zu Beginn mit 25 Instruktionen bzw. Reaktionen initialisiert, die maximal erlaubte Größe beträgt 500. Der Rekombinationsoperator stellt sicher, dass dieser Wert nicht überschritten wird. Beim LGP-System steigt die durchschnittliche absolute Größe auf ca. 300 Instruktionen für das Sinus- und das Thyroidproblem und für das Parityproblem sogar auf ca. 400 Instruktionen. Der Bloat ist deutlich zu erkennen. Die Größe einer algorithmischen Chemie hingegen beträgt im Populationsdurchschnitt für die betrachteten Problemstellungen unter 41 Reaktionen. Bei den Varianten mit der Totalsynthese sind die Durchschnittswerte durchgängig sogar leicht unter dem Initialwert. Die absolute Größe der AC-Individuen ändert sich somit kaum.

Diese Beobachtung stehen im Einklang mit der aufgrund der Hypothese 10.1 geäußerten Erwartungen, dass die Evolution algorithmischer Chemien nicht zu Bloat

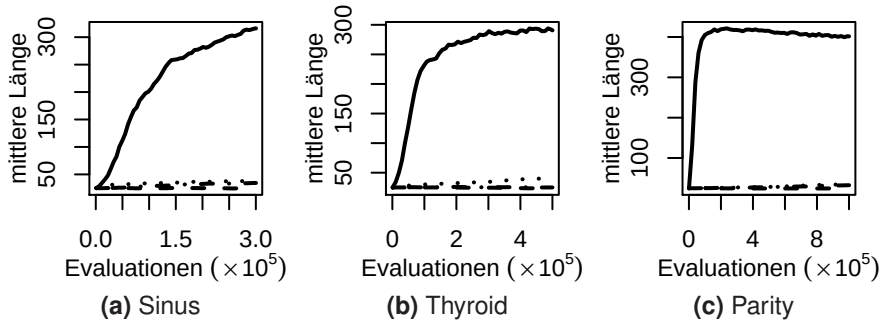


Abbildung 10.2.: Absolute Größe der Individuen im Populationsdurchschnitt, gemittelt über alle Läufe. Die durchgehende Linie entspricht dem Verhalten beim LGP-System, die anderen Linien stehen für die unterschiedlichen AC-Varianten und werden aufgrund ihrer Ähnlichkeit nicht weiter differenziert.

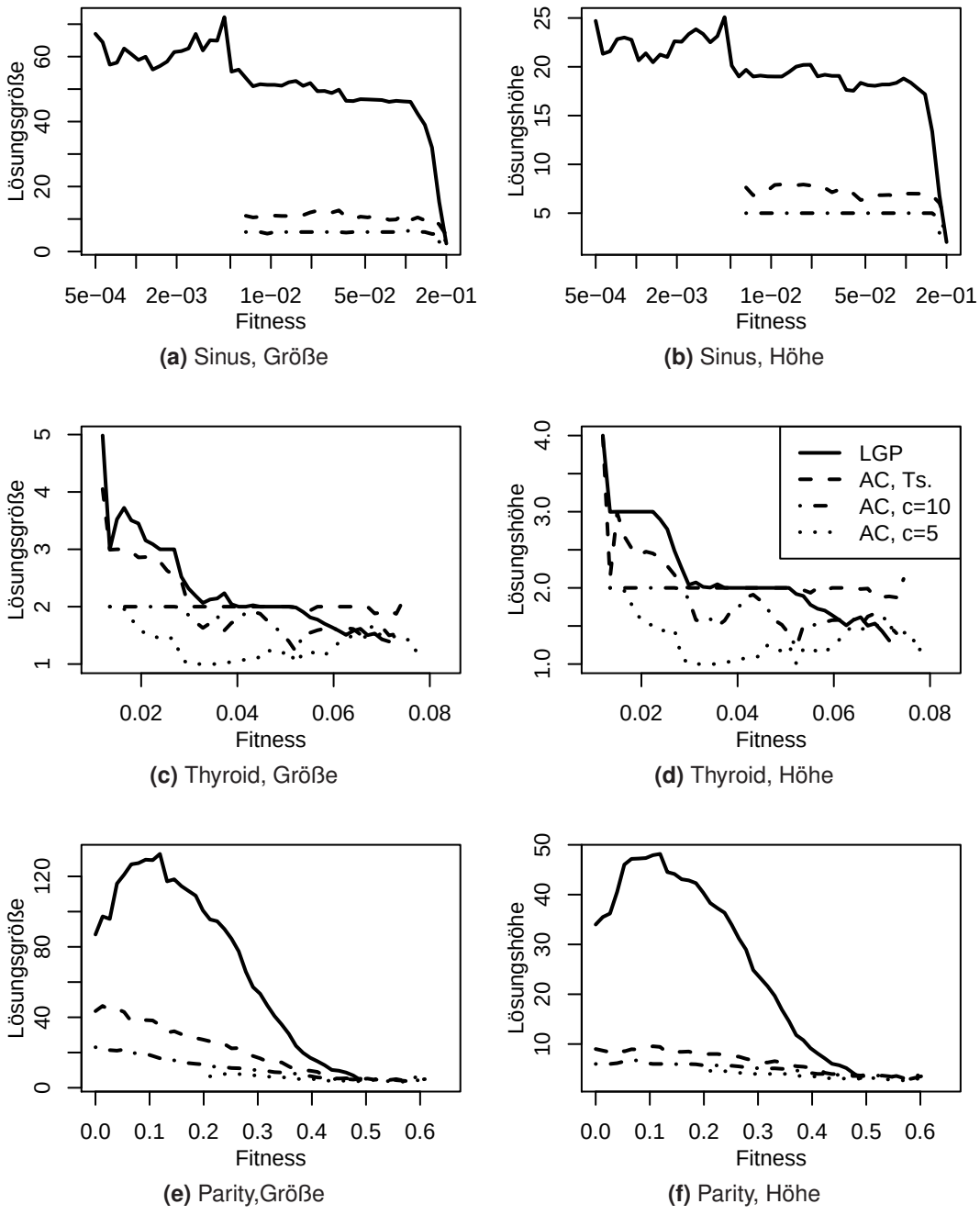
tendiert. Die Evolution algorithmischer Chemien ist dabei aber in ihrer Möglichkeit zur Bloatbildung aufgrund geringer Rekombinationsraten eingeschränkt. Die AC-Läufe würden trotzdem eine deutlich höhere absolute Größe aufweisen, wenn durch die Größe ein Selektionsvorteil entstünde. Die geringe Größenzunahme steht auch im Widerspruch zu der eigenen anfänglichen Erwartung an die Dynamik der Evolution. Diese Erwartung gingen von einer Akkumulation benötigter Reaktionen aus und beeinflusste auch die Umsetzung der Rekombination.

10.3. Effektive Individuengröße und -höhe

Abbildung 10.3 beschreibt die Größe und die Höhe des effektiven Codes des besten Individuums in Abhängigkeit des Evolutionsfortschritts (Fitness) für die drei Problemstellungen. Da nicht jeder Lauf jeden Fitnesswert annimmt, wurde ein System in einem Bereich nur dann berücksichtigt, wenn mindestens 10 Läufe diesen oder einen besseren Fitnesswert erreicht haben. Zwischen den Messpunkten wurde linear interpoliert. Die Abbildungen 10.3 (a, c & e) stellen den Median der effektiven Größe des besten Individuums dar. Die Abbildungen 10.3 (b, d & f) zeigen den Median der Höhe des effektiven Codes des besten Individuums in Abhängigkeit der Fitness.

Verlauf der effektiven Größe

Das evolvierte lineare Individuum bestehen für alle Probleme aus deutlich mehr effektiven Instruktionen als bei den gefundenen algorithmischen Chemien an Reaktionen ausgeführt werden müssen.



(a) Sinus, Größe **(b)** Sinus, Höhe

(c) Thyroid, Größe **(d)** Thyroid, Höhe

(e) Parity,Größe **(f)** Parity, Höhe

Abbildung 10.3.: Eigenschaften des effektiven Codes des besten Individuums in Abhängigkeit seiner Fitness. Die Werte stellen den Median von mindestens zehn Läufen dar, geringe Fitnesswerte stehen für eine bessere Eignung des Individuums.

Parityproblem

Beim LGP-System auf dem Parityproblem steigt die Größe der gefunden Individuen kontinuierlich an, nimmt dann zur perfekten Lösung hin aber wieder ab (Abb. 10.3e). Ein Grund hierfür könnte die Hypothese seien, dass auch durch Verringerung der effektiven Größe die effektive Fitness erhöht werden kann. Diese Reduzierung der effektiven Größe findet, wie auch die Erhöhung der absoluten Größe durch Bloat, demnach in Phasen der Stagnation statt. Mit Erreichen eines (lokalen) Optimums treten die Läufe in eine solche Phase ein. Die Abbildung 10.2c zeigt, dass die Möglichkeit zur Steigerung der effektiven Fitness mittels Bloat nach einem Fünftel der zur Verfügung stehenden Evaluationen erschöpft ist, sodass die effektive Fitness nur durch Abnahme der effektiven Größe gesteigert werden kann.

Ein deutlich langsamerer Anstieg der effektiven Größe findet für die Variante der AC mit Totalsynthese statt, eine vollständige Lösung besteht bei ihr im Median aus ca. 50 Reaktionen. In Kapitel 9.4 wurde bereits beobachtet, dass die Evolution hier intensiv davon Gebrauch macht, dass bei einem Zykluseintritt der Wert 0 garantiert wird. Ein Teil der Reaktion könnte somit noch der gezielten Nutzung dieses Effekts dienen. Die Unbestimmtheit, mit der ein Molekül durch eine Reaktion mit einem Wert belegt wird, gegen 0. Dieses bedeutet, dass die Belegung der Moleküle bei hinreichend großer Anzahl ausgeführter Reaktionen nahezu eindeutig ist. Eine Mehrfachzählung der Reaktionen in unterschiedlichen Zweigen des Datenflussbaums ist damit unnötig. Sie führt dazu, dass die Größe höher als nötig bewertet wird. Die Größe lässt sich in diesen Fällen mit der Anzahl zustandsveränderlicher Moleküle abschätzen (hier: 24). Bei den beiden AC-Varianten ohne Totalsynthese steigt die effektive Größe des besten Individuums noch langsamer an. Die Variante mit dem Vielfachen $c = 5$ erreicht nur selten die besseren Fitnessbereiche und wird daher in diesem Bereich nicht mehr dargestellt.

Sinusproblem

Bei der Evolution einer Lösung für das Sinusproblem (vgl. Abb. 10.3a) fällt unter Verwendung der linearen Repräsentation der nahezu sprunghafte Anstieg der effektiven Größe im Bereich des lokalen Optimums auf. Nach diesem Anstieg findet nur noch ein langsames Wachstum der effektiven Größe statt. Auch bei den AC-Varianten ist eine Größenzunahme in diesem Bereich zu beobachten, allerdings fällt die Steigerung der Größe hier wesentlich geringer aus.

Thyroidproblem

Die Betrachtung der Lösungen für das Thyroidproblem in Abb. 10.3c zeigt zunächst, dass die effektive Größe der Individuen bei allen betrachteten Systemen gering ausfällt. Die aus der Literatur bekannten Werte lassen einen Bedarf an mehr Instruktionen bzw. Reaktionen vermuten. So besteht eine von Gathercole [48], S. 127 abgebildetes Individuum eines Baum-GP Systems aus 87 inneren Knoten. Die Höhe des Baums beträgt 16 Knoten. Die von Schiffmann u. a. [103] evolvierten neuronalen Netze bestehen aus 48-50 Neuronen, die über 276-286 gewichte Kanten miteinander verbunden sind. Die geringe benötigte Größe dürfte mit ein Grund

dafür sein, dass auch bei einem relativ geringen Vielfachen von $c = 5$ gute algorithmische Chemien mit einem Klassifikationsfehler von nahezu 1% gefunden werden. Im Bereich sehr guter Lösungen ist noch einmal ein Anstieg der effektiven Größe zu beobachten. Die Verbesserung umfasst nur wenige der betrachteten Fallbeispiele, sodass möglicherweise die Vergrößerung der Individuen zugunsten einer Spezialisierung stattfindet.

Aufwand

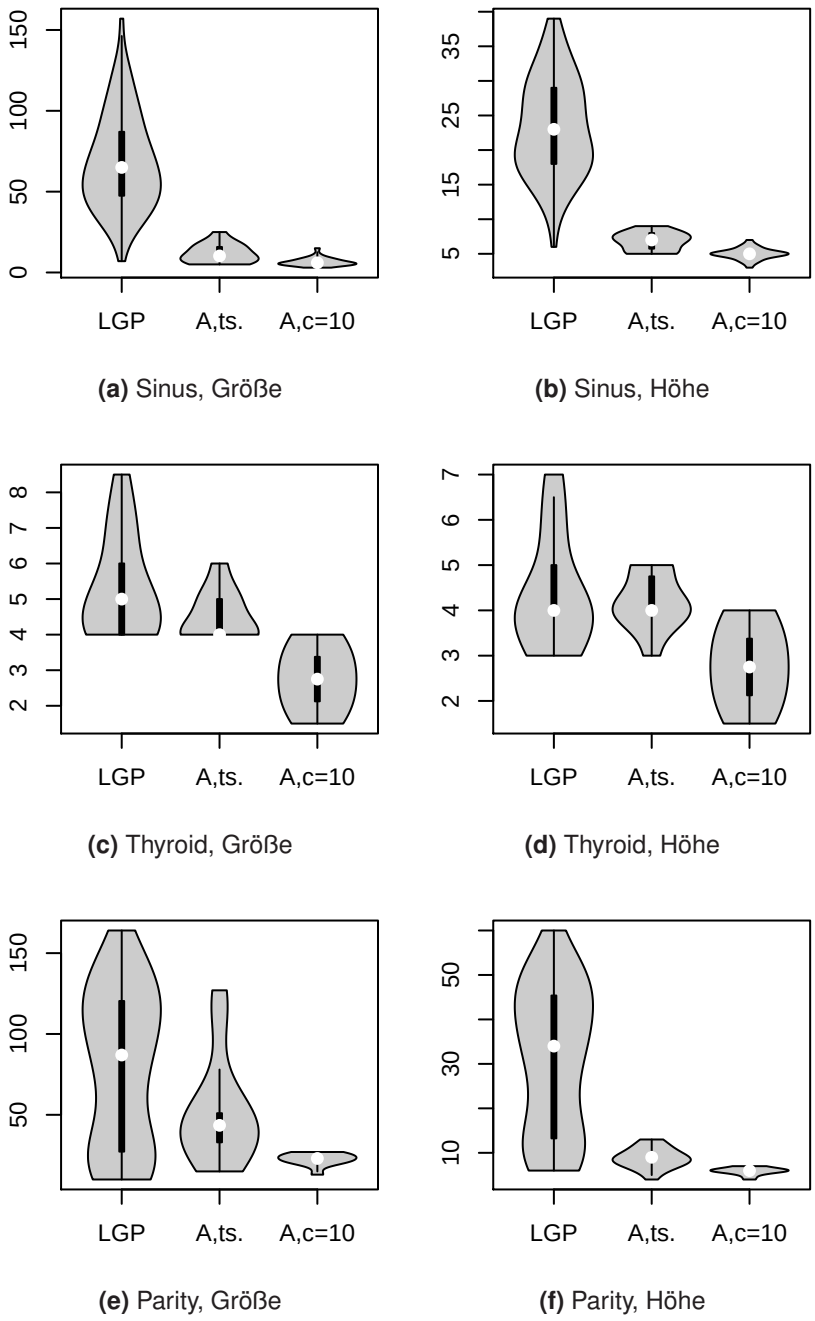
Die geringe effektive Größe der evolvierten ACs macht sich bei der Variante mit Totalsynthese positiv im entstandenen Gesamtaufwand bemerkbar. Tabelle 10.1 listet in der zweiten Spalte den durchschnittlichen Gesamtaufwand für einen Evolutionslauf, gemessen an der Anzahl ausgeführter Reaktionen bzw. Instruktionen. Bei den beiden AC-Varianten ohne Totalsynthese macht sich die geringere effektive Größe nicht durch eine Verringerung des Aufwands bemerkbar, da hier ein Vielfaches c der absoluten Größe an Reaktionen zufällig gewählt und ausgeführt wird. Dieses führt zu einem deutlich höheren Mehraufwand, der sich aber vergleichsweise einfach parallelisieren lässt, da bei dieser Form der Ausführung keine Abhängigkeiten zwischen den Reaktionen beachtet werden. Es können folglich zu einem Zeitpunkt mehrere Reaktionen unabhängig voneinander gewählt und ausgeführt werden. Die absolute Zahl der Reaktionen und Instruktionen berücksichtigt bereits, dass die algorithmischen Chemien zum Teil eine deutlich größere Anzahl an Fallbeispielen auswerten, um so das mit der nichtdeterministischen Auswertung verbundene Rauschen zu reduzieren (Kap. 8.2.1). Die dritte Spalte in Tab. 10.1 trägt diesem Umstand durch die Auflistung der pro Fallbeispiel benötigten Reaktionen/Instruktionen Rechnung.

Verlauf der effektiven Höhe

Die Höhe des Datenflusses der Individuen, dargestellt in den Abbildungen 10.3 (b, d & f), folgt beim LGP-System qualitativ dem Verlauf der Größe. Bei den AC-Varianten ist die Höhe nach einer anfänglichen Wachstumsphase weitestgehend konstant. Des Weiteren fällt die Höhe bei den algorithmischen Chemien, die ohne Verwendung der Totalsynthese evolviert wurden, geringer aus, als mit Totalsynthese oder bei den mittels LGP evolvierten Individuen. Während bei der Totalsynthese die Assemblierung einer eindeutigen (zyklenfreien) Lösung beliebiger Höhe gewährleistet ist, nimmt bei der rein randomisierten Ausführung die Wahrscheinlichkeit für die vollständige Assemblierung mit der Höhe ab. Kapitel 12 wird auf diesen Aspekt genauer eingehen.

Effektive Größe und Höhe guter Individuen

Die bisherigen Betrachtungen galten dem Verhalten im Median in Abhängigkeit der Fitness. Nun wird die Verteilung dieser Eigenschaft für „gute“ Individuen betrachtet. Dieses sind beim Parityproblem jene Individuen, die die Parität aller



(a) Sinus, Größe

(b) Sinus, Höhe

(c) Thyroid, Größe

(d) Thyroid, Höhe

(e) Parity, Größe

(f) Parity, Höhe

Abbildung 10.4.: Verteilung von Größe und Höhe der gefundenen Individuen bei vorgegebener Fitness.

Tabelle 10.2.: Größe und Höhe der, von den unterschiedlichen Systemen gefundenen, Individuen im Median für die betrachteten Aufgabenstellungen.

	Anz.	Größe	Höhe
Sinus			
LGP	73	65.00	23.00
AC, Ts.	16	10.25	7.00
AC, $c = 10$	26	6.00	5.00
Thyroid			
LGP	21	5.00	4.00
AC, Ts.	10	4.00	4.00
AC, $c = 10$	6	2.75	2.75
Parity			
LGP	66	87.00	34.00
AC, Ts.	21	43.50	9.00
AC, $c = 10$	11	23.00	6.00

Tabelle 10.3.: p-Werte der paarweisen Wilcoxon-Rangsummentests auf den Lösungsgrößen und -höhen. Die p-Werte wurden nach Holm adjustiert.

	Größe		Höhe	
	LGP	AC, TS.	LGP	AC, TS.
Sinus				
AC, Ts.	< 0.001	–	< 0.001	–
AC, $c = 10$	< 0.001	0.001	< 0.001	< 0.001
Thyroid				
AC, Ts.	0.149	–	0.849	–
AC, $c = 10$	< 0.001	0.002	0.009	0.008
Parity				
AC, Ts.	0.054	–	< 0.001	–
AC, $c = 10$	0.003	0.001	< 0.001	< 0.001

möglichen Eingabekombinationen korrekt bestimmen (Fitness gleich 0). Beim Thyroidproblem sind dies Individuen mit einem Klassifikationsfehler von 1.1%-1.2% und bei dem Sinusproblem werden hier die Individuen als gut bezeichnet, die einen mittleren quadratischen Fehler zwischen $n_{\max} = 2$ und $n_{\max} = 3$ (vgl. Tab. 8.2) aufweisen.

Abbildung 10.4 zeigt die Verteilung von effektiver Größe und Höhe der Individuen in diesem Bereich. Ein Lauf, der in dem jeweiligen Bereich in mehreren Generationen ein Individuum hervorbringt, wird jeweils durch den Median aller entsprechenden Individuen vertreten. Tabelle 10.2 enthält neben den Medianen auch die Anzahl an Läufen, die mit dem jeweiligen Verfahren Individuen in den entsprechenden Bereichen hervorgebracht haben. Für das Thyroidproblem wurde die Evolution einer algorithmischen Chemie mit $c = 10$ in die Betrachtung einbezogen, obwohl hier nur in 6 Experimenten ein entsprechendes Individuum gefunden wurde, um diese Variante der AC-Evolution betrachten zu können. Der Median zeigt zum Teil deutlich, dass der Datenfluss bei der Evolution einer algorithmischen Chemie aus weniger Reaktionen besteht, als Instruktionen in den gefundenen linearen Programmen verwendet werden. Wenngleich die Verteilung der Individueneigenschaften zeigt, dass auch bei der LGP Individuen geringer Größe und Höhe gefunden werden können, so werden doch vermehrt größere und höhere Individuen gefunden. Tabelle 10.3 enthält die p-Werte der paarweisen Wilcoxon-Rangsummentests auf den gemessenen Werten. Dabei zeigt sich, dass die Hypothese auf Gleichheit der Systeme zumeist zum gewählten Signifikanzniveau $\alpha = 0.05$ verworfen werden kann. Eine Ausnahme bilden die Läufe für das Thyroidproblem, bei dem von allen Systemen gute Individuen geringer Größe und Höhe gefunden werden.

Visualisierung

Die Abbildungen 10.5 und 10.6 visualisieren exemplarisch in den Individuen kodierte Datenflüsse und damit den effektiven Code der Individuen (vgl. Kap. 6.5). Die dargestellten Individuen stammen aus Evolutionsläufen für das Parityproblem. Abbildung 10.5 stellt Beispiele aus dem LGP-System dar. Anhand großer und hoher Individuen wird deutlich, dass sich eine Analyse solcher Individuen als aufwendig erweisen kann (Abb. 10.5a und 10.5b). Wenngleich das LGP-System eine Tendenz zu solchen Individuen aufweist, werden in den LGP-Läufen auch kleinere Lösungen gefunden (Abb. 10.5c). Die Abbildung 10.6 enthält unterschiedliche Individuen aus dem AC-System. Das Ergebnis des Individuums in Abb. 10.6a hängt von der Häufigkeit ab, mit der die einzig effektive Reaktion während der Ausführung aufgerufen wird. Bei jedem Aufruf der Reaktion wechselt der Zustand zwischen den beiden booleschen Werten. Ein solches Individuum erhält im Schnitt eine Fitness von 0.5 (vgl. auch Kap. 8.2.1). Abbildung 10.6b zeigt ein Individuum mit einer Fitness von 0.148. Seine Chemie enthält zyklische Zugriffe auf Molekülzustände und Moleküle, deren Zustand durch mehr als eine Reaktion verändert wird. Zyklen bleiben bei Verwendung der Totalsynthese auch in den vollständigen Lösungen

zumeist erhalten (Abb. 10.6c). Ohne Totalsynthese werden die Zyklen wie auch die Unbestimmtheit in der Regel im Laufe der Evolution reduziert (Abb. 10.6d).

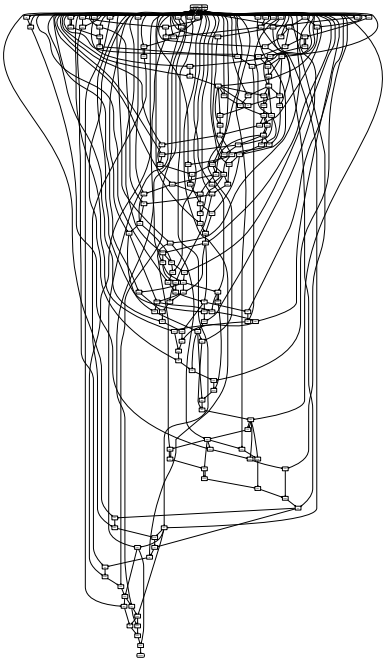
Zusammenfassung und Diskussion

Nach einer Einführung in das Thema Bloat und Introns, den Hypothesen über ihren Ursprung, den aus der Literatur bekannten Gegenmaßnahmen, sowie der Auflistung einiger bloatfreier Repräsentationen wurde die Entwicklung der absoluten Größe der Individuen für die hier untersuchten Systeme auf den drei verschiedenen Problemstellungen betrachtet. Während die Evolution linearer Programme erwartungsgemäß ein Bloatverhalten zeigte, wies die Evolution einer algorithmischen Chemie eine solche Tendenz nicht auf.

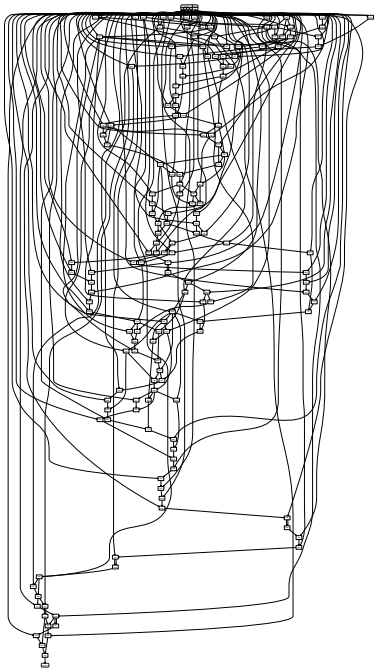
Der Berechnungsaufwand für die Ausführung wird durch die effektive Größe des Individuums bestimmt. Die Betrachtungen der effektiven Größe ergibt, dass die evolvierten algorithmischen Chemien aus weniger Reaktionen bestehen als Instruktionen in vergleichbar guten linearen Programmen benötigt werden. Für das LGP-System und bei der algorithmischen Chemie mit Totalsynthese lässt sich der Aufwand auch während der Evolution auf die Ausführung der effektiven Instruktionen bzw. Reaktionen reduzieren. Für die anderen Varianten der algorithmischen Chemie ist eine Reduktion auf die effektiven Instruktionen nicht vorgesehen. Die Totalsynthese soll für sie diese Aufgabe nur während der Parameteroptimierung erfüllen. Während der Evolution ohne Totalsynthese soll der Einfluss von Individuengröße und -höhe im Zusammenspiel mit der Anzahl ausgeführter Reaktionen auf den Assemblierungserfolg erhalten bleiben. Nach der Evolution ist auch bei ihnen die Reduzierung des durchschnittlichen Aufwands auf die hier angegebene effektive Größe möglich.

Neben einem geringen Aufwand und einer möglicherweise besseren Generalisierung kann eine geringere effektive Größe noch weiter Vorteil haben. So verwendet die oben bereits aufgeführte Baum-GP Lösung für das Thyroidproblem von Gathercole [48] alle reellwertigen sowie 13 der 15 boolesche Eingaben. Auch wenn dieses hier nicht explizit gemessen und betrachtet wurde, so kann aufgrund der gegebenen Größe davon ausgegangen werden, dass weit weniger Daten benötigt wurden. Da die Werte bei diesem Problem durch medizinische Untersuchungen erhoben werden müssen, reduziert ein geringerer Bedarf an Eingaben unter Umständen den finanziellen Aufwand zur Ermittlung der Werte. Tatsächlich dürfte auch die Menge der benötigten Eingaben für die Baum-GP Lösung geringer sein, da die Repräsentation als Baum ausschließlich semantische Introns ermöglicht, sodass Teile der Eingabe vermutlich im Bereich der Introns verwendet werden und somit nicht ermittelt werden müssten. Auch die hier ermittelten linearen Individuen sind deutlich kleiner.

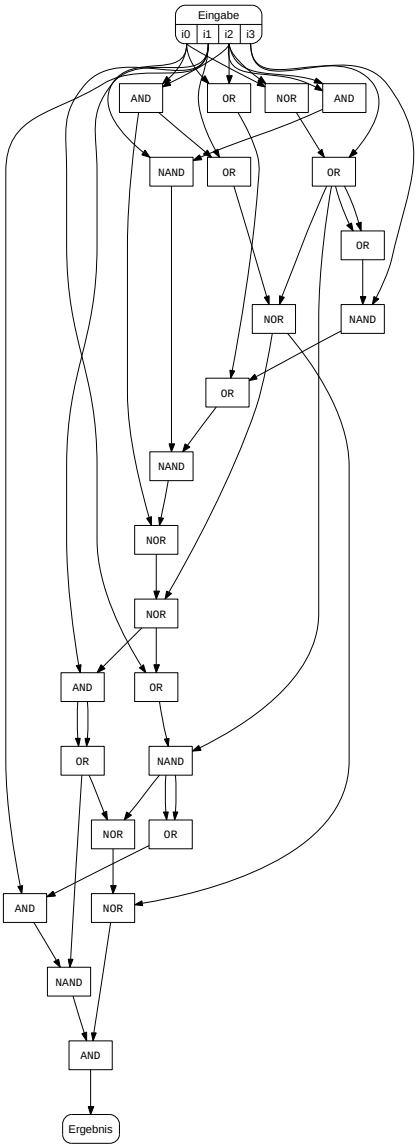
Der Verlauf der Höhe folgt qualitativ dem der Größe. Die Vermutung bestätigt sich, dass die Evolution zu einer Chemie geringer Höhe tendiert. Eine Reduzierung sequenzieller Abhängigkeit erhöht die Wahrscheinlichkeit, dass der Datenfluss



(a) Fitness $= 0.0625$

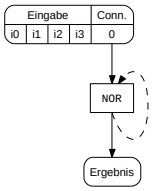


(b) Fitness $= 0$

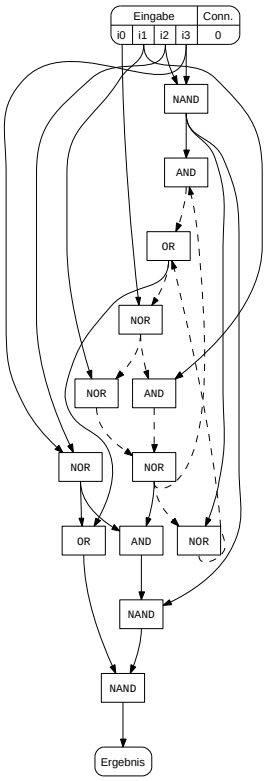


(c) Fitness $= 0$

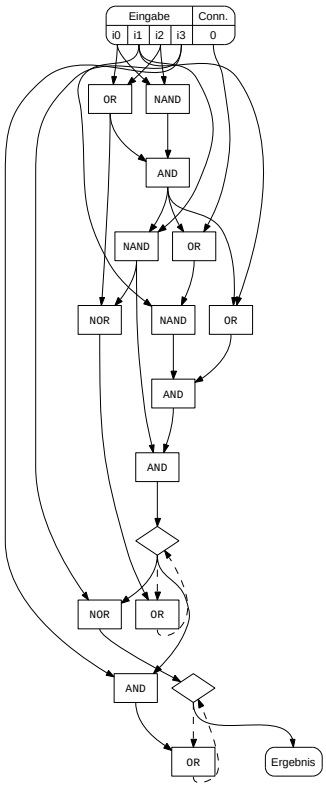
Abbildung 10.5.: Individuen des LGP-Systems für das Parityproblem.



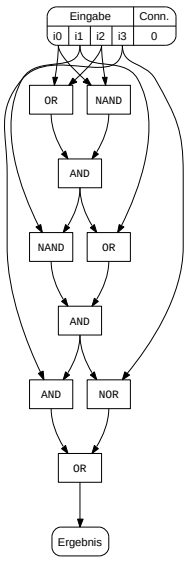
(a) $c = 10$, Fitness $= 0.5$



(c) Totalsynthese, Fitness $= 0$



(b) $c = 10$, Fitness $= 0.148$



(d) $c = 10$, Fitness $= 0$

Abbildung 10.6.: Individuen des AC-Systems für das Parityproblem.

der enthaltenen Lösung vollständig assembliert werden kann. Der praktische Vorteil von Individuen geringer Höhe liegt darin, dass ihre Laufzeit bei idealer Parallelisierung geringer ausfällt. Auch in alternativen Rechnerarchitekturen, in denen sich die Komponenten vor Beginn der Berechnung organisieren müssen, kann die Organisation schneller geschehen, da diese meist schrittweise und bottom-up erfolgt (vgl. Kap. 3.2).

Teil V.

Erweiterungen

11. Homologe Rekombination

Bei vielen evolutionäre Algorithmen wie auch bei der genetischen Programmierung wird die Rekombination als primärer Suchoperator betrachtete. Dieses geht zurück auf die Building Block Hypothese, die annimmt, dass in den Eltern codierte unterschiedliche Teillösungen bei der Rekombination zu besseren Lösungen kombiniert werden. Der Mutation wird dann häufig nur die Funktion zugestanden, verloren gegangene Information im Genpool der Population, z. B. bestimmte Terminale, neu zu generieren.

Die für alle betrachteten Anwendungen durchgeführte sequenzielle Parameteroptimierung hat für die Evolution einer algorithmischen Chemie ergeben, dass Nachkommen nur mit einer geringen Wahrscheinlichkeit (0.1%–2.7%) aus der Rekombination hervorgehen sollten. Dieses entspricht nahezu einem Verzicht auf den Rekombinationsoperator und könnte dann als kritisch angesehen werden, wenn die Rekombination ihrer zentralen Bedeutung gerecht würde. Zunächst werden daher einige Arbeiten betrachtet, die die Bedeutung der Rekombination bei der GP genauer untersuchen. Anschließend wird die Evolution algorithmischer Chemien ohne Rekombination wiederholt, um zu entscheiden, ob es sich bei den geringen Rekombinationsraten um einen Verzicht auf Rekombination oder um das notwendige Maß an Rekombination handelt. In einem weiteren Schritt wird dann die homologe Rekombinationsoperation eingeführt und empirisch untersucht.

11.1. Rekombination in der genetischen Programmierung

Die Bedeutung der Rekombination für die genetische Programmierung ist nicht unumstritten. O'Reilly und Oppacher [88] versuchen in ihrer GP Building Block Hypothese die Entstehung besserer Lösungen über die Kombination guter Teillösungen zu erklären. Dieses gelingt nur mit Annahmen, die von den Autoren als fragwürdig eingestuft werden, sodass sie Zweifel daran äußern, dass Erfolge der GP über die parallele Akkumulation von Funktionalität und ihrer hierarchischen Kombination erklärt werden können. Empirische Untersuchungen haben ihnen zusätzlich gezeigt, dass rein mutationsbasierte Ansätze, wie Hillclimbing oder Simulated Annealing, bei der Erzeugung von Programmen zu vergleichbaren Ergebnissen führen.

Die Building Block Hypothese hat ihren Ursprung bei den genetischen Algorithmen. Angeline [4] stellt fest, dass dort die Semantik der Building Blocks gebunden ist an die Semantik ihrer Position im Genom. Bei der genetischen Programmierung sind die bei der Rekombination ausgetauschten Einheiten nicht an ihre Position

gebunden, ihre Semantik steht damit erst mit ihrer Position fest. Das Ignorieren dieser Positionssemantik ist das charakteristische Merkmal der genetischen Programmierung. Der Teilbaumrekombination spricht Angeline daher die Fähigkeit zum Austausch und zur Erzeugung von Building Blocks ab. Eine solche Form der Rekombination ist seiner Meinung nach besser durch eine Makromutation beschrieben, die durch den Code der anderen Individuen beschränkt ist. Um dieses zu untermauern, verwendet er die „headless chicken"-Rekombination, bei der einer der beiden Rekombinationspartner rein zufällig erzeugt wurde, und vergleicht diese Evolution mit Evolutionsläufen, bei denen beide Rekombinationspartner der Population entnommen wurden. Auf drei verschiedenen Problemstellungen angewendet, zeigen die beiden Varianten entweder keinen statistisch signifikanten Unterschied, oder die Variante mit der „headless chicken"-Rekombination schneidet geringfügig besser ab. Hypothese 10.1 stellt eine Verallgemeinerung von Angelines Aussage dar und macht das Problem an der fehlenden Bewahrung des Kontext fest. Der Kontext ist bei vielen Repräsentationen gegeben durch die Position der Operation im Genom und den Positionen anderer Operationen.

Chellapilla [29] bearbeitet 14 Problemstellungen mithilfe der rein mutationsbasierten evolutionären Programmierung (EP) und vergleicht den evolutionären Aufwand mit dem eines GP-Systems (mit Rekombination). Er stellt fest, dass EP die Probleme mit zumeist geringerem Aufwand löst. Luke und Spector [73] vergleichen Baum-GP Läufe mit und ohne Rekombination und ermitteln, dass bei den Problemstellungen, die von der Verwendung der Rekombination profitieren, der Gewinn äußerst gering ist. Andere Parameter wie die Tournamentgröße haben hier wesentlich mehr Einfluss auf den Erfolg des Evolutionsverlaufs.

Für die hier verwendete lineare GP-Variante ermitteln die SPO-Läufe Rekombinationsraten in der Größenordnung von 30%–40%. Die aus den SPO-Läufen gewonnenen Effektplots (Abb. 8.3, A.2 & A.4) zeigen eine Verschlechterung bei stark abweichender Rekombinationsrate. Diese gibt allerdings keinen Aufschluss darüber, ob hier ein erfolgreicher Austausch von Informationen zwischen den Individuen oder eine Form von Makromutation stattfindet. Die Interaktionsplots zeigen jedoch, dass bei geringer Mutationsrate eine höhere Rekombinationsrate zu bevorzugen ist und umgekehrt. Diese „Austauschbarkeit" ist zumindest ein Hinweis auf eine mutative Wirkung der Rekombination.

11.2. AC-Evolution ohne Rekombination

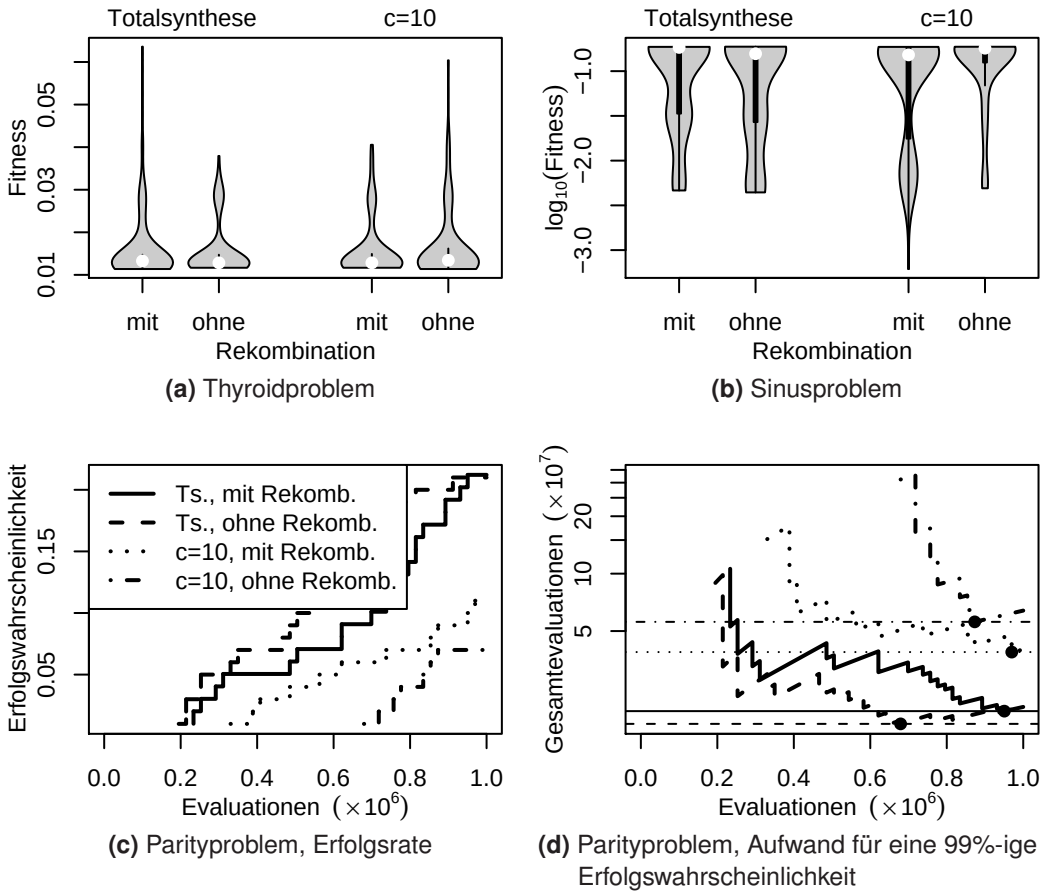
Die in Kapitel 5.3 vorgestellte Rekombination ist dadurch motiviert, dass Unterschiede bei der Ausführung einer algorithmischen Chemie und eines linearen Programms auf die Verwendung unterschiedlicher Zuordnungen zwischen Programmzeiger und der ausgeführten Instruktion bzw. Reaktion zurückgeführt werden können. Die 1-Punkt-Rekombination wird ebenfalls über den Programmzeiger definiert. Demnach wird der erste Teil des Nachkommens durch jene Instruktionen des ersten Elters gebildet, die bei diesem während des Zeitraums 0 bis t_1 ausge-

führt werden. Den hinteren Teil bilden Instruktionen aus dem zweiten Elter, die hier zwischen einem Zeitpunkt t_2 und der vollständigen Ausführung ausgewertet werden. Die so definierte 1-Punkt-Rekombination wird auch für die Rekombination zweier Individuen verwendet, die eine algorithmische Chemie codieren. Durch die gleichverteilte Wahl einer Instruktion, unabhängig von der Position des Programmzeigers, wird bei der Rekombination eine zufällige Reaktionsmultimenge aus beiden Eltern entnommen und im Nachkommen vereint. Die Rekombinationspunkte t_1 und t_2 bestimmen dabei nur die Anzahl der übernommenen Reaktionen und haben ansonsten keinen weiteren Einfluss auf die Rekombination.

Die ursprüngliche Annahme über das Systemverhalten war, dass benötigte Instruktionen im Individuum akkumuliert würden und damit auch bei dieser Form der Rekombination eine hohe Wahrscheinlichkeit besteht, dass Nachkommen zumindest eine Instanz jeder Reaktion enthalten, die sie zur Assemblierung einer Lösung benötigen. Ohne diese Akkumulation ist die Wahrscheinlichkeit gering, dass bei der 1-Punkt-Rekombination von jeder benötigten Reaktion eine Instanz in den Nachkommen gelangt. Die Rekombination führt dann zwangsläufig zu sehr schlechten, da unvollständigen Nachkommen.

Die von der Parameteroptimierung vorgeschlagene Rekombinationsrate ist mit Werten zwischen 0.1% und 2.7% nahe, aber nicht gleich 0. Hierfür kann es zwei Gründe geben. Zum einen arbeitet die Parameteroptimierung in keiner Phase als rein lokale Suche. Alle betrachteten Einstellungen entstammen einem globalen Latin Hypercube Design. Dies hat zur Folge, dass je kleiner der Raum zwischen dem aktuell besten Designpunkt und dem tatsächlichen Optimum ist, desto geringer ist die Wahrscheinlichkeit, dass eine weitere Verbesserung erzielt werden kann. Die verbleibende Rekombinationsrate könnte durch die Parameteroptimierung verursacht sein. Andererseits kann es sich aber auch um genau das benötigte Maß an Rekombination handeln. Etwa dann, wenn diese, wie oben vermutet, als Makromutation wirkt und hilft, einem lokalen Optimum zu entkommen. Um Letzteres zunächst auszuschließen, wurden die Experimente zur Evolution einer AC mit einer Rekombinationsrate von 0% sowohl unter Verwendung der Totalsynthese als auch mit einem Vielfachen von $c = 10$ wiederholt.

Die Ergebnisse sind in Abb. 11.1 dargestellt. Die Läufe mit und ohne Rekombination wurden für das Sinusproblem (Ts.: $p = 0.083$, $c = 10$: $p = 0.007$) und das Thyroidproblem (Ts.: $p = 0.619$, $c = 10$: $p = 0.173$) mittels Wilcoxon-Rangsummentest verglichen. Die Hypothese über die Gleichheit der Systeme kann bei einem Signifikanzniveau von $\alpha = 0.05$ nur im Fall des Sinusproblems bei Verwendung eines Vielfachen ($c = 10$) zugunsten der ungerichteten Alternativhypothese verworfen werden. Der Vergleich der Erfolgsraten auf dem Parityproblem erlaubt weder unter Verwendung der Totalsynthese ($p = 1$) noch unter Verwendung des Vielfachen $c = 10$ ($p = 0.459$) die Nullhypothese (Gleichheit) zugunsten der Alternativhypothese zu verwerfen. Es ist davon auszugehen, dass die Verwendung der 1-Punkt-Rekombination, in dem geringem Ausmaß wie sie bisher stattfindet, in den meisten Fällen keinen Einfluss auf die Evolution algorithmischer Chemien hat.



(a) Thyroidproblem

(b) Sinusproblem

(c) Parityproblem, Erfolgsrate

(d) Parityproblem, Aufwand für eine 99%-ige Erfolgswahrscheinlichkeit

Abbildung 11.1.: AC-Evolution mit und ohne 1-Punkt-Rekombination

11.3. Homologe Rekombination in der Genetischen Programmierung

Die Vorstellung, dass im Verlauf der Evolution unterschiedliche Funktionalität parallel entsteht und mittels Rekombination zu bedeutenderer Funktionalität kombiniert wird, macht die Rekombination als Suchoperator interessant und bleibt auch dann wünschenswert, wenn die Eingangs diskutierten Arbeiten Zweifel an der Funktionsweise der Standardrekombinationsoperatoren für die verschiedenen Repräsentationen äußern. Viele dieser Arbeiten stellen fest, dass die unterschiedlichsten Problemstellungen mit der Mutation als einzige Operation häufig genau so gut gelöst werden können. Dem gegenüber steht ein Beweis von Jansen und Wegener [53], die für einen genetischen Algorithmus und ein speziell konstruiertes Problem zeigen, dass die Evolution einer Lösung durch Verwendung von Rekombination deutlich beschleunigt wird. Auch wenn ein solcher Beweis für

die genetische Programmierung im Speziellen nicht existiert, zeigt er doch, dass ein funktionierender Rekombinationsoperator für evolutionäre Algorithmen im Allgemeinen von Bedeutung sein kann. Aus diesen Gründen wird im Folgenden ein weiterer Rekombinationsoperator konstruiert. Der Rekombinationsoperator ist angelehnt an die *homologen Rekombinationsoperatoren* anderer Repräsentationen in der genetischen Programmierung. Diese Operatoren versuchen in den Individuen Bereiche gleichen Ursprungs oder ähnlicher Funktionalität zu identifizieren und tauschen diese zwischen den Individuen aus.

Francone u. a. [46] beschreiben eine homologe Rekombinationsoperation für das lineare GP. Sie argumentieren, dass Homologie den Austausch von Genen vergleichbarer Funktionalität ermöglicht, wobei die funktionale Homologie erst durch die Existenz positioneller Homologie emergiert. Positionelle Homologie wird durch das Binden komplementärer Basenpaare während der Rekombination ermöglicht. Dementsprechend unterstützt ihre homologe Rekombinationsoperation zunächst nur die positionelle Homologie, indem ausschließlich Code an der gleichen Position zwischen den Eltern ausgetauscht wird. Die effektive Fitness führt ihren Erwartungen nach dazu, dass die Individuen funktional ähnlichen Code an der gleichen Position tragen. *Die effektive Fitness eines Individuums wird im Gegensatz zur reinen Fitness zusätzlich durch die Existenz von Eigenschaften bestimmt, die jenseits der Selektion die Einflussnahme des Individuums auf die weitere Evolution ermöglichen.* In diesem Fall erhöht die Ausbildung von Funktionalität an einer bestimmten Position im Genom die Wahrscheinlichkeit des erfolgreichen Austausches mit anderen Individuen bei der Rekombination und damit die Fitness der Nachkommen.

effektive Fitness

Für die baumbasierten Repräsentationen formuliert Langdon [66, 67] einen homologen Rekombinationsoperator. Diese versucht zum einen möglichst gleich große Teilbäume auszutauschen, zum anderen die Distanz zwischen den Knoten an denen diese Teilbäume entnommen werden gering zu halten und somit auch hier eine positionelle Homologie zu erzielen. Die Distanz zweier Knoten in unterschiedlichen Bäumen wird dabei über die Tiefe des Knotens bewertet, an dem ihr Pfad zur Wurzel divergiert.

Sowohl Francone u. a. als auch Langdon erzielen Resultate, die mit Ergebnissen unter Verwendung nicht-homologer Rekombination vergleichbar sind, zudem melden sie eine deutliche Reduktion in der absoluten Größe ihrer Individuen. Dieses stützt die in Kap. 10 vorgestellte Hypothese, dass Bloat dem Schutz vor der destruktiven Wirkung der Rekombination dient. Francone u. a. [46] sprechen beim Bloat sogar von einer emergenten Form positionsunabhängiger Homologie, wenngleich die Homologie darin besteht, dass zwischen den beiden Eltern funktionsloser Code ausgetauscht wird.

11.4. Homologe Rekombination Algorithmischer Chemien

Die Semantik von Building Blocks in einer algorithmischen Chemie ist nicht an die Position ihrer Codierung im Genom gebunden. Diese Positionsabhängigkeit ist der

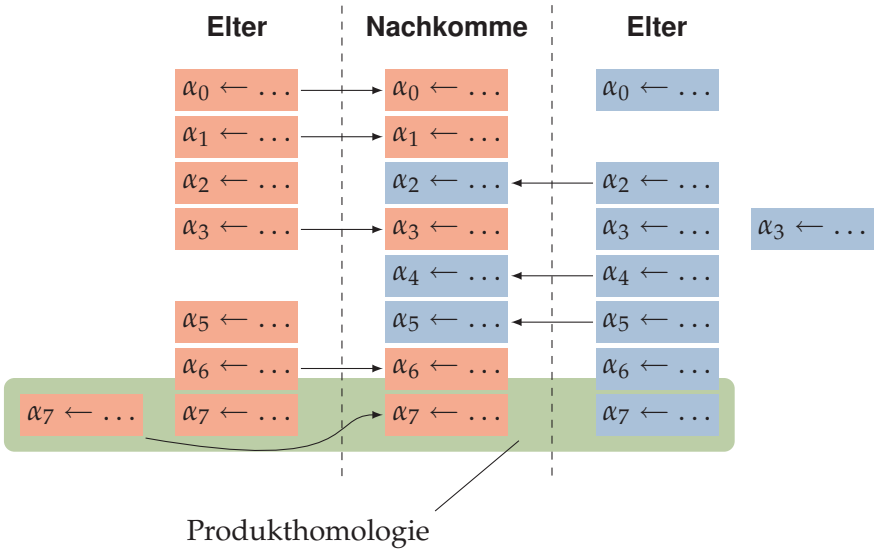


Abbildung 11.2.: Homologe Rekombination zweier algorithmischer Chemien. Homologie bezieht sich auf die Produkte der Reaktionen. Für jedes potentielle Produkt wird genau eine Reaktion mit diesem Produkt aus einem der beiden Eltern zufällig gewählt.

Grund dafür, dass etwa Angeline [4] gängigen Repräsentationen die Fähigkeit zur Kombination von Building Blocks abspricht. Aber auch ohne diese Abhängigkeit aufzuweisen, ist die Rekombination algorithmischer Chemien nicht erfolgreich. Als Grund hierfür wird angenommen, dass die Wahrscheinlichkeit für die (vollständige) Übernahme eines Building Block in den Nachkommen zu gering ist.

Die effektive Fitness der Individuen könnte gesteigert werden, indem die zu den Building Blocks gehörenden Reaktionen in der Chemie akkumuliert werden. Die Evolution hat Gelegenheit hierzu, macht jedoch davon keinen Gebrauch. Stattdessen bevorzugt die SPO sogar den Verzicht auf die Rekombination. Es lassen sich zwei Hypothesen aufstellen, welche die Entscheidung gegen die Akkumulation von Instruktionen erklären könnten. Zunächst einmal erhöht jede Kopie einer Reaktion die Wahrscheinlichkeit, dass eine ihrer Kopien durch eine Mutation geändert wird. Hierdurch erhöht sich die Unbestimmtheit der Chemie, was bei guten Individuen zu einer Verschlechterung der Fitness führen kann. Bei schlechten Individuen andererseits verringert eine Akkumulation, hier gesehen als eine Form von Redundanz, die Intensität der Wahrnehmung einer mutativen Verbesserung, da sich die Verbesserung nur bei einem Teil der Verhaltensvarianten bemerkbar macht. In beiden Fällen führt die Akkumulation von Reaktionen nicht zu der Integration von Robustheit und Flexibilität, welche von Kirschner und Gerhart [61] als essenziell für die Evolutionsfähigkeit erachtet wird.

Bei der algorithmischen Chemie bezieht sich die Homologie auf die Produkte der Reaktionen. Die Produkte sind jene an einer Reaktion teilhabenden Moleküle, die in ihrem Zustand verändert werden. Es wird dabei für jedes potenzielle Pro-

Tabelle 11.1.: Systemkonfiguration bei unterschiedlichen Rekombinationsoperatoren. Die ersten drei Spalten geben die Definitionsbereiche an, innerhalb derer die Parameter optimiert werden. Die Obergrenze der Rekombinationsrate liegt bei der 1-Punkt-Rekombination bei 30%, und wird für die homologe Rekombination, wie bereits beim LGP-System, auf 100% gesetzt. Es folgen jeweils zwei Spalten mit den für die 1-Punkt-Rekombination und der homologen Rekombination gefundenen Einstellungen.

	„region of interest“			1-Punkt		homolog	
Parameter	min	max	$\mathbb{N}/\mathbb{R}$	Par4	Sin	Par4	Sin
Eltern	100	4000	$\mathbb{N}$	150	133	470	167
Selektionsdruck	1.01	7	$\mathbb{R}$	6.8	6.64	3.81	6.73
Mutationsrate (in %)	0	4	$\mathbb{R}$	2.27	3.14	1.05	2.19
Rekomb.rate (in %)	0	30/100	$\mathbb{R}$	0.12	2.7	25.9	30.2
Moleküle/Register	15	30	$\mathbb{N}$	24	27	30	29

dukt aus den Reaktionen beider Eltern, genau eine Reaktion zufällig gewählt, die dessen Zustand ändert. In Abbildung 11.2 ist dieses Vorgehen skizziert. Hierdurch soll erreicht werden, dass ohne Akkumulation von Reaktionen der Kontext einer Reaktion existiert. Dieser wird gewählt von den Edukten der Reaktion und fehlt, wenn zustandsveränderliche Moleküle gewählt werden, deren Zustand durch keine Reaktion bestimmt wird.

Nach der Rekombination existiert in den Nachkommen für jedes zustandsveränderliche Molekül zunächst genau eine Reaktion, die dieses adressiert, vorausgesetzt mindestens eines der beiden Eltern besitzt eine solche Reaktion. Erst in der anschließenden Mutationsphase kann es dazu kommen, dass zwei Reaktionen mit dem gleichen Produkt im Individuum definiert sind. Dieses geschieht dann, wenn durch Mutation das Produkt einer Reaktion verändert wird. Zwei identische Eltern erzeugen durch Rekombination einen Nachkommen der ihnen gleicht, wenn bei ihnen jedes Molekül von maximal einer Reaktion in seinem Zustand verändert wird.

Nach wenigen Generationen ist davon auszugehen, dass für jedes potenzielle Produkt Reaktionen in den Individuen codiert sind, die seinen Zustand ändern. Die Größe der Individuen ist dann annähernd konstant und bestimmt durch die Anzahl der Moleküle, die als Produkt wählbar sind. Die Größe des darin codierten Datenflusses ist aber weiterhin variabel. Auch andere Systeme arbeiten mit einem Genom fester Größe, in dem variabel große Datenflüsse codiert werden, so z. B. Cartesian GP [78] (vgl. Kap. 1.2.5).

11.5. Empirische Ergebnisse

Die neue Rekombinationsoperation wird im Folgenden verwendet, um Lösungen für das Parity- und Sinusproblem zu evolvieren. In einem ersten Schritt erfolgt zunächst wieder die sequenzielle Parameteroptimierung. Hierbei wurde weitestgehend so verfahren, wie bereits in Kapitel 8. Im Unterschied dazu beträgt aber

die Obergrenze für die Bestimmung der Rekombinationsrate nun 100% anstelle von 30%, da hier ein erstes Screening nicht darauf hindeutet, dass ausschließlich geringe Rekombinationsraten erfolgreich sind. Dieses entspricht der für das LGP-System verwendeten Obergrenze. Zudem wurde zur Optimierung der Parameter auf dem Sinusproblem der Logarithmus der Fitness herangezogen. Damit wird der Beobachtung bei der Potenzreihe Rechnung getragen, dass bei linearer Zunahme der Komplexität die Fitness nur exponentiell abnimmt.

Die letzten beiden Spalten der Tab. 11.1 enthalten die besten gefundenen Einstellungen. Die Tabellen A.3 und A.4 im Anhang tabellieren weitere betrachtete Einstellungen und den Verlauf der SPO. Die Abbildungen A.5 und A.6 zeigen die aus dem zugrunde liegenden Modell gewonnen Effekt- und Interaktionsplots. Während die 1-Punkt-Rekombination durch die Wahl sehr kleiner Rekombinationsraten im Rahmen der Parameteroptimierung nahezu ausgeschaltet wurde, schlägt die SPO unter Verwendung der homologen Rekombination Raten von ca. 26% und 30% vor. Die zugehörigen Effektplots zeigen für geringere Rekombinationsraten zum Teil eine deutliche Verschlechterung der im Durchschnitt der anderen Parameter zu erwartenden Fitness. Aber auch bei zu hohen Rekombinationsraten wird eine Verschlechterung angezeigt. Eine starke Interaktion zwischen Rekombinations- und Mutationsrate wie beim LGP-System ist hier nicht zu beobachten. Das System bevorzugt wieder eine geringe Populationsgröße, der gewählte Selektionsdruck ist für das Sinusproblem wie zuvor recht hoch, beim Parityproblem ist er mit einem Wert von 3.81 deutlich geringer. Auch der Effektplot (Abb. A.5) deutet auf eine höchstens geringe Verbesserung durch einen höheren Selektionsdruck hin. Im Folgenden werden die Einstellungen aus Tabelle 11.1 verwendet.

11.5.1. Fitness

In zusätzlichen Evolutionsläufen werden sowohl die Totalsynthese als auch unterschiedliche Vielfache $c \in \{10, 15, 20, 40\}$ der Genomgröße als Anzahl ausgeführter Reaktionen verwendet. Das Vielfache c wird hier aus zwei Gründen in den angegebenen Schritten variiert. Zum einen zeigen die Ergebnisse mit dem kleinsten auch in Kap. 9 gewählten Wert deutliche Abweichungen zu den mit Totalsynthese erzielten Ergebnissen. Abweichungen in diesem Ausmaß waren zuvor nicht zu beobachten. Zum anderen ist die Größe der Individuen, die als zweiter Faktor die Anzahl ausgeführter Reaktionen bestimmt, nahezu fest. Aus diesem Grund kann die Evolution nicht mehr durch eine veränderte Individuengröße Einfluss auf die Anzahl ausgeführter Reaktion nehmen. Der Einfluss Anzahl ausgeführter Reaktionen auf den Erfolg der Ausführung und damit auf die Evolution wird in Kap. 12 noch genauer diskutiert. Für jede Variante der Parametrisierung werden 100 Läufe auf jedem der beiden Probleme durchgeführt.

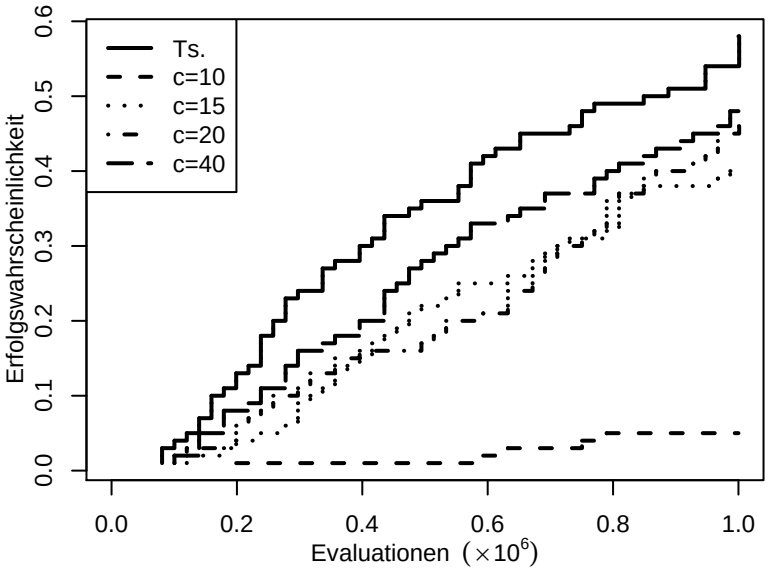


Abbildung 11.3.: Erfolgsrate auf dem Parityproblem in Abhängigkeit des Evolutionsaufwands unter Verwendung der homologen Rekombination. Variiert wird die Anzahl ausgeführter Instruktionen über das Vielfache c. Zusätzlich findet die Totalsynthese Verwendung.

Tabelle 11.2.: Aufwand, um mit 99%-iger Wahrscheinlichkeit eine Lösung für das Parityproblem zu evolvieren. Für die unterschiedlichen Systemvarianten ist die maximal beobachtete Erfolgsrate $P_s^{\max}$, die ideale Anzahl Evaluationen pro Lauf $e^\star = \arg\min R(99\%, e) \cdot e$, die dazugehörige Anzahl an Läufen $R(z, e^\star)$ sowie der daraus resultierende Gesamtaufwand $e^\star \cdot R(z, e^\star)$ angegeben. Die Berechnung dieser Werte ist in Kap. 9.3 beschrieben.

Variante	$P_s^{\max}$	$e^\star$	$R(z, e^\star)$	$e^\star \cdot R(z, e^\star)$
LGP				
	0.66	376740	8	3013920
AC, 1-Punkt				
Totalsyn.	0.21	950640	20	19012800
AC, homolog				
Totalsyn.	0.58	277605	18	4996890
$c = 10$	0.05	789831	90	71084790
$c = 15$	0.40	829233	10	8292330
$c = 20$	0.46	967140	8	7737120
$c = 40$	0.48	573120	12	6877440

Tabelle 11.3.: p-Werte der paarweisen Vergleiche der Anteile erfolgreicher Läufe auf dem Parityproblem unter der Nullhypothese einer gleichen Erfolgsrate. Die p-Werte wurden nach Holm [52] angepasst.

Vergleich mit	LGP	AC, 1-Punkt, Ts.	AC, homolog, Ts.	AC, homolog, $c = 40$	AC, homolog, $c = 20$	AC, homolog, $c = 15$
AC, 1-Punkt, Ts.	<0.001	–	–	–	–	–
AC, homolog, Ts.	1.000	<0.001	–	–	–	–
AC, homolog, $c = 40$	0.121	0.002	1.000	–	–	–
AC, homolog, $c = 20$	0.065	0.005	0.717	1.000	–	–
AC, homolog, $c = 15$	0.005	0.065	0.121	1.000	1.000	–
AC, homolog, $c = 10$	<0.001	0.016	<0.001	<0.001	<0.001	<0.001

Parityproblem

Abbildung 11.3 zeigt für die Läufe auf dem Parityproblem die Erfolgsrate im Verlauf der Evolution, Tab. 11.2 enthält für die unterschiedlichen Varianten die optimale Laufzeit und die Anzahl der Einzelläufe sowie den daraus resultierenden minimalen Gesamtaufwand zum Evolvieren einer Lösung mit 99%-iger Erfolgswahrscheinlichkeit. Wie bereits oben angedeutet weist die Variante mit einem Vielfachen von $c = 10$ den größten Unterschied zu den anderen Läufen auf. Sie schneiden mit einer Erfolgsrate von 5% am schlechtesten ab, wohingegen bereits ab einem Vielfachen von $c = 15$ mit einer Erfolgsrate von 40% nur ein geringer Unterschied zu den anderen Varianten festzustellen ist. Lediglich die Läufe unter Verwendung der Totalsynthese zeigen eine etwas höhere Erfolgsrate. Der Aufwand, um mit der erforderlichen Wahrscheinlichkeit einen Erfolg zu erzielen, wird durch Verwendung der homologen Rekombination deutlich reduziert, wenngleich dieser noch etwas höher liegt als beim linearen Referenzsystem.

Die Hypothese von einer gleichen Erfolgsrate dieser sieben unterschiedlichen Systeme kann verworfen werde, $p < 0.001$. Die p-Werte der anschließenden paarweisen Vergleiche der Anteile erfolgreicher Läufe sind in Tab. 11.3 dargestellt. Die p-Werte wurden nach Holm [52] angepasst. Innerhalb der Gruppe von Systemen, in denen eine algorithmische Chemie unter Verwendung der homologen Rekombination evolviert wird, kann die Hypothese einer identischen Erfolgsrate nur im Vergleich mit der Verwendung eines Vielfachen $c = 10$ zum Signifikanzniveau von $\alpha = 0.05$ abgelehnt werden. Diese Einstellung hat bei weitem die geringste beobachtete Erfolgsrate. Die Ablehnung der Gleichheitshypothese beim Vergleich der AC-Systeme

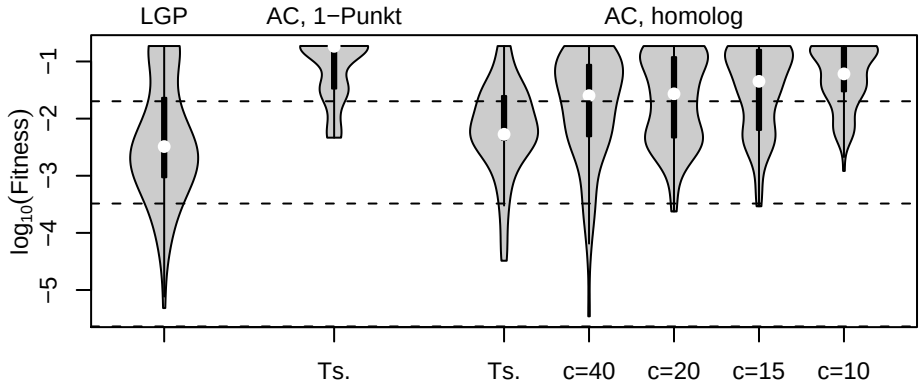


Abbildung 11.4.: Verteilung der Güte gefundener Lösungen auf dem Sinusproblem in Abhängigkeit der gewählten Anzahl ausgeführter Reaktionen bzw. bei Verwendung der Totalsynthese. Rechts dargestellt sind die Läufe mit homologer Rekombination. Zusätzlich abgebildet ist die Verteilung unter Verwendung der 1-Punkt-Rekombination und die Verteilung der linearer GP Läufe.

mit dem linearen GP-System gelingt nur für die 1-Punkt-Rekombination und für die Varianten mit homologer Rekombination und einem Vielfachen $c \leq 15$.

Sinusproblem

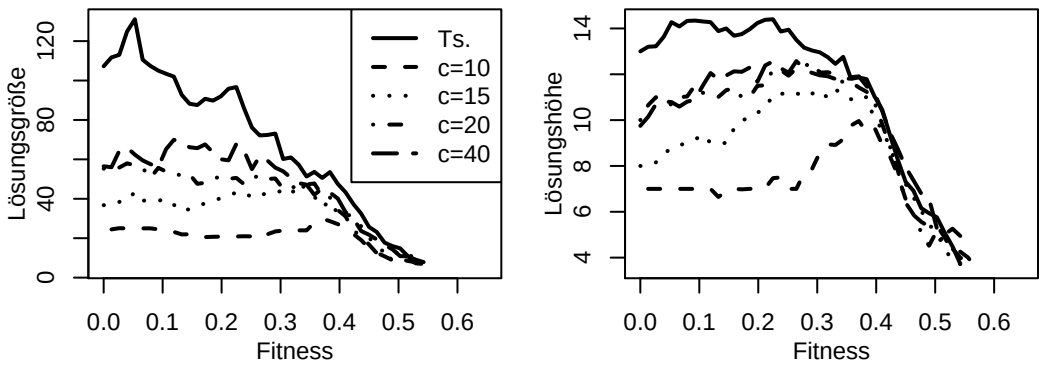
Abbildung 11.4 zeigt für das Sinusproblem die Verteilung der besten Ergebnisse über 100 Läufe. Der Kruskal-Wallis-Rangsummentest zeigt, dass die Hypothese einer Performancegleichheit aller Systeme zum Signifikanzniveau $\alpha = 0.05$ verworfen werden kann, $p < 0.001$. Tabelle 11.4 enthält die p-Werte aller paarweisen Vergleiche der Systeme mittels Wilcoxon-Rangsummentests. Der Alphafehlerkumulierung wird durch Anpassung der p-Werte nach Holm entgegengewirkt. Bei den meisten Kombinationen kann die Gleichheitshypothese zum gewählten Signifikanzniveau $\alpha = 0.05$ verworfen werden. Ausnahmen hiervon bilden zum einen die Vergleiche jener AC-Systeme, die mit nahe beieinanderliegenden Werten für das Vielfache c arbeiten. Zum anderen kann die Hypothese, dass die Evolution einer algorithmischen Chemie unter Verwendung homologer Rekombination und Totalsynthese Approximationen der Sinusfunktion in gleicher Güte hervorbringt wie die Evolution eines linearen Programms, nicht zum erforderlichen Signifikanzniveau verworfen werden ($p = 0.072$).

11.5.2. Individuengröße und -höhe

Die Abbildungen 11.5 und 11.6 zeigen den Verlauf der Mediane von Höhe und Größe der besten Individuen in Abhängigkeit ihrer Fitness. Die Werte wurden wie in Kapitel 10 beschrieben ermittelt. Zunächst einmal ist der Einfluss der Anzahl ausgeführter Reaktionen (in Form des Vielfachen c) auf Höhe und Größe der evolvierten Individuen zu beobachten. Diese nehmen mit steigender Anzahl Reaktionen zu und haben ihr Maximum bei der Totalsynthese, welche versucht, das Verhalten

Tabelle 11.4.: p-Werte der paarweisen Wilcoxon-Rangsummentests für die Resultate auf dem Sinusproblem. Die p-Werte wurden nach Holm [52] angepasst.

Vergleich mit	*LGP*	*AC, 1-Punkt, Ts.*	*AC, homolog, Ts.*	*AC, homolog, $c = 40$*	*AC, homolog, $c = 20$*	*AC, homolog, $c = 15$*
AC, 1-Punkt, Ts.	<0.001	–	–	–	–	–
AC, homolog, Ts.	0.072	<0.001	–	–	–	–
AC, homolog, $c = 40$	<0.001	<0.001	0.001	–	–	–
AC, homolog, $c = 20$	<0.001	<0.001	0.001	0.488	–	–
AC, homolog, $c = 15$	<0.001	<0.001	<0.001	0.043	0.115	–
AC, homolog, $c = 10$	<0.001	0.043	<0.001	<0.001	<0.001	0.115



(a) Größe **(b)** Höhe

Abbildung 11.5.: Median von Größe und Höhe der evolvierten booleschen Funktionen zur Bestimmung der Parität dargestellt in Abhängigkeit der Fitness und der Anzahl ausgeführter Reaktionen. Verwendet wurde die homologe Rekombination.

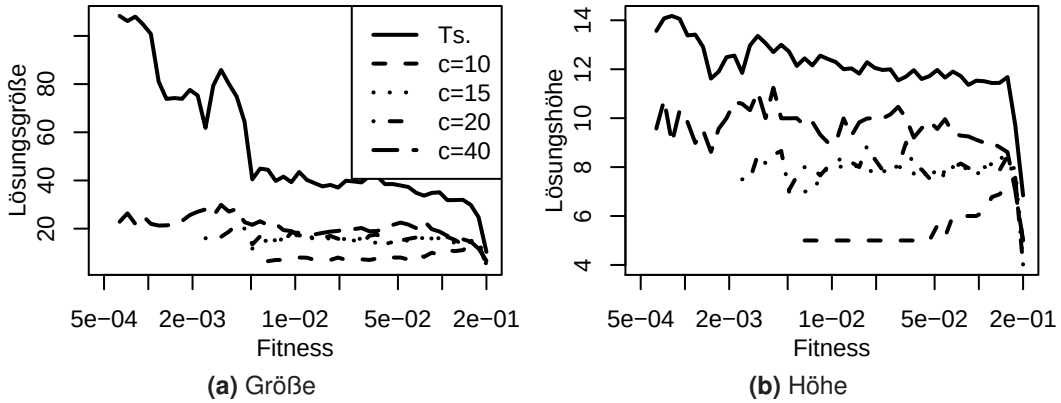


Abbildung 11.6.: Median von Größe und Höhe der evolvierten Approximation der Sinusfunktion in Abhängigkeit der Fitness und der Anzahl ausgeführter Reaktionen. Verwendet wurde die homologe Rekombination.

bei unendlicher Laufzeit anzunähern. Die Läufe auf dem Parityproblem zeigen zudem eine leichte Reduktion der Höhe bei Annäherung an die optimale Fitness (Abb. 11.5b). Eine deutliche Reduktion der Höhe findet bei beiden Problemstellungen für die Evaluationsläufe mit der geringsten Anzahl ausgeführter Reaktionen ($c = 10$) bereits bei schlechter Fitness statt. Der Median der Individuenhöhe erreicht zu diesem Zeitpunkt Werte zwischen 8 und 10. Diese Systemvariante weist auch eine vergleichsweise frühe Stagnation bzw. eine geringe Erfolgsrate auf. Die Zusammenhänge zwischen der Anzahl ausgeführter Reaktionen, dem Evolutionsfortschritt und der Individuenhöhe werden im nächsten Kapitel diskutiert.

Die Individuumsgröße ist beim Parityproblem für die Variante ohne Totalsynthese nach einer anfänglichen Wachstumsphase weitestgehend konstant (Abb. 11.5a). Bei Verwendung der Totalsynthese steigt diese jedoch bis zum Erreichen der perfekten Lösung und nimmt dabei Werte in der Größenordnung des linearen GP-Systems an. Die Höhe der evolvierten Individuen bleibt dabei deutlich unter der des LGP-Systems (vgl. Abb. 10.3 mit 11.5b). Ein ähnliches Verhalten ist bei der Evolution der symbolischen Regression des Sinus zu beobachten, hier findet bei Verwendung der Totalsynthese ein erneuter Anstieg der Größe ab einer Fitness von $5 \cdot 10^{-3}$ statt. Ein solcher Anstieg, wenn gleich in geringerem Ausmaß, lässt sich an dieser Stelle auch bei der Evolution linearer Programme beobachten. Die Datenflussgröße algorithmischer Chemien wird aufgrund der Annahme von Mehrdeutigkeit pessimistisch abgeschätzt. Werden in parallelen Zweigen identische Moleküle verwendet, so wird der Teil des Datenflusses, der zur Bestimmung ihres Wertes führt mehrfach gezählt. Dieses geschieht, da die Moleküle zu unterschiedlichen Zeitpunkten verwendet worden sein können und bei fehlender Eindeutigkeit zu diesen Zeitpunkten möglicherweise unterschiedlich belegt sind. Durch die homologe Rekombination ist Mehrdeutigkeit zumindest bei Verwendung der Totalsynthese nur noch bei

Tabelle 11.5.: Median von Größe und Höhe sowie die Anzahl der von den verschiedenen Systemen gefundenen „guten“ Individuen für das Parity- und Sinusproblem.

	Sinus			Parity		
System, Variante	Größe	Höhe	Anz.	Größe	Höhe	Anz.
LGP	65.00	23.00	73	87.00	34.00	66
AC, 1-Punkt, Ts.	10.25	7.00	16	43.50	9.00	21
AC, homolog, Ts.	40.50	11.50	69	107.25	13.00	58
AC, homolog, $c = 40$	18.00	9.00	45	56.50	9.75	48
AC, homolog, $c = 20$	15.00	8.00	46	55.25	10.00	46
AC, homolog, $c = 15$	14.00	7.00	29	36.75	8.00	40
AC, homolog, $c = 10$	7.00	5.00	18	21.00	6.00	5

entsprechender Mutation eines Reaktionsprodukts möglich, sodass die tatsächliche Individuengröße nach oben durch die Anzahl zustandsveränderlicher Moleküle (hier 29 bzw. 30) begrenzt sein dürfte.

Der weitere Vergleich bezieht sich auf die „guten“ Individuen wie sie bereits in Kap. 10.3 definiert wurden. Die Mediane von Individuengröße und -höhe aller Läufe mit „guten“ Individuen sind zusammen mit ihrer Anzahl in Tab. 11.5 dargestellt. Tabelle 11.6 enthält die nach Holm [52] angepassten p-Werte aus den paarweisen Wilcoxon-Rangsummentests.

Bei Betrachtung der Läufe auf dem Sinusproblem fällt auf, dass Höhe und Größe der vom LGP-System gefundenen Lösung die der anderen Systeme übersteigen. Die Hypothese auf Gleichheit kann beim Vergleich mit allen anderen Systemen zum Signifikanzniveau $\alpha = 0.05$ verworfen werden. Das AC-System evolviert unter Verwendung der Totalsynthese die zweit größten Individuen. Auch hier kann beim Vergleich mit den anderen AC-Varianten die Hypothese auf Größen- und Höhengleichheit verworfen werden. Die beiden Systeme weisen aber nicht nur vergleichsweise große und hohe Individuen auf, sondern von ihnen werden häufiger die guten Lösungen gefunden, ca. 60%–70% der Läufe weisen die hierfür benötigte Fitness auf. Bei der Betrachtung des Parityproblems ist ein ähnliches Verhältnis zwischen Systemtyp, Individuenhöhe und Erfolgsrate zu beobachten.

Zusammenfassung und Diskussion

Zentrales Thema dieses Kapitels ist die Rekombination. Die Designs der AC-Variante haben gezeigt, dass unter Verwendung der 1-Punkt-Rekombination die besten Resultate bei einer geringen Rekombinationsrate erzielt werden. Der Anteil Nachkommen, die durch Rekombination entstanden sind, wird dabei so gering gewählt, dass sich die Frage stellt, ob der zunächst verwendete Rekombinationsoperator überhaupt zum Erfolg der Evolution beiträgt. Läufe, die zum Vergleich ohne Rekombination durchgeführt werden, zeigen, dass dieses nicht der

Tabelle 11.6.: p-Werte der paarweisen Wilcoxon-Rangsummentests für die Größe und Höhe „guter“ Individuen von Sinus- und Parityproblem. Die p-Werte wurden nach Holm [52] angepasst.

Vergleich mit	*LGP*	*AC, 1-Punkt, Ts.*	*AC, homolog, Ts.*	*AC, homolog, $c = 40$*	*AC, homolog, $c = 20$*	*AC, homolog, $c = 15$*
Sinusproblem, Größe						
AC, 1-Punkt, Ts.	<0.001	–	–	–	–	–
AC, homolog, Ts.	0.002	<0.001	–	–	–	–
AC, homolog, $c = 40$	<0.001	0.002	<0.001	–	–	–
AC, homolog, $c = 20$	<0.001	0.239	<0.001	0.066	–	–
AC, homolog, $c = 15$	<0.001	0.520	<0.001	0.028	0.520	–
AC, homolog, $c = 10$	<0.001	0.062	<0.001	<0.001	<0.001	<0.001
Sinusproblem, Höhe						
AC, 1-Punkt, Ts.	<0.001	–	–	–	–	–
AC, homolog, Ts.	<0.001	<0.001	–	–	–	–
AC, homolog, $c = 40$	<0.001	<0.001	0.001	–	–	–
AC, homolog, $c = 20$	<0.001	0.300	<0.001	0.005	–	–
AC, homolog, $c = 15$	<0.001	0.604	<0.001	0.001	0.539	–
AC, homolog, $c = 10$	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
Parityproblem, Größe						
AC, 1-Punkt, Ts.	0.322	–	–	–	–	–
AC, homolog, Ts.	0.013	<0.001	–	–	–	–
AC, homolog, $c = 40$	0.778	0.206	0.001	–	–	–
AC, homolog, $c = 20$	0.558	0.558	<0.001	0.778	–	–
AC, homolog, $c = 15$	0.007	0.558	<0.001	<0.001	<0.001	–
AC, homolog, $c = 10$	0.293	0.293	0.005	0.005	0.010	0.293
Parityproblem, Höhe						
AC, 1-Punkt, Ts.	<0.001	–	–	–	–	–
AC, homolog, Ts.	<0.001	<0.001	–	–	–	–
AC, homolog, $c = 40$	<0.001	0.086	0.002	–	–	–
AC, homolog, $c = 20$	<0.001	0.242	<0.001	0.271	–	–
AC, homolog, $c = 15$	<0.001	0.242	<0.001	<0.001	<0.001	–
AC, homolog, $c = 10$	0.003	0.086	0.003	0.001	0.003	0.116

Fall ist.

Die Funktionsweise des Rekombinationsoperators aus Kap. 5.3 entspricht der Kombination zufällig aus den beiden Eltern gewählter Reaktionsmultimengen. Dem liegt die Annahme zugrunde, dass für die Ausführung wichtige Reaktionen in den Individuen akkumuliert werden. Ältere Untersuchungen zeigen, dass eine Akkumulation stattfindet, wenn eine höhere Rekombinationsrate verwendet wird [16]. Unter der geringen Rekombinationsrate aus der SPO findet jedoch keine Akkumulation statt. Die Beobachtung führt zur Vermutung, dass die Akkumulation von Reaktionen in algorithmischen Chemien der zentralen Erfordernis der Evolutionsfähigkeit widerspricht: der Integration von Flexibilität und Robustheit.

Die Bedeutung der Rekombination für die GP ist nicht unumstritten. Auch wenn viele Beispiele in der Literatur zeigen, dass GP ohne Rekombination erfolgreich sein kann, bleibt zum einen die Kombination guter Teillösungen zu besseren Lösungen wünschenswert und kann zum anderen die bewiesene Bedeutung der Rekombination für verwandte evolutionäre Algorithmen nicht ignoriert werden. Die Suche nach alternativen Umsetzungen der Rekombination führen zu den homologen Rekombinationsoperatoren. Die existierenden Varianten dieses Operators arbeiten zumeist mit einer positionsbezogenen Homologie. Dies ist den bei algorithmischen Chemien aufgrund fehlender Struktur nicht möglich. Alternativ wird hier daher die Homologie auf die Produkte der Reaktionen bezogen. Bei der Rekombination wird nun für jedes Molekül, welches in seinem Zustand verändert werden kann, genau eine Reaktion aus den Chemien beider Eltern gewählt. Die als kritisch angenommene Akkumulation wird damit unterbunden.

Die erneute Anwendung der SPO führt zu Algorithmendesigns mit deutlich höherer Rekombinationswahrscheinlichkeit p_c von 26% bzw. 30%. Die Läufe des AC-Systems mit dieser Einstellung zeigen signifikant bessere Ergebnisse als die Evolution algorithmischer Chemien unter Verwendung der 1-Punkt-Rekombination. Auch im Vergleich der AC-Variante mit Totalsynthese und dem LGP-System zeigen sich keine signifikanten Unterschiede, bezogen auf das Signifikanzniveau $\alpha = 0.05$.

Die verbesserten Evolutionsresultate stützen die Hypothese, dass die Akkumulation von Reaktionen die Evolutionsfähigkeit reduziert. Um die Hypothese weitere zu untersuchen wurde eine weitere Rekombinationsoperation implementiert und in zusätzlichen hier nicht dargestellten Experimenten betrachtet. Auch dieser Operation lag eine auf dem Reaktionsprodukt basierende Homologie zugrunde. Durch die Wahl je einer Reaktion pro Produkt und Elter fand jedoch eine beschränkte Akkumulation statt. Pro zustandsveränderlichen Molekül waren nun nach der Rekombination zwei Instruktionen im Nachkommen enthalten, eine von jedem Elter. Die SPO legte nahe, auch diese Form der Rekombination abzulehnen.

Bei der Evolution algorithmischer Chemien unter Verwendung homologer Rekombination zeigt sich hier die Wechselwirkung zwischen Evolutionserfolg, der Höhe der evolvierten Individuen und der Anzahl ausgeführter Reaktionen. Diese Wechselwirkungen werden im folgenden Kapitel über die Wahrscheinlichkeit erklärt, mit der eine vollständige Assemblierung der Lösung stattfindet.

12. Assemblierungswahrscheinlichkeit

Das vorherige Kapitel zeigt eine Abhängigkeit zwischen der Höhe des effektiven Codes im Individuum, der Anzahl ausgeführter Reaktionen und dem Evolutionserfolg. Die in den Medianen beobachteten Individuengrößen und -höhen nehmen bei den homolog rekombinierenden AC-Varianten mit sinkender Anzahl ausgeführter Reaktionen ab. Leider lässt sich aber auch beobachten, dass die Häufigkeit, mit der diese Lösungen gefunden werden, abnimmt. Langdon und Poli [68] haben im Zusammenhang mit der Ursachensuche für das Bloatverhalten bereits festgestellt, dass mit der Größe der Individuen ihre Anzahl steigt. Die effektive Suche im Raum größerer Individuen bleibt den Systemvarianten mit einer geringen Anzahl ausgeführter Reaktionen allerdings verwehrt. Ein größeres Individuum kann zwar in der Chemie eines Individuums codiert sein, die Wahrscheinlichkeit, dass diese Lösung bei der Ausführung erfolgreich assembliert wird, sinkt aber mit der Anzahl ausgeführter Reaktionen. Ohne hinreichend hohe Wahrscheinlichkeit einer erfolgreichen Assemblierung ist ein erfolgreiches Durchsuchen dieses Teils des Suchraums nicht möglich. Bei der Totalsynthese auf der anderen Seite, die eine vollständige Assemblierung garantiert, macht sich das durch deutlich größere und höhere Individuen bemerkbar. Die Diskussion über Individuengröße und -höhe versus Anzahl ausgeführter Reaktionen ist hier ähnlich zu führen, wie bei der Verwendung expliziter Strafterme zur Reduzierung der absoluten Individuengröße.

Im Folgenden wird eine obere Grenze für die Wahrscheinlichkeit der erfolgreichen Assemblierung eines Individuums der Höhe H bis zum Zeitpunkt T hergeleitet, um die Wechselwirkung zwischen der Höhe des effektiven Codes, der Anzahl ausgeführter Reaktionen und dem Evolutionserfolg zu erfassen. Die Zeit wird hier bemessen an der Anzahl ausgeführter Reaktionen und ist in den Experimenten in Form eines Vielfachen c der im Individuum codierten Reaktionen vorgegeben. Es handelt sich um eine Obergrenze der Assemblierungswahrscheinlichkeit, da gegenüber dem allgemeinen Fall zwei Einschränkungen gemacht werden, welche die Wahrscheinlichkeit einer erfolgreichen Assemblierung senken:

1. Wie in Abbildung 12.1 dargestellt wird nur die Assemblierung der längsten, nicht zyklischen Sequenz von Reaktionen berücksichtigt. Diese Sequenz zeichnet sich dadurch aus, dass alle Reaktionen, mit Ausnahme jener am Anfang des Datenflusses, als eines der Edukte das Produkt einer anderen Reaktion der Sequenz benötigen. Das Produkt der letzten Reaktion, dieser längsten im Individuum codierten Sequenz, stellt das Ergebnis der Berechnung dar. Es handelt sich vor allem dann um eine Vereinfachung, wenn weitere nicht überlappende Sequenzen in den evolvierten Individuen die annähernd gleiche Länge haben.

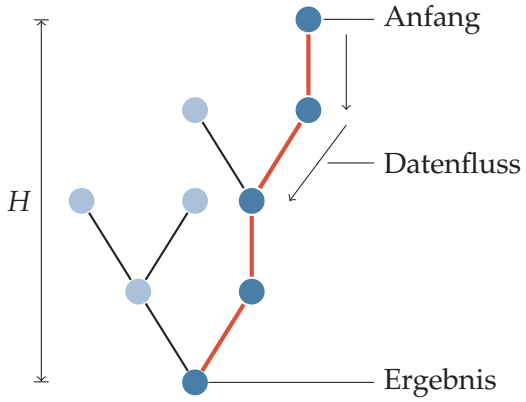


Abbildung 12.1.: Bei der Berechnung der Wahrscheinlichkeit für die vollständige Assemblierung wird ausschließlich der längste Pfad berücksichtigt. Dieser ist hier hervorgehoben, seine Länge entspricht der Höhe H des Datenflussbaumes. Jeder Knoten entspricht einer Reaktion, deren erfolgreiche Ausführung davon abhängt, dass auch die vorherige Reaktion im Datenfluss erfolgreich ausgeführt wurde. Das Produkt der letzten Reaktion enthält das Ergebnis der Berechnung.

2. Die zweite Vereinfachung besteht in der Annahme, dass der Zustand eines jeden in dieser Sequenz benötigten Moleküls von genau einer Reaktion verändert wird. Letzteres ist nach der Erzeugung eines Individuums durch die homologe Rekombination zunächst gewährleistet, kann aber durch anschließende Mutation des Produkts einer Reaktion wieder zerstört werden.

Eine Reaktion ist nur dann zielführend, wenn ihre Edukte, falls notwendig, durch entsprechende Reaktionen gesetzt wurden. Ist im Folgenden von einer Teilsequenz die Rede, so beginnen diese daher immer am Anfang der Gesamtsequenz (vgl. Abb. 12.1).

Die Berechnung der Wahrscheinlichkeit für die vollständige Assemblierung des Datenflusses geschieht mittels dynamischer Programmierung und basiert auf der Anzahl q_h^t möglicher Folgen von Reaktionen, die genau zum Zeitpunkt t eine Teilsequenz des Datenflusses der Länge h assemblieren, der in einem Blatt des Datenflussgraphen beginnt. In der Chemie sind insgesamt n Reaktionen codiert. Die Anzahl an Reaktionsfolgen, die die Teilsequenz der Länge h zu einem bestimmten Zeitpunkt assemblieren, lässt sich aus zwei überlappenden Teilproblemen berechnen. Alle q_h^{t-1} Reaktionsfolgen, die den Datenfluss der Länge h genau im vorherigen Zeitpunkt assembliert haben, enden auf der letzten hierfür nötigen Reaktion. Bei den $(n-1)$ anderen Reaktionen wäre dieser nicht assembliert worden und könnte jeweils im aktuellen Schritt vervollständigt werden. Hinzu kommen jene q_{h-1}^{t-1} Reaktionsfolgen, bei denen erst im vorherigen Zeitschritt die Teilsequenz der Länge $h-1$ assembliert wurde, diese können nun durch die entsprechende Reaktion zur Teilsequenz der Länge h erweitert werden. Die Anzahl Reaktionsfolgen, die *im*

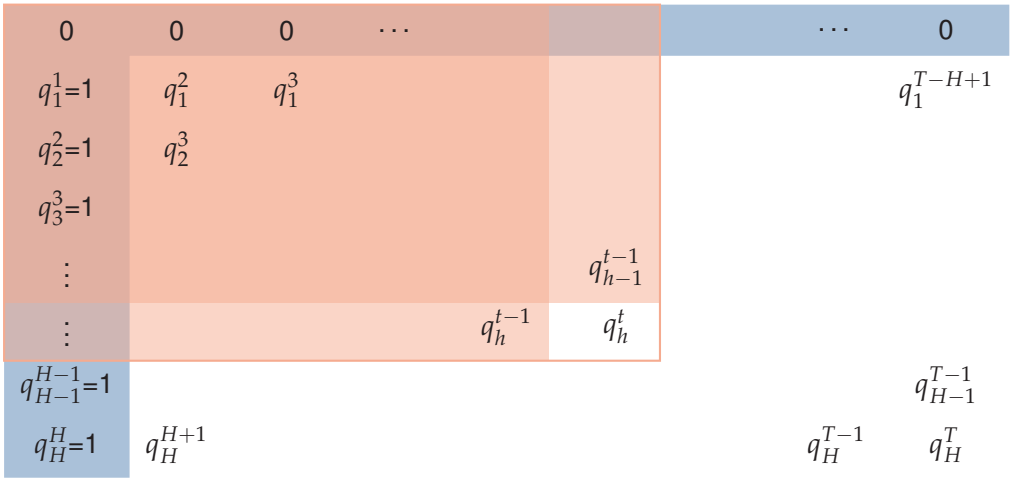


Abbildung 12.2.: Memoizationsmatrix für die Ermittlung der Assemblierungswahrscheinlichkeit. Die blau markierten Bereiche zeigen an, welche Werte als Teil der Initialisierung gesetzt werden, die restlichen Bereiche der Matrix werden anschließend bottom-up berechnet. Der Wert für q_h^t setzt sich aus den Teillösungen q_h^{t-1} und q_{h-1}^{t-1} zusammen. Die überlappenden Bereiche der in diese beiden Teillösungen wiederum eingehenden Teillösungen, sind rot markiert.

Zeitpunkt t die Teilsequenz der Länge h erfolgreich assemblieren, beträgt damit:

$$q_h^t = \begin{cases} 0 & \text{falls } h = 0 \text{ oder } t < h, \\ 1 & \text{falls } h = t,\ h > 0, \\ q_h^{t-1} \cdot (n-1) + q_{h-1}^{t-1} & \text{sonst.} \end{cases}$$

Die erste Möglichkeit eine (Teil-)Sequenz der Länge h zu assemblieren besteht zum Zeitpunkt $t = h$, davor ist die Anzahl Reaktionsfolgen, die die gesuchte Sequenz enthalten null. Die Angabe für $h = 0$ schaffen die Rahmenbedingungen für die Berechnung bei einer Sequenzlänge von $h = 1$.

Hiermit wird nun in einem bottom-up orientierten Ansatz die Anzahl der Reaktionsfolgen berechnet, die bis zum Zeitpunkt T einen Datenfluss der Höhe H assemblieren. Zur Memoization von Zwischenergebnissen dient dabei eine Matrix der Größe $(T - (H-1)) \times (H+1)$. Die Matrix sowie die Platzierung der Zwischenergebnisse sind in Abb. 12.2 dargestellt. Die Einträge in der ersten Zeile werden mit dem Wert null initialisiert, die verbleibenden Matrixeinträge der ersten Spalte werden mit eins initialisiert. Anschließend werden die restlichen Werte in der Matrix zeilen- oder spaltenweise berechnet.

Alle Einträge $q_H^{H \leq t \leq T}$, die für die Entstehung des vollständigen Datenflusses zu einem Zeitpunkt t stehen, gehen in die Berechnung der Wahrscheinlichkeit für die vollständige Assemblierung ein. Wurde der Datenfluss im Zeitpunkt $H \leq t < T$ (frühzeitig) vollständig assembliert, so können im Anschluss n^{T-t} unterschiedliche Reaktionsfolgen der Länge $T - t$ folgen, ohne das am Ende des Datenflusses stehende Ergebnis zu ändern. Grund hierfür ist die Annahme, dass kein Molekül

gegeben : Anzahl n unterschiedlicher Moleküle/Reaktionen, Höhe H des zu assemblierenden Individuums und die dafür zur Verfügung stehende Zeit T
gesucht : Wahrscheinlichkeit P für die vollständige Assemblierung

```
// Matrix Initialisierung
for i = 1 to H do
    mat [i,0]=1
end
for i = 0 to T- H do
    mat [0,i]=0
end
// Matrix Berechnung
for i = 1 to H do
    for j = 1 to T- H do
        mat [i,j]=mat [i,j-1] · (n-1) + mat [i-1,j]
    end
end
// Häufigkeit der vollständigen Assemblierung
Q = 0
for i = 0 to T- H do
    Q = Q + mat [H,i] ·n^(T−H−i)
end
// die Assemblierungswahrscheinlichkeit
P = n^(−T) · Q
```

Algorithmus 12.1 : Algorithmus zur Berechnung einer oberen Schranke für die Assemblierungswahrscheinlichkeit.

durch mehr als eine Reaktion verändert wird. Damit beträgt die Anzahl aller Reaktionsfolgen die *bis zum Zeitpunkt T* den Datenfluss assemblieren:

$$Q(T,H) = \sum_{t=H}^{T} q_H^t n^{T-t}.$$

Bei insgesamt n^T möglichen Reaktionsfolgen beträgt die Wahrscheinlichkeit für eine Assemblierung innerhalb der zur Verfügung stehenden Zeit:

$$P(T,H) = \frac{1}{n^T} Q(T,H).$$

Der Algorithmus zur Berechnung als Ganzes ist noch einmal im Alg. 12.1 dargestellt.

Diese Erkenntnis kann nun genutzt werden, um den Anstieg der Erfolgsrate beim Parityproblem in Abb. 11.3 bei Erhöhung des Vielfachen von $c = 10$ auf $c = 15$ zu erklären. Abbildung 12.3 zeigt dafür zunächst den Einfluss von Ausführungszeit

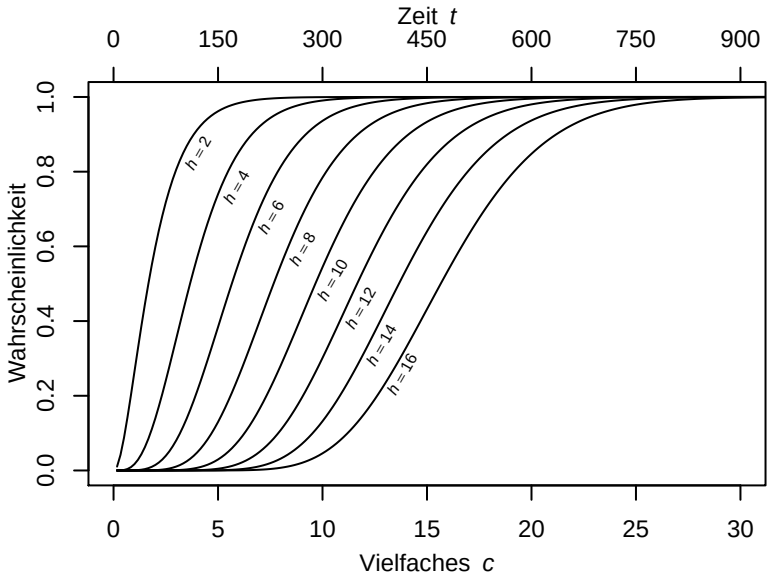


Abbildung 12.3.: Wahrscheinlichkeit der vollständigen Assemblierung eines Individuums in Abhängigkeit der Ausführungsdauer t und der Höhe h des effektiven Codes des zu assemblierenden Individuums. Die untere Achse gibt zusätzlich die Wahl eines Vielfachen c an, dabei wurde mit einer Individuengröße von 30 Reaktionen gerechnet.

und Datenflusshöhe auf die Assemblierungswahrscheinlichkeit. In Abbildung 11.5b lässt sich beobachten, dass die Populationen der meisten Systemvarianten auf ihrem Weg zur optimalen Lösung Individuen hervorbringen, deren Datenfluss eine Höhe von 10-12 aufweist. Bei einem Vielfachen von $c = 10$ ist dieses nicht der Fall, da die Wahrscheinlichkeit für die vollständige Assemblierung eines Individuums dieser Höhe im Bereich zwischen 30% und 54% liegt (vgl. Abb. 12.3). Hier nimmt die Höhe im Durchschnitt maximal einen Wert zwischen 8 und 10 an. Bereits für das Vielfache $c = 15$ steigt diese Wahrscheinlichkeit Individuen der Höhe 8–10 vollständig zu assemblieren auf Werte zwischen 82% und 93%. Auch bei einem Wert von $c = 10$ werden optimale Lösungen für diese Instanz des Parityproblems evolviert. Allerdings scheinen die hierbei zwischenzeitlich evolvierten Individuen mit Datenflüssen geringerer Höhe schlechtere Ausgangspunkte für die Evolution der optimalen Lösung zu sein, sodass hier eine deutlich geringere Erfolgsrate erzielt wird.

Die Höhe des perfekten Individuums (Fitness gleich null, Werte siehe Tab. 11.5) liegt für alle Varianten im Median in dem Bereich, bei der die Wahrscheinlichkeit der vollständigen Assemblierung deutlich über 90% beträgt. Dieses ist für $c = 10$ die Höhe 6 ($P(10 \cdot 30, 6) = 94\%$) und für $c = 15$ die Höhe 8 ($P(15 \cdot 30, 8) = 98\%$). Für größere Werte von c erreicht die Individuenhöhe einen Wert von 10 ($P(20 \cdot 30, 10) = 99.6\%$).

Auch beim Sinusproblem kann der Einfluss des Vielfachen c auf die Höhe der Individuen beobachtet werden (Abb. 11.6b). Während für ein Vielfaches von $c = 10$

der Median der Individuenhöhe im Maximum nicht über 8 steigt, nimmt dieser für $c \geq 15$ Werte zwischen 9 und 11 an. Die Tabelle 11.5 zeigt, dass die Individuen für das Sinusproblem nicht so hoch ausfallen, wie für das Parityproblem. Zum einen kann der Höhenunterschied problembedingt sein, zum anderen kann der Grund in dem Bedarf an einer höheren Assemblierungswahrscheinlichkeit liegen. Denn während der Einfluss einer einzelnen misslungenen Assemblierung auf die Fitness beim Parityproblem durch den Kehrwert der Trainingsmengengröße beschränkt ist, ist dieser beim Sinusproblem beliebig groß, da hier die Fitness über die mittlere quadratische Abweichung zur Sinusfunktion definiert ist.

13. Zusammenfassung und Ausblick

Zusammenfassung

Aus der Motivation werden drei wünschenswerte Eigenschaften für die Repräsentation einer Lösung abgeleitet: (asynchrone) Parallelität, Nichtdeterminismus und die Notwendigkeit zur Selbstorganisation der Komponenten (Kap. 4.1). Dieses sind Eigenschaften von Systemen mit denen sich die ALife-Forschung beschäftigt und führt zum Teilgebiet der künstlichen Chemie. Als zusätzlicher Wunsch kommt hinzu, die Repräsentation aus einer etablierten Repräsentation mit möglichst wenig Änderungen abzuleiten und die zugrunde liegende Repräsentation später als Referenz heranzuziehen. Unterschiede sind damit leichter auf die Änderungen zurückzuführen. Als Referenz wird die lineare genetische Programmierung verwendet. Der Zugriff auf die Instruktionen wird abstrahiert, sodass die gewählte Instruktion von einer Verteilungsfunktion bestimmt wird. Diese Funktion ist über eine Zeitkomponente (einen Programmzeiger) parametrisiert (Kap. 4.2.2). Eine diskrete, zeitabhängige Verteilung sorgt für ein Verhalten wie bei linearen Programmen. Eine Gleichverteilung führt zur zufälligen Ausführungsreihenfolge, dem Verhalten algorithmische Chemien. Zum Erlernen algorithmischer Chemien wird, wie im Referenzsystem, die genetische Programmierung verwendet (Kap. 5). Der Ablauf lässt sich hier weitestgehend übernehmen. Die Rekombination wird auf die 1-Punkt-Rekombination des linearen GP unter Verwendung der geänderten Verteilungsfunktion zurückgeführt (Kap. 5.3). Dadurch werden algorithmische Chemien bei der Rekombination zufällig gemischt.

Algorithmische Chemien verhalten sich bei der Ausführung nichtdeterministisch. Das Ausmaß ihrer Undeterminiertheit wird über die Unbestimmtheit bei der Molekülzustandsbestimmung und die Anzahl der Datenflusszweige, die aufgrund von Zyklen nicht vollständig assembliert werden können, bewertet (Kap. 6.1-6.3). Neben der Güte der Lösung als zentrales Kriterium ist auch die Komplexität gemessen an der effektiven Größe und Höhe der Datenflüsse von Interesse (Kap. 6.4).

Nach Ableitung der Systeme zur Evolution algorithmischer Chemien auf der einen und linearer Programme auf der anderen Seite aus einem gemeinsamen Kern werden die Aufgabenstellungen (Problemdesign) spezifiziert und die beiden Systeme für diese Aufgabenstellung parametrisiert (Algorithmendesign). Als Problemstellung wird die symbolische Regression der Sinusfunktion, die Evolution der booleschen Parityfunktion und mit der Thyroidfunktion ein realweltliches, medizinisches Klassifikationsproblem herangezogen (Kap. 8).

Das unabhängige Design beider Algorithmen bildet die Grundlage für einen fairen Vergleich. Für diese Aufgabe wird die sequenzielle Parameteroptimierung

(SPO) implementiert (Kap. 7). Sie ermöglicht es, beide Systeme mit demselben Aufwand und Ausmaß an algorithmenspezifischem Vorwissen zu optimieren. Die Durchführung der Parameteroptimierung bedarf zweier Änderungen. Die erste Änderung betrifft die Ausführung der algorithmischen Chemie. Das für die Parameteroptimierung als notwendig erachtete Budget von 1000-2000 Evolutionsläufen pro Problemstellung würde zu einem sehr hohen Aufwand führen. Dieses gilt vor allem für die Optimierung des AC-Systems, bei der die Auswertung jedes Individuums mit einer Vielzahl zufällig ausgeführter Reaktionen verbunden ist. Für die Ausführung algorithmischer Chemien während der SPO wird daher die Totalsynthese entwickelt (Kap. 5.4). Sie führt algorithmische Chemien mithilfe vereinfachender Annahmen beschleunigt aus.

Die zweite notwendige Änderung betrifft das Verfahren zur Parameteroptimierung. Die SPO erweist sich zunächst für das AC-System als nicht erfolgreich. Die dem SPO zugrunde liegenden Techniken stammen aus dem Bereich des Designs und der Analyse deterministischer Algorithmen und wurden um statistische Verfahren erweitert, um dem Nichtdeterminismus der zu optimierenden Algorithmen Rechnung zu tragen. Die Genauigkeit des zugrunde liegenden Modells wird dabei schrittweise am bisher besten betrachteten Designpunkt verfeinert, zudem wird ein neuer Designpunkt hinzugefügt. Die hohe Varianz der Resultate an den Designpunkten führt dazu, dass der SPO ein ungenaues Modell zugrunde liegt. Die Bestimmung vielversprechender weiterer Designpunkte wird dadurch erschwert. Die SPO wird um ein Rangbildungs- und Selektionsverfahren erweitert, mit dem Ziel, vielversprechende Bereiche des Parameterraums schneller zu identifizieren und die Auflösung des Modells hier zu verbessern (Kap. 7.5.2). Die SPO-Läufe werden damit auch für das AC-System erfolgreich durchgeführt.

Für die bestimmten Problemdesigns werden unter Verwendung der gefundenen Algorithmendesigns Lösungen evolviert. Dieses gelingt mit der Repräsentation von Lösungen als lineares Programm zunächst deutlich besser als mit der Repräsentation als algorithmische Chemie (Kap. 9.1-9.3). Die Ordnungsmaße zeigen, dass die Evolution die Reproduzierbarkeit des Verhaltens algorithmischer Chemien mit besser werdender Fitness erhöht (Kap. 9.4). Während die LGP-Läufe die von ihnen bekannte Tendenz zum Bloat zeigen, bleibt die absolute Größe der algorithmischen Chemien nahezu konstant. Auch erweisen sich die evolvierten linearen Individuen bei gleicher Fitness als deutlich komplexer, bezogen auf ihre effektive Größe und Höhe (Kap. 10).

Die Parameteroptimierung zeigt die Bevorzugung sehr geringer Rekombinationsraten bei der Evolution algorithmischer Chemien. Damit weist sie darauf hin, dass der aus der 1-Punkt-Rekombination abgeleitete Rekombinationsoperator für algorithmische Chemien ungeeignet ist. Weitere Experimente zeigen, dass die Verwendung des Rekombinationsoperators in dem von der SPO bestimmten Ausmaß keinen Einfluss auf den Evolutionsverlauf hat (Kap. 11.2). Eine Diskussion über die Notwendigkeit der Rekombination in der genetischen Programmierung (Kap. 11.1) führt zusammen mit der Ungewissheit, in wiefern das Fehlen dieses Operators die Evolution algorithmischer Chemien negativ beeinflusst, zur Umsetzung eines

weiteren Rekombinationsoperators. Dieser basiert auf dem Gedanken der Homologie, der Identifikation austauschbarer Komponenten in beiden Individuen (Kap. 11.3). Das Design des so veränderten Algorithmus weist eine deutlich höhere Rekombinationsrate auf und deutet auf eine stärkere Einflussnahme der neuen Rekombination auf die Evolution hin. Die Evolution algorithmischer Chemien weist nun keinen oder nur einen geringen Performanceunterschied zur Evolution linearer Programme auf.

Bei den neuen Experimenten lässt sich der Einfluss der pro Individuum und Fallbeispiel ausgeführten Reaktionen auf die Höhe der evolvierten Lösungen beobachten. Die Höhe nimmt mit der Anzahl ausgeführter Reaktionen ab. Mit ihr sinkt auch der Evolutionserfolg. Beide Einflüsse und ihr Zusammenhang lassen sich über die Wahrscheinlichkeit erklären, mit der die in einer Chemie kodierte Lösung in Abhängigkeit ihrer Höhe und der Anzahl ausgeführter Reaktionen assembliert werden kann. Die Berechnung dieser Assemblierungswahrscheinlichkeit wird abschließend hergeleitet (Kap. 12).

Ausblick

Totalsynthese Die Totalsynthese wurde eingeführt, um das Verhalten algorithmischer Chemien bei einer großen Anzahl ausgeführter Reaktionen anzunähern, ohne den damit verbundenen Aufwand zu erzeugen. Sie reduziert den Evolutionsaufwand während der SPO. Bei der Umsetzung der Totalsynthese werden zwei Annahmen gemacht. Zum Einen, dass eine unendliche Laufzeit simuliert werden soll, zum Anderen werden die Zustände der Moleküle, bei denen eine Zykluseintritt stattfindet, nach dem ersten Durchlauf auf den Wert 0 gesetzt. Die Totalsynthese erfüllt ihre ursprüngliche Aufgabe: Bei reduziertem Aufwand für die Experimente während der SPO lassen sich die gefundenen Designs auf die Evolution algorithmischer Chemien ohne Totalsynthese übertragen. Eine hohe Anzahl ausgeführter Reaktionen anstelle der Totalsynthese ist die Voraussetzung, um vergleichbarer Resultate zu erzielen. Die unter Totalsynthese gefunden Lösungen weichen in ihren Eigenschaften deutlich von Lösungen ab, die ohne Totalsynthese evolviert wurden. Sie sind, bezogen auf ihre Höhe und Größe, deutlich komplexer und weisen zudem mehr Zykleneintritte auf.

Ein Mechanismus ähnlich der Totalsynthese, mit gesteigerter Übereinstimmung zur zufälligen Ausführung, ist aus zwei Gründen wünschenswert. Zunächst ist die Evolution von algorithmischen Chemien ohne Totalsynthese dann besonders erfolgreich, wenn die Anzahl ausgeführter Reaktion bei der Betrachtung eines Fallbeispiels hoch ist. Es wird davon ausgegangen, dass der Grund hierfür in der steigenden Assemblierungswahrscheinlichkeit liegt. Die verringerten Evolutionserfolge bei einer kleinen Anzahl ausgeführter Reaktionen können aber auch auf eine unzureichende Parametrisierung zurückzuführen sein. Während der SPO wurde schließlich eine unendliche Anzahl ausgeführter Reaktionen durch die Totalsynthese simuliert und die verwendeten Einstellungen auf dieser Grundlage gefunden.

Die Parametrisierung des Systems für die Evolution algorithmischer Chemien unter Verwendung einer geringeren Anzahl ausgeführter Reaktionen ist aufwendig. Mit der Totalsynthese kann hierfür das Verhalten bisher nicht nachgebildet werden, sodass bei jeder Evaluation die entsprechende Anzahl Reaktionen gezogen und ausgeführt werden muss. Die effiziente Annäherung des Verhaltens bei endlicher Anzahl ausgeführter Reaktionen wäre eine wünschenswerte Eigenschaft für eine Alternative zur bisherigen Totalsynthese. Zudem könnte ein Mechanismus, bei dem die Eigenschaften der evolvierten Individuen erhalten bleiben, auch außerhalb der SPO zur Evolution herangezogen werden.

Ein Verfahren, das die Ausführung einer vorgegebenen, endlichen Anzahl Reaktionen effizient annähert, wird im Folgenden skizziert. Ausgehend von dem Molekül, dessen Zustand als Endergebnis interpretiert wird, erfolgt der Aufbau des Datenflusses. Dies gleicht der Totalsynthese. Zusätzlich wird aber die Zeit simuliert, die jeweils bis zur Ausführung einer geeigneten Reaktion vergangen ist. Ist α ein Molekül, dessen Zustand in das Endergebnis eingeht, $\mathcal{S}_\alpha$ die Reaktionsmultimenge, die potenziell den Zustand des Moleküls α bestimmen, und $\mathcal{R}$ die Multimenge aller Reaktionen in der Chemie. Dann beträgt die Wahrscheinlichkeit, dass der Zustand von α durch eine Reaktion aus $\mathcal{S}_\alpha$ im Zeitschritte t, gemessen von einem beliebigen Zeitschritt an, bestimmt wird:

$$\frac{|\mathcal{S}_\alpha|}{|\mathcal{R}|} \cdot \left(1 - \frac{|\mathcal{S}_\alpha|}{|\mathcal{R}|}\right)^{t-1}.$$

Dieses kann genutzt werden, um den Zeitbedarf t_α zu simulieren, der in einem Zweig benötigt wird, um den Zustand von α zu bestimmen. Ist der simulierte Zeitbedarf kleiner als die noch zur Verfügung stehende Zeit, so wird gleichverteilt eine Reaktion $r \in \mathcal{S}_\alpha$ für die Belegung von α gewählt. In der Rekursion wird nun die Belegung der Edukte von r betrachtet. Die zur Verfügung stehende Zeit wird dabei um t_α reduziert. Dieses Vorgehen stellt in sofern eine Vereinfachung dar, als dass der in parallelen Zweigen entstandene Aufwand unabhängig voneinander betrachtet wird.

Epigenetische Variation und Lernen Bei jeder Ausführung wird das Genom des Individuums erneut exprimiert. Die daraus resultierenden Phänotypen können dabei stark variieren, das System zeigt eine Form epigenetischer Variation. Bisher findet keine Nutzung dieser Variation statt, jede Variante wird auf einem einzelnen Fallbeispiel ausgewertet. Die Güte des Individuums ergibt sich aus dem Mittel der einzelnen Auswertungen. Die Evolution algorithmischer Chemien unterscheidet sich in ihrem Umgang mit der Variation von anderen Verfahren mit nichtdeterministischer Genotyp-Phänotyp-Abbildung. Bei Estimation-of-Distribution Algorithmen beispielsweise werden die besten Varianten selektiert und beeinflussen dann den weiteren Verlauf des Algorithmus.

Bei der Evolution algorithmischer Chemien könnte das Potenzial eines Individuums berücksichtigt werden, indem gute Varianten bei der Fitnessbewertung

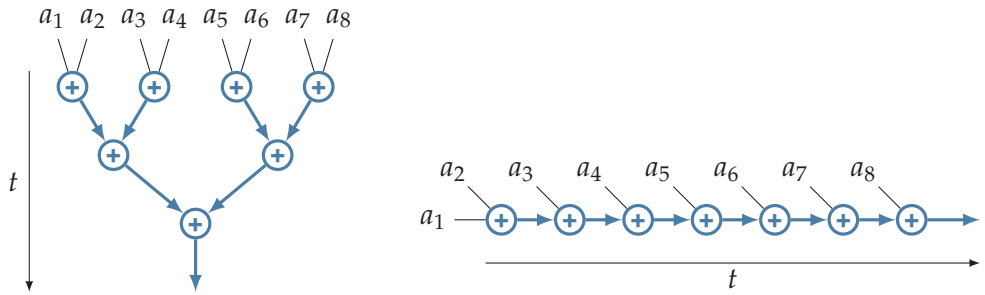
einen stärkeren Einfluss erhalten. Die primäre Hoffnung ist, dass dadurch die Evolution schneller zu guten Lösungen konvergiert. Aber es existieren weitere Aspekte, die die Berücksichtigung der Variation interessant erscheinen lassen. So wurde vermutet, dass die Akkumulation von Reaktionen unter anderem deshalb nicht zustande kommt, da dann eine positive Mutation in einer Reaktion r bei der Ausführung der Chemie nur selten zum Tragen kommt. Der Grund hierfür ist, dass die Reaktion r mit den nicht mutierten Varianten um die Ausführung konkurriert. Diese Konkurrenz reduziert die Flexibilität, d. h. die Empfänglichkeit für evolutionäre Änderungen. Eine Fitnessbewertung, die das Potenzial eines Individuums betont, könnte diese Flexibilität wiederherstellen, indem positive Änderungen besser wahrgenommen werden.

Der Umstand, dass einzelne Varianten je nach Fitnessfunktion starken Einfluss auf die Fitness der Individuen nehmen können, lässt zudem die Berücksichtigung der Variation sinnvoll erscheinen. Bei den hier betrachteten Problemstellungen macht sich ein Einfluss besonders beim Sinusproblem negativ merkbar. Die Fitness eines Individuums berechnet sich dabei aus der mittleren quadratischen Abweichung zwischen dem gesuchten und dem von der algorithmischen Chemie berechneten Wert. Eine einzelne ungeeignete Variante kann damit zu einer schlechten Bewertung guter Individuen führen. Die Berücksichtigung der Variation kann hier der robusteren Bewertung der Individuen dienen.

Anstatt durch epigenetische Variation verursacht, kann die entstehende Variation auch als Verhaltensvariante eines Individuums betrachtet werden. Diese Sichtweise wäre die Grundlage für Ansätze, bei denen die Individuen aus Feedback auf ihr exploratives Verhalten lernen. Reaktionen, die an erfolgreichen Betrachtungen eines Fallbeispiels teilhaben, könnten beispielsweise dabei in ihrer Konzentration verstärkt, andere könnten abgeschwächt werden.

Robustheit Ein Genotyp kann auf variierende Datenflüsse als Phänotyp abgebildet werde. Für die Variation existieren zwei Quellen. Zum einen kann der Datenfluss mehrdeutig codiert sein, wenn z. B. ein Molekül von mehr als einer Reaktion als Produkt verwendet wird. Zum anderen werden bei der Ausführung nicht immer alle Teile des Datenflusses vollständig assembliert. Die Evolution kann diesem Umstand Rechnung tragen. So haben die Individuen einen Selektionsvorteil, die auch als unvollständig assemblierte Lösungen noch ein gutes Resultat bei der Fitnessberechnung erzielen. Eine entsprechende Lösung erfüllt im Ansatz auch von Neumanns [80] Forderung, Fehler als essenziellen Teil eines Prozesses zu sehen (Kap. 4.1).

Eine solche Robustheit bezüglich der Unzuverlässigkeit von Lösungskomponenten ist sicherlich in vielen Anwendungsszenarien interessant und könnte Bestandteil weiterer Untersuchungen sein. Einige Anwendungen, in denen eine solche Robustheit von Vorteil ist, wurden bereits in den Kapiteln 3 und 4 vorgestellt. Amorphous Computing etwa beschreibt eine Architektur, in der jede datenverarbeitende Einheit zunächst einmal als unzuverlässig eingeschätzt wird (Kap. 3.2.1).



(a) Parallele Addition **(b)** Serielle Addition

Abbildung 13.1.: Die Addition von n Variablen kann (a) bei maximaler Parallelität in $t = \lceil \log_2 n \rceil$ Berechnungsschritten erfolgen, rein (b) sequenziell benötigt sie $t = n - 1$ Schritte.

Es ist die Aufgabe der Organisation dieser Einheiten eine mit ihnen umgesetzte Lösung robust zu gestalten. Eine Robustheit seitens der Lösung könnte hier die Organisation entlasten. Probabilistische Bits, als weiteres Anwendungszenario, sind Informationsträger, deren Information nur mit einer Wahrscheinlichkeit kleiner 1 korrekt wiedergegeben wird. Dafür ist ihr Energiebedarf deutlich geringer als bei normalen Informationsträgern (Kap. 4.1). Als Einsatzgebiet dieser PBits werden probabilistische Verfahren und die Signalverarbeitung genannt, da sie dem nichtdeterministischen bzw. fehlerbehafteten Charakter der Information Rechnung tragen. Robuste Lösungen, die diesem Charakter ebenfalls Rechnung tragen, könnten das Einsatzspektrum der PBits auf Probleme anderer Bereiche erweitern.

Jenseits dieser Anwendungsaspekte stellt sich zudem die Frage nach dem Einfluss auf die Evolution, da die Robustheit auch den evolutionären Änderungen gelten kann.

Zeit versus Raum Instruktionen auf einem Pfad zwischen einem Blatt und der Wurzel im Datenflussbaum einer evolvierten algorithmischen Chemie müssen zeitlich aufeinander folgend ausgeführt werden. Unterschiedliche Zweige hingegen können parallel ausgeführt werden. Die parallele Ausführung erfordert die Existenz von mehr als einer datenverarbeitenden Einheit. Diese Einheiten sind räumlich voneinander getrennt. Beim Spatial Computing wird davon gesprochen, dass Berechnungen von der Zeit in den Raum verlagert werden. Abbildung 13.1 soll das am Beispiel der Addition von acht Variablen $a_1, \ldots, a_8$ verdeutlichen.

Die Evolution algorithmischer Chemien meidet es, Berechnungen in die Zeit, d. h. in die Höhe des Datenflussbaums, zu verlagern. Dieses haben sowohl die experimentell erhobenen Daten, als auch die Herleitung der Assemblierungswahrscheinlichkeit gezeigt. Eine stärkere Integration zeitlicher Zusammenhänge kann aber durchaus wünschenswert sein, wofür im Folgenden zwei Beispiele genannt werden. Im Anschluss wird skizziert, wie algorithmische Chemien um einen regulierbaren, zeitlichen Aspekt ergänzt werden können.

Die zeitliche Abfolge von Instruktionen ist in den meisten Repräsentationen

explizit codiert. Sie stellt eine weitere Information dar, die evolviert wird. Damit ist der Ausführungszeitpunkt ein zusätzlicher Freiheitsgrad für die Evolution. Bei den Instruktionssequenzen linearer Individuen stellt der zeitliche Ablauf sicher, dass ein Registerinhalt von der zuletzt auf das Register schreibenden Instruktion bestimmt wird. Dies erleichtert Phänomene wie den Bloat, können doch beliebig viele Instruktionen vor der benötigten Instruktion in ein Register schreiben, ohne das Endergebnis zu verändern. Somit nutzt die Evolution den über den Faktor Zeit geschaffenen, zusätzlichen Freiheitsgrad in Form des Bloats für ihre Zwecke aus. Bei den hier betrachteten Problemstellungen gelingt die Evolution algorithmischer Chemien ohne evolutionäre Anpassung einer zeitlichen Abfolge. Dennoch kann nicht ausgeschlossen werden, dass ein solcher Freiheitsgrad für andere Problemstellungen von Nutzen ist.

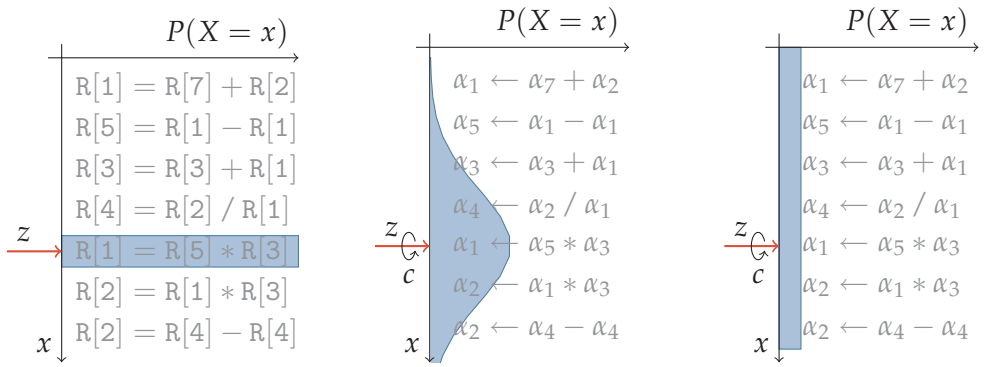
Die Evolution algorithmischer Chemien meidet die sequenzielle Abhängigkeit von Instruktionen. Darin ähnelt sie Ansätzen aus dem Bereich des Spatial Computing. Nicht zuletzt ist auch die Betrachtung algorithmischer Chemien durch den Wunsch motiviert, Algorithmen für diese Systeme zu evolvieren. Ansätze aus dem Bereich des Spatial Computing benötigen aber zunächst einen hinreichend großen Raum. Für das BLOB Computing bedeutet dies beispielsweise, dass sich genügend Partikel auf dem Array der datenverarbeitenden Elemente befinden müssen. Beim Amorphous Computing wird die Größe des Raums durch die Anzahl der Recheneinheiten im Medium bestimmt. Steht der erforderliche Raum zur Verfügung, so muss er effizient nutzbar zu machen sein, um in ihm effizient Berechnungen durchführen zu können. Păun [94] stellt für seine P-Systeme fest:

> Because P systems are parallel computing devices, it is expected that they can solve hard problems in an efficient manner, and this expectation is confirmed for systems provided with ways for producing an exponential workspace in linear time.

Entspricht entweder der zur Verfügung stehende Raum nicht den Erfordernissen oder kann er nicht effizient nutzbar gemacht werden, so kann es nützlich sein, Teile der Berechnung zu sequenzialisieren.

Die Ausführung algorithmischer Chemien wurde bereits im Hinblick darauf formuliert, die Reaktionen um zeitliche Informationen zu erweitern, um so sequenzielle Abhängigkeiten zu codieren. Der Einfluss der Zeit auf die Ausführung soll regulierbar sein. Eine mit dem Programmzeiger parametrisierte Wahrscheinlichkeitsverteilung wandert bei der verwendeten Betrachtungsweise über eine Liste von Instruktionen bzw. Reaktionen (Abb. 13.2). An jeder Position des Programmzeigers werden c Reaktionen entsprechend dieser Verteilung entnommen und ausgeführt. Die in Gl. (5.1) beschriebene diskrete Verteilung mit $c = 1$ erzeugt das Verhalten eines linearen Programms. Wird eine Gleichverteilung herangezogen, so entspricht das Verhalten dem einer algorithmischen Chemie.

Diese Verteilung kann nun durch eine (diskrete) Normalverteilung ersetzt werden, deren Mittelwert der Position des Programmzeigers entspricht und deren Varianz σ ein neuer Parameter des Systems ist (Abb. 13.2b). Für $\sigma \to \infty$ entspricht



(a) Lineares Programm: Zeit bestimmt Instruktion

(b) beschränkter Einfluss der Zeit

(c) Algorithmische Chemie: Zeit ohne Einfluss

Abbildung 13.2.: Einfluss der Zeit auf die ausgeführte Reaktion bzw. Instruktion. Die Bilder links und rechts sind bereits aus Kapitel 4 bekannt. Während bei (a) linearen Programmen ausschließlich die Zeit in Form des Programmzeigers die Wahl der Instruktion beeinflusst, ist bei (c) algorithmischen Chemien die Wahl der Reaktionen unabhängig vom aktuellen Zeitpunkt. Eine Möglichkeit der Erweiterung besteht in der Wahl anderer Verteilungen, die für die Zuordnung zwischen Zeit und Instruktion/Reaktion zuständig ist. Bei der Wahl einer (b) Normalverteilung hat die Zeit einen begrenzten Einfluss auf die Wahl der Instruktion ohne sie vollständig zu determinieren.

das Verhalten dem der algorithmischen Chemie. Mit kleiner werdenden σ steigt der Einfluss der Zeit (Programmzeigerposition) auf die Wahl der nächsten Instruktion. Dieser Einfluss kann dann auch von der Evolution genutzt werden. Für $\sigma \to 0$ verschwindet der Zufall aus der Wahl der Instruktionen und einzig die Zeit bestimmt die auszuführende Instruktion. Der Einfluss auf die evolvierten Lösungen bleibt zu untersuchen. Aufgrund zuverlässiger Zeitinformationen wird erwartet, dass die evolvierten Lösungen für $\sigma \to 0$ zu mehr Sequenzialität tendieren.

Rechnerarchitektur Da keine Synchronisation zwischen den Instruktionen notwendig ist, kann die Ausführungsgeschwindigkeit algorithmischer Chemien beschleunigt werden, indem Reaktionen parallel gewählt und ausgeführt werden. Die einfache Skalierbarkeit und weitere im Folgenden vorgestellte Aspekte lassen eine Rechnerarchitektur auf der Grundlage des Reaktors als interessanten Gegenstand für weitere Betrachtungen erscheinen. Zunächst wird diese Architektur zum besseren Verständnis kurz skizziert, Abb. 13.3a stellt sie schematisch dar.

Den Kern der Architektur bildet eine beliebige Anzahl arithmetisch-logischer Einheiten (ALU[1]). Sie erhalten zufällig gewählte Reaktionen aus einem gemeinsamen Programmspeicher. Die ALUs bekommen die für die Ausführung benötigten Daten (Edukte der Reaktionen) aus einem gemeinsamen Datenspeicher, der die Zustände der Moleküle verwaltet. Die Ergebnisse der Berechnung (Produkte der

[1] arithmetic logical unit

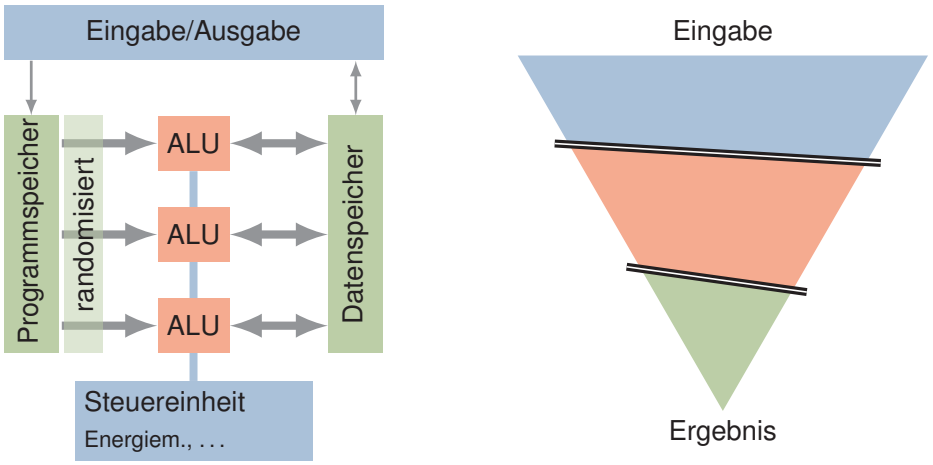


Abbildung 13.3.: Schemazeichnung einer (a) Rechnerarchitektur für die Ausführung algorithmischer Chemien. Die Effizienz könnte durch (b) einen Pipelinebetrieb gesteigert werden. Bei ihm befinden sich die Zwischenergebnisse unterschiedlicher Eingaben, hier farblich dargestellt, zeitgleich in der Bearbeitung.

Reaktionen) werden ebenfalls wieder in diesem Speicher abgelegt. Die ALUs können vollkommen asynchron laufen, die dargestellte gemeinsame Steuereinheit dient lediglich erweiterter Systemfunktionalität, wie z. B. dem Energiemanagement (s.u.). Ein gemeinsamer Speicher kann prinzipiell auch bei dieser Architektur für Programme und Daten dienen. Eine getrennte Darstellung findet hier aufgrund stark unterschiedlicher Zugriffsweisen statt, die sich unter Umständen in einer unterschiedlichen Konstruktion der Speicher niederschlagen. Die ALUs lesen aus dem Programmspeicher bei jedem Zugriff lediglich eine Instruktion aus, die aufgrund der zufälligen Wahl nicht adressiert werden muss. Aus dem Datenspeicher hingegen benötigen sie die Zustände definierter (adressierter) Moleküle und greifen auch schreibend auf den Speicher zu. Parallel stattfindende Lese- und Schreibzugriffe werden nicht blockiert, da die algorithmische Chemie zu keinem Zeitpunkt einen gültigen Zustand voraussetzt. Trotzdem müssen während der Ausführung gültige Zustände hergestellt werden, weshalb bei zeitlich überlappenden Schreibzugriffen ein beliebiger erfolgreich sein muss. Ein letztes Modul ist für die Ein- und Ausgabe, d. h. für die Kommunikation mit dem System zuständig. Dieses legt zunächst die Programme im Programmspeicher ab. Anschließend können Eingaben an das System durch Setzen entsprechender Molekülzustände erfolgen. Nach einer vorgegebenen Anzahl ausgeführter Reaktionen werden die Zustände bestimmter Moleküle als Ergebnis zurückgegeben.

ALUs können dem System zur Laufzeit beliebig hinzugefügt oder aus dem System entfernt werden. Dies ist möglich, da die ALUs nicht explizit miteinander kommunizieren und keine Synchronisation zwischen den auf ihnen ausgeführten

Instruktionen stattfindet. Zudem erzeugt die Entfernung einer ALU keinen störenden Datenverlust und es existiert kein ihr zugeordneter Prozess, der migriert werden müsste. Neben der Skalierung der Performance kann diese Eigenschaft zum Zwecke des Energiemanagements benutzt werden. Das System toleriert auch den störungsbedingten Totalausfall einzelner ALUs und quittiert diesen nur mit einer verlangsamten Ausführung.

Extra Mechanismen für das Scheduling von Programmen sind in diesem System nicht notwendig. Bei der Speicherung unterschiedlicher Programme muss nur sichergestellt werden, dass ihre Reaktionen disjunkte Molekülmengen benutzen. Die Programme werden anschließend parallel ausgeführt. Die Priorität eines Programms kann über seine Konzentration im Programmspeicher erhöht werden. Hierfür wird z. B. eine weitere Kopie seiner Reaktionen im Programmspeicher abgelegt.

Die Skalierbarkeit des Systems wird durch geringe Effizienz erkauft. Das liegt daran, dass viele Reaktionen durchgeführt werden, die aktuell nicht notwendig sind. So ist die Ausführung von Reaktionen nahe der Wurzel des Datenflussbaums überflüssig bis zu dem Zeitpunkt, an dem die Eingaben durch den Baum in die Edukte dieser Reaktionen propagiert wurden. Zu diesem Zeitpunkt ist die Ausführung von Reaktionen nahe den Eingaben überflüssig, hat doch die Zustandsbestimmung ihrer Produkte bereits stattgefunden. Bei der Bearbeitung ganzer Eingabeströme kann die Effizienz durch eine Betriebsweise gesteigert werden, die am ehesten den Pipelines in heutigen Prozessoren ähnelt. Dabei wird eine neue Eingabe angelegt, sobald die Zwischenergebnisse der vorherigen Eingabe weit genug durch den Baum in Richtung Wurzel propagiert wurden (Abb. 13.3b). Dieses geschieht noch bevor das Ergebnis der vorherigen Eingabe ausgelesen wird. So findet zeitgleich die Berechnung für mehrere Eingaben in der Chemie statt. Neben einem hinreichend großen zeitlichen Abstand zwischen dem Wechsel der Eingabe ist eine annähernd gleiche Anzahl Reaktionen auf jedem Pfad von einem Blatt (einem Eingabewert) zur Wurzel notwendig. Dieser kann aber durch Identitätsreaktionen, bei denen der Zustand des Edukts auf das Produkt übertragen wird, nachträglich hergestellt werden. Eine Ähnlichkeit der Werte zwischen den Eingaben verringert die Fehler bei dieser Form des Betriebs. Eine solche Ähnlichkeit der Eingaben kann dann gegeben sein, wenn die Eingaben aus der gleichen Quelle kommen, z. B. aus wiederholt erhobenen, sensorischen Daten.

Über den Autor

Von 10/1995 bis 2/2002 Studium der Ingenieurinformatik mit Anwendungsfach Elektrotechnik an der Universität Dortmund (Abschluss: Dipl.-Inform.). Von 4/2002 bis 2/2006 wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Prof. Dr. Wolfgang Banzhaf am Lehrstuhl für Systemanalyse der Universität Dortmund. Die Forschungsgebiete in dieser Zeit waren Sozionik, genetische Programmierung, Parameteroptimierung und naturanaloge Optimierung. Von 3/2006 bis 5/2006 „visiting research student“ am Fachbereich Informatik der Memorial University of Newfoundland in St. John’s, Kanada.

Publikationsliste

1. C. W. G. Lasarczyk. Trainingsmengenselektion auf der Grundlage einer Fitnesscase-Topologie. Diplomarbeit, Universität Dortmund, 2002.

2. C. W. G. Lasarczyk and T. Kron. Globale Kohärenz in sozialen Systemen. In H.-D. Burkhard, T. Uthmann, and G. Lindemann, editors, *Modellierung und Simulation menschlichen Verhaltens*, number 163 in Informatik Berichte, pages 77-91, Berlin, 2003. Professoren des Institutes für Informatik.

3. T. Kron, C. W. G. Lasarczyk, and U. Schimank. Zur Bedeutung von Netzwerken für Kommunikationssysteme - Ergebnisse einer Simulationsstudie / Double Contingency and the Relevance of Networks for Communication Systems - Results of a Simulation Study. *Zeitschrift für Soziologie*, 32(5), 2003.

4. C. W. G. Lasarczyk. Simulation menschlichen Verhaltens. *Künstliche Intelligenz*, (3), 2003.

5. C. W. G. Lasarczyk, P. Dittrich, and W. Banzhaf. Dynamic subset selection based on a fitness case topology. *Evolutionary Computation*, 12(2):223-242, 2004.

6. M. Preuß and C. W. G. Lasarczyk. On the Importance of Information Speed in Structured Populations. In X. Yao, H.-P. Schwefel, et al., editors, *Parallel Problem Solving from Nature - PPSN VIII, Proc. Eighth Int’l Conf., Birmingham*, pages 91-100, Berlin, 2004. Springer.

7. W. Banzhaf and C. W. G. Lasarczyk. Genetic programming of an algorithmic chemistry. In U.-M. O’Reilly, T. Yu, R. Riolo, and B. Worzel, editors, *Genetic*

Programming Theory and Practice II, volume 8 of *Genetic Programming*, pages 175-190. Kluwer/Springer, Boston MA, 2005.

8. W. Banzhaf and C. W. G. Lasarczyk. A new programming paradigm inspired by artificial chemistries. In J.-P. Banâtre, P. Fradet, J.-L. Giavitto, and O. Michel, editors, *Proceedings of Unconventional Programming Paradigms (UPP'04)*, volume 3566 of *Lecture Notes in Computer Science*, pages 73-83, Berlin, 2005. Springer.

9. T. Bartz-Beielstein, C. W. G. Lasarczyk, and M. Preuß. Sequential parameter optimization. In B. McKay et al., editors, *Proc. 2005 Congress on Evolutionary Computation (CEC'05), Edinburgh, Scotland*, volume 1, pages 773-780, Piscataway NJ, 2005. IEEE.

10. C. W. G. Lasarczyk and W. Banzhaf. An algorithmic chemistry for genetic programming. In M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, editors, *Proceedings of the Eighth European Conf. Genetic Programming (EuroGP'05)*, volume 3447 of *Lecture Notes in Computer Science*, pages 1-12, Berlin, 2005. Springer.

11. C. W. G. Lasarczyk and W. Banzhaf. Total synthesis of algorithmic chemistries. In H.-G. Beyer et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conf. (GECCO 2005), Washington D.C.*, volume 2, pages 1635-1640, New York, 2005. ACM.

12. C. W. G. Lasarczyk and T. Kron. Coordination in scaling actor constellations: The advantages of small-world networks. In K. Fischer, M. Florian, and T. Malsch, editors, *Socionics: Scalability of Complex Social Systems*, volume 3413 of *Lecture Notes in Computer Science*, pages 199-217. Springer, Berlin, 2005.

13. J. Mehnen, T. Michelitsch, C. W. G. Lasarczyk, and T. Bartz-Beielstein. Multi-objective evolutionary design of mold temperature control using DACE for parameter optimization. In H. Pfützner and E. Leiss, editors, *Short Paper Proc. Twelfth Int'l Symp. Interdisciplinary Electromagnetics, Mechanics, and Biomedical Problems (ISEM 2005)*, volume L11-1, pages 464-465. Vienna Magnetics Group Reports, 2005.

14. J. Mehnen, T. Michelitsch, C. W. G. Lasarczyk, and T. Bartz-Beielstein. Multi-objective evolutionary design of mold temperature control using DACE for parameter optimization. *International Journal of Applied Electromagnetics and Mechanics*. IOS (in Druck)

A. SPO-Tabellen, Effekt- und Interaktionsplots

Zunächst sind die besten Einstellungen für die GP-Systeme zur Evolution eines linearen Programms und einer algorithmischen Chemie aus den SPO-Läufe tabelliert. Tabellen A.1 (a)&(b) enhalten die Einstellungen für die Evolution einer Funktion zur Ermittlung gerader Parität für einen Input der Größe 4 (siehe Kap. 8.2), die Tabellen A.2 (a)&(b) enthalten während der SPO betrachtete Einstellungen für die Evolution einer Klassifikation für die Schilddrüsenfunktion (siehe Kap. 8.3). Anschliessend stellen die Abb. A.1 - A.4 die zusätzlich aus den Daten ermittelten Effekt der Parameter und ihrer Interaktionen dar.

Tabelle A.1.: Verlauf der sequentiellen Parameteroptimierung für die Evolution eines linearen Programms und einer algorithmischen Chemie zur Überprüfung einer 4 Bit Eingabe auf gerade Parität. Dargestellt sind die 20 besten Einstellungen, die jeweils erreichte, mittlere Fitness, die Standardabweichung so wie in den letzten Spalten die Anzahl der damit durchgeführten Experimente in den einzelnen Runden.

(a) Evolution einer algorithmischen Chemie

	Eltern	Sel.d.	Rekomb.	Mut.	Mol.	mean	stddev	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	807	4.08	0.00119	0.0227	24	0.147	0.096																			10	40	5
2	816	5.06	0.00554	0.0196	23	0.153	0.113																		10	58	21	0
3	1290	5.66	0.00524	0.0217	22	0.155	0.0953														10	59	0	0	4	0	0	0
4	970	4.61	0.00419	0.0209	23	0.157	0.0997																	10	29	14	12	0
5	865	3.35	0.0237	0.0156	24	0.164	0.12																				10	41
6	1300	5.17	0.015	0.026	21	0.167	0.0903													10	66	3	0	0	0	0	0	0
7	1490	4.93	0.00919	0.0199	22	0.17	0.113																10	70	38	0	0	0
8	1340	4.76	0.0397	0.0236	21	0.186	0.109															10	69	0	0	0	0	0
9	1660	6.01	0.0343	0.0235	22	0.186	0.107	10	9	10	1	15	2	2	6	7	0	6	7	7	0	2	0	0	0	0	0	0
10	2060	5.41	0.00439	0.0258	23	0.204	0.101				10	35	0	4	1	0	0	9	2	1	0	0	0	0	0	0	0	0
11	1790	2.98	0.00861	0.0151	29	0.207	0.109						10	29	5	19	0	3	8	0	0	0	0	0	0	0	0	0
12	1770	5.62	0.0138	0.0232	25	0.208	0.118			10	56	0	0	0	0	0	0	3	10	3	0	0	0	0	0	0	0	0
13	1610	5.41	0.071	0.025	23	0.209	0.0938					10	40	0	0	0	0	1	6	0	0	0	0	0	0	0	0	0
14	630	2.04	0.000632	0.0187	28	0.216	0.0951	10	1	9	0	0	0	0	0	0	0	6	5	3	0	0	0	0	0	0	0	0
15	2500	4.05	0.000323	0.0187	29	0.217	0.101								10	35	0	9	5	0	0	0	0	0	0	0	0	0

(b) Lineares GP

	Eltern	Sel.d.	Rekomb.	Mut.	Reg.	mean	stddev	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	165	2.93	0.438	0.00544	15	0.0321	0.057	10	0	4	8	3	3	0	0	7	0	0	0	0	8	2	3	6	6	9	1	2
2	119	4.47	0.526	0.00883	17	0.0344	0.0624											10	15	55	0	0	0	0	0	0	0	0
3	366	5.76	0.631	0.012	18	0.0344	0.0551															10	10	29	0	0	0	0
4	366	3.19	0.288	0.00631	20	0.036	0.0692									10	14	2	5	10	15	0	0	0	0	3	0	0
5	382	5.28	0.529	0.012	19	0.0367	0.071														10	47	2	4	0	0	0	0
6	384	5.33	0.36	0.0131	20	0.0367	0.0675	10	10	8	5	7	0	3	0	0	6	19	5	0	7	0	0	0	0	0	0	0
7	218	4.8	0.56	0.0105	19	0.0369	0.0696												10	1	7	18	13	10	0	7	0	0
8	232	4.7	0.271	0.0108	19	0.043	0.0782																			10	70	0
9	327	3.08	0.411	0.00589	21	0.0437	0.0742										10	32	21	0	4	0	0	0	0	3	0	0
10	100	4.92	0.382	0.0107	17	0.0472	0.0641								10	34	0	0	7	0	0	0	0	0	0	2	0	0
11	1220	3.78	0.485	0.00739	22	0.0482	0.0692	10	3	2	6	10	0	0	0	2	9	11	7	0	0	0	0	0	0	1	0	0
12	561	3.58	0.354	0.01	21	0.0503	0.0586																		10	13	2	11
13	149	4.79	0.36	0.015	21	0.0504	0.071						10	42	0	0	0	0	0	0	0	0	5	4	0	6	0	0
14	101	6.43	0.55	0.0181	19	0.0513	0.0662																				10	29
15	106	4.92	0.471	0.0145	19	0.0527	0.0781							10	42	0	3	0	0	0	0	0	11	2	0	2	0	0

Tabelle A.2.: Verlauf der sequentiellen Parameteroptimierung für die Evolution eines linearen Programms und einer algorithmischen Chemie zur Klassifikation der Schildrüsenfunktion. Dargestellt sind in beiden Fällen die 10 besten Einstellungen, die jeweils erreichte, mittlere Fitness, die Standardabweichung so wie in den letzten Spalten die Anzahl der damit durchgeführten Experimente in den einzelnen Runden.

(a) Evolution einer algorithmischen Chemie

	Eltern	Sel.d.	Rekomb.	Mut.	Mol.	mean	stddev	0	1	2	3	4	5	6	7	8	9	10
1	156	6.92	0.00314	0.0368	21	0.0165	0.00847			10	21	41	8	4	18	32	17	15
2	174	7	0.017	0.0375	30	0.0169	0.00909				10	9	0	5	43	30	46	14
3	207	6.81	0.0317	0.036	18	0.0176	0.00741						10	51	0	0	0	0
4	301	6.79	0.00796	0.0338	22	0.0191	0.0105					10	51	0	0	0	0	0
5	920	6.45	0.0718	0.0336	15	0.0206	0.00889	10	1	25	0	0	0	0	0	0	0	0
6	451	4.53	0.198	0.0345	29	0.0208	0.0123	10	17	0	0	4	0	0	0	0	0	0
7	995	4.05	0.196	0.0318	22	0.0212	0.00899	10	21	0	0	0	0	0	0	0	0	0
8	1030	6.87	0.00582	0.0354	28	0.0214	0.00831							10	0	0	0	0
9	262	5.54	0.0767	0.0201	23	0.023	0.015	10	2	31	0	0	0	0	0	0	0	0
10	313	6.5	0.0897	0.0324	20	0.0231	0.00832									10	0	0

(b) Lineares GP

	Eltern	Sel.d.	Rekomb.	Mut.	Reg.	mean	stddev	0	1	2	3	4	5	6	7	8	9	10
1	167	3.88	0.422	0.0357	19	0.0129	0.00047									10	12	0
2	263	3.89	0.381	0.0384	24	0.0141	0.00438	10	1	6	4	26	14	22	11	23	0	0
3	240	3.28	0.195	0.0394	30	0.0144	0.00445	10	0	3	8	9	29	0	0	28	0	0
4	284	3.58	0.129	0.0374	15	0.0145	0.00458					10	16	9	0	8	3	0
5	113	2.57	0.131	0.0396	18	0.0149	0.00479							10	50	0	0	0
6	124	1.7	0.206	0.0367	16	0.0149	0.00477				10	13	0	14	0	0	0	0
7	545	6.92	0.166	0.0399	15	0.0149	0.00531		10	6	13	2	0	15	0	0	0	0
8	107	3.37	0.306	0.0364	22	0.0153	0.00511								10	2	6	0
9	1610	3.04	0.18	0.0386	20	0.0155	0.0076	10	29	0	0	0	0	0	0	0	6	0
10	377	3.18	0.726	0.0372	19	0.0156	0.00535	10	0	14	0	0	0	0	0	0	7	0

Tabelle A.3.: Verlauf der sequentiellen Parameteroptimierung für die Evolution einer algorithmischen Chemie zur Überprüfung einer 4 Bit Eingabe auf gerade Parität unter Verwendung homologer Rekombination. Dargestellt sind die 15 besten Einstellungen, die jeweils erreichte, mittlere Fitness, die Standardabweichung so wie in den letzten Spalten die Anzahl der damit durchgeführten Experimente in den einzelnen Runden.

	Eltern	Sel.d.	Rekomb.	Mut.	Mol.	mean	stddev	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	470	3.81	0.259	0.0105	30	0.0298	0.0496																				10	32
2	703	6.91	0.237	0.0216	30	0.0322	0.053																10	21	16	19	2	0
3	279	4.46	0.327	0.0146	30	0.041	0.0633							10	41	6	6	9	8	1	7	1	1	0	0	0	0	0
4	450	6.03	0.248	0.0114	30	0.0412	0.0657																		10	22	12	12
5	122	6.56	0.384	0.0167	30	0.0456	0.0624												10	60	0	0	0	0	0	1	0	0
6	1460	2.82	0.23	0.0161	24	0.0488	0.0548	10	3	8	2	7	14	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1010	3.6	0.18	0.0154	30	0.0494	0.0688				10	7	9	13	6	11	4	0	0	0	0	0	0	3	10	6	0	0
8	1160	5.84	0.292	0.0204	30	0.0509	0.0734	10	16	4	0	9	0	1	0	15	24	0	0	0	0	0	0	0	0	0	0	0
9	530	6.16	0.114	0.0226	30	0.0538	0.069													10	30	6	14	8	0	0	0	0
10	441	5.62	0.195	0.0229	30	0.0553	0.0756					10	1	0	0	0	0	1	5	5	17	14	19	13	0	0	0	0
11	521	2.61	0.397	0.0109	29	0.0573	0.0767	10	0	2	10	11	8	7	2	17	6	2	5	0	0	0	0	2	2	0	0	0
12	1180	3.58	0.268	0.0132	27	0.0579	0.0746														10	47	0	15	13	0	0	0
13	1670	4.5	0.347	0.0103	29	0.0581	0.0741										10	27	39	0	0	0	0	0	3	0	0	0
14	310	4.82	0.0497	0.0145	30	0.0642	0.084		10	16	18	11	13	0	0	1	11	7	1	0	0	0	0	2	0	0	0	0
15	1220	6.53	0.202	0.0139	30	0.0681	0.0798																	10	22	21	3	0
16	307	5.54	0.272	0.00474	30	0.0741	0.0836																			10	56	0
17	845	4.03	0.334	0.0172	22	0.0785	0.0771	10	5	0	2	1	5	17	6	0	0	10	0	0	0	0	0	0	0	0	0	0
18	534	6.98	0.476	0.0246	30	0.0818	0.093															10	35	6	5	0	0	0
19	1200	2.2	0.0815	0.0112	28	0.0818	0.0885	10	8	22	12	0	0	1	0	0	0	0	1	0	7	0	0	0	0	0	0	0
20	941	6.13	0.747	0.0178	26	0.0945	0.0941	10	13	4	5	0	0	10	0	0	0	2	0	0	0	0	0	0	0	0	0	0
21	3900	4.46	0.0605	0.0212	20	0.0973	0.0728	10	7	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	960	3.91	0.151	0.0307	18	0.0997	0.0902	10	0	0	1	1	5	0	1	4	8	0	0	0	3	0	0	0	0	0	0	0
23	409	6.48	0.297	0.0208	15	0.101	0.0504			10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	2150	6.25	0.14	0.0302	23	0.106	0.0782	10	0	0	1	1	3	3	0	0	1	2	0	0	0	0	0	0	0	0	0	0
25	2860	3.06	0.283	0.00876	18	0.111	0.0818	10	5	0	0	1	0	1	0	0	0	1	5	0	0	0	0	0	0	0	0	0

Tabelle A.4.: Verlauf der sequentiellen Parameteroptimierung für die Evolution einer algorithmischen Chemie zur Approximation der Sinusfunktion unter Verwendung homologer Rekombination. Dargestellt sind die 10 besten Einstellungen, die jeweils erreichte, mittlere Fitness, die Standardabweichung so wie in den letzten Spalten die Anzahl der damit durchgeführten Experimente in den einzelnen Runden.

	Eltern	Sel.d.	Rekomb.	Mut.	Mol.	mean	stddev	0	1	2	3	4	5	6	7
1	167	6.73	0.302	0.0219	29	−2.21	0.803						10	60	30
2	122	6.82	0.332	0.0202	29	−2.15	1.02							10	15
3	116	6.3	0.258	0.0291	30	−1.96	0.636				10	58	34	0	0
4	101	6.67	0.115	0.0377	29	−1.62	0.634		10	45	26	0	0	0	0
5	421	6.37	0.238	0.0323	22	−1.57	0.656	10	39	3	31	0	0	0	0
6	2880	4.7	0.442	0.00518	16	−1.51	0.322	10	16	0	0	0	0	0	0
7	3380	5.72	0.499	0.003	23	−1.48	0.31	10	0	0	0	0	0	0	0
8	185	6.64	0.193	0.0051	29	−1.45	0.773					10	25	0	0
9	3040	6.72	0.553	0.00673	18	−1.43	0.288	10	0	0	0	0	0	0	0
10	1160	5.6	0.424	0.00016	30	−1.38	0.413	10	0	0	0	0	0	0	0
11	1830	5.36	0.31	0.014	26	−1.36	0.413	10	0	0	0	0	0	0	0
12	512	5.05	0.228	0.0398	20	−1.27	0.575	10	0	8	0	0	0	0	0
13	3810	4.13	0.201	0.00956	25	−1.25	0.36	10	0	0	0	0	0	0	0
14	2240	4.26	0.111	0.00804	29	−1.23	0.556	10	0	0	0	0	0	0	0
15	3200	6.31	0.524	0.0171	17	−1.19	0.354	10	0	0	0	0	0	0	0
16	3660	6.27	0.322	0.0000178	15	−1.18	0.277			10	0	0	0	0	0
17	2520	6.79	0.057	0.0108	24	−1.1	0.495	10	0	0	0	0	0	0	0
18	109	1.65	0.351	0.00193	26	−0.994	0.581	10	0	0	0	0	0	0	0
19	1900	6.66	0.408	0.0364	29	−0.97	0.32	10	0	0	0	0	0	0	0
20	2970	3.92	0.33	0.0232	21	−0.958	0.336	10	0	0	0	0	0	0	0

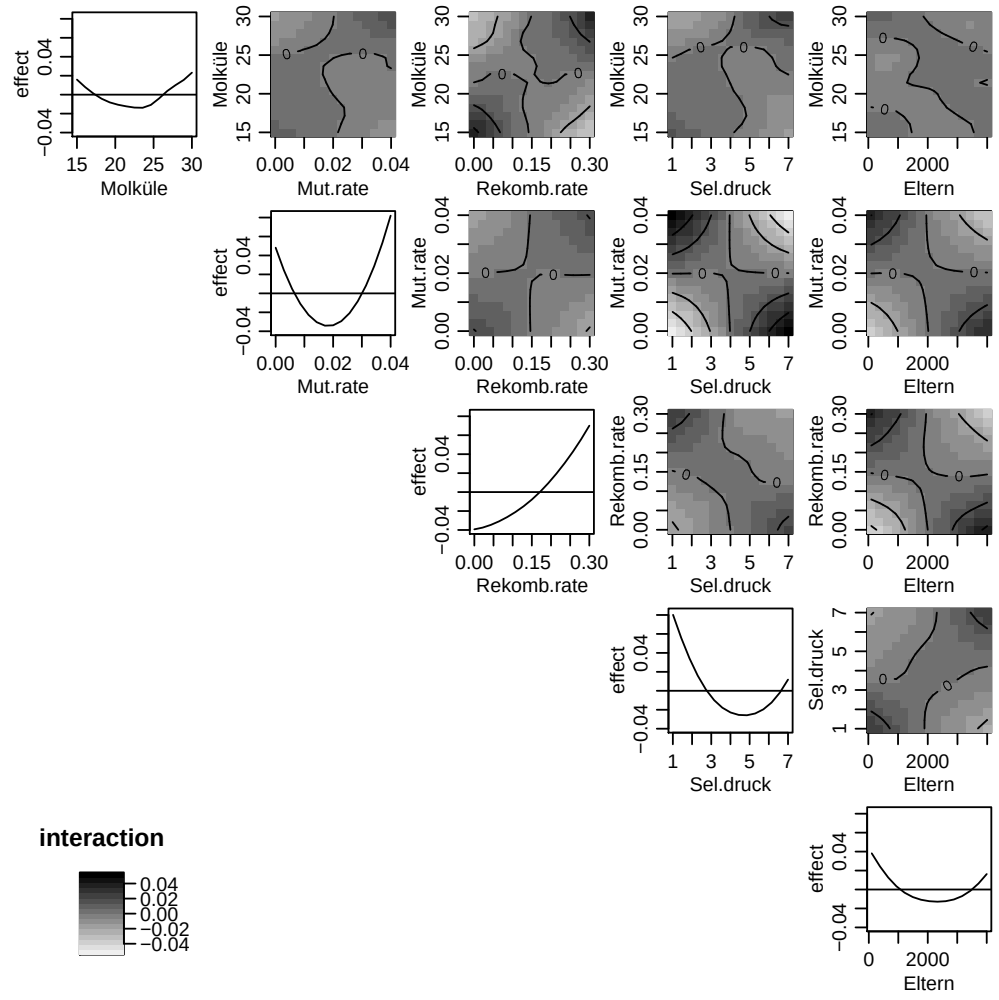


Abbildung A.1.: Effekt- und Interaktionsplots für die Parameter des AC-Systems unter Verwendung 1-Punkt-Rekombination auf dem Parityproblem.

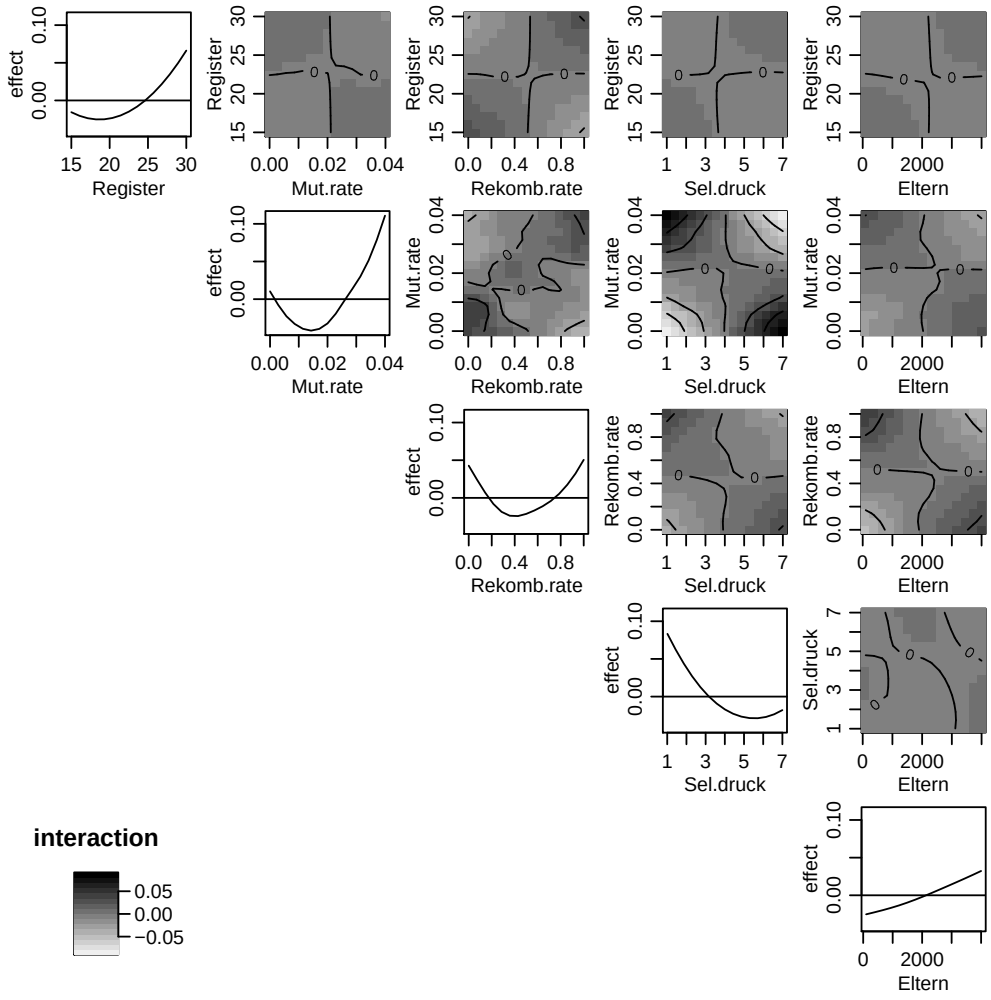


Abbildung A.2.: Effekt- und Interaktionsplots für die Parameter des LGP-Systems unter Verwendung 1-Punkt-Rekombination auf dem Parityproblem.

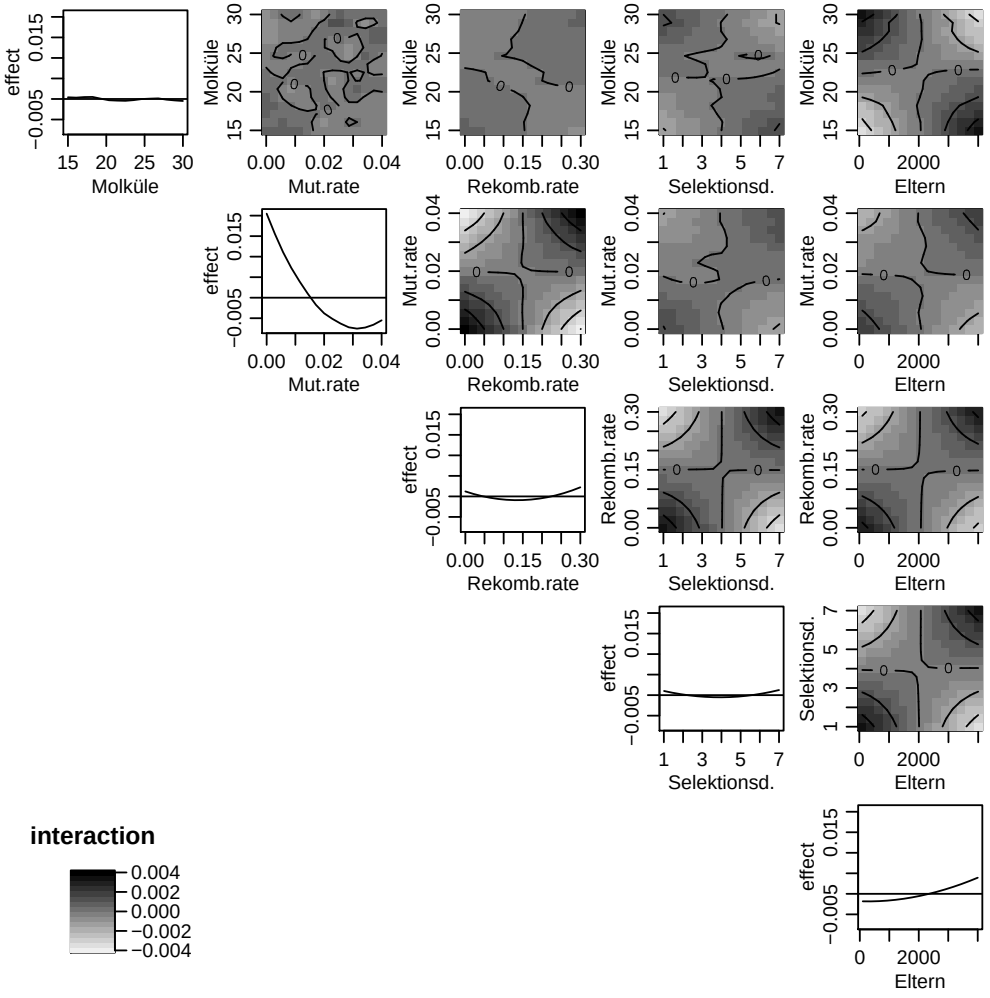


Abbildung A.3.: Effekt- und Interaktionsplots für die Parameter des AC-Systems unter Verwendung von 1-Punkt-Rekombination auf dem Thyroidproblem.

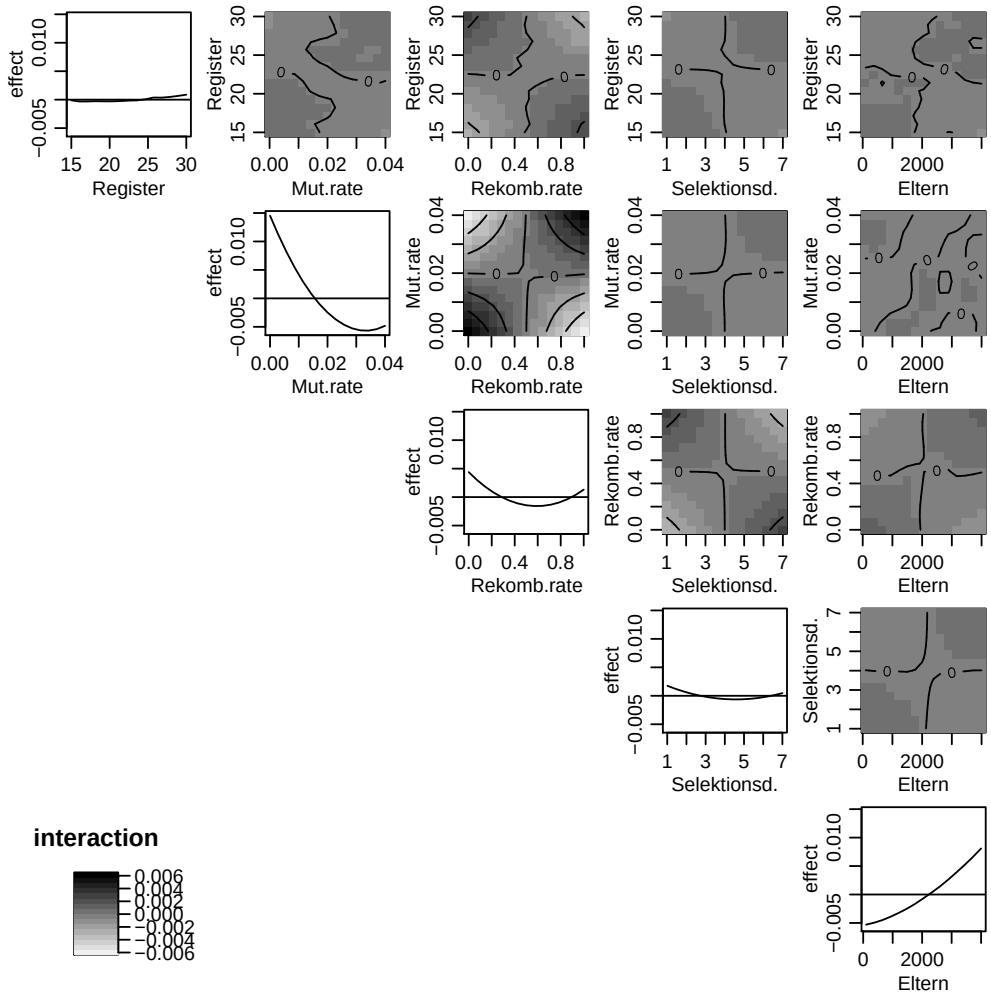


Abbildung A.4.: Effekt- und Interaktionsplots für die Parameter des LGP-Systems unter Verwendung von 1-Punkt-Rekombination auf dem Thyroidproblem.

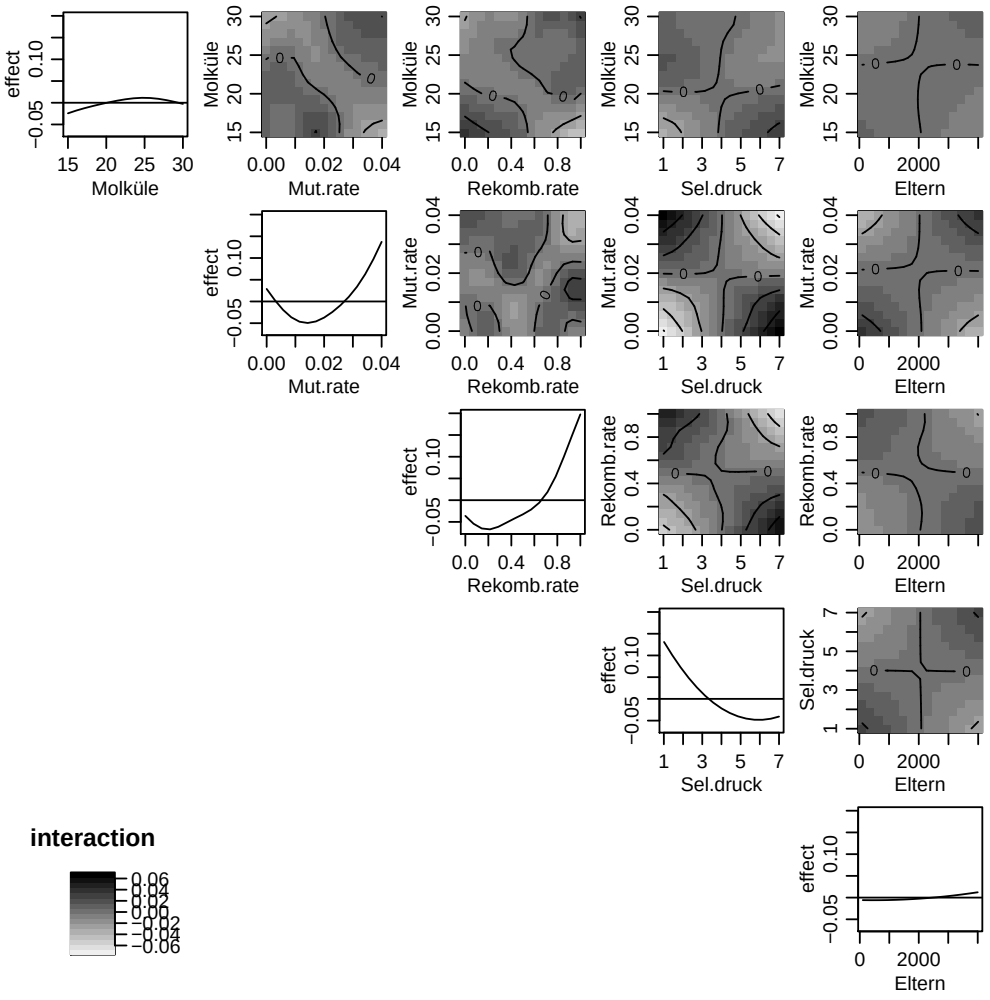


Abbildung A.5.: Effekt- und Interaktionsplots für die Parameter des AC-Systems unter Verwendung von homologer Rekombination auf dem Parityproblem.

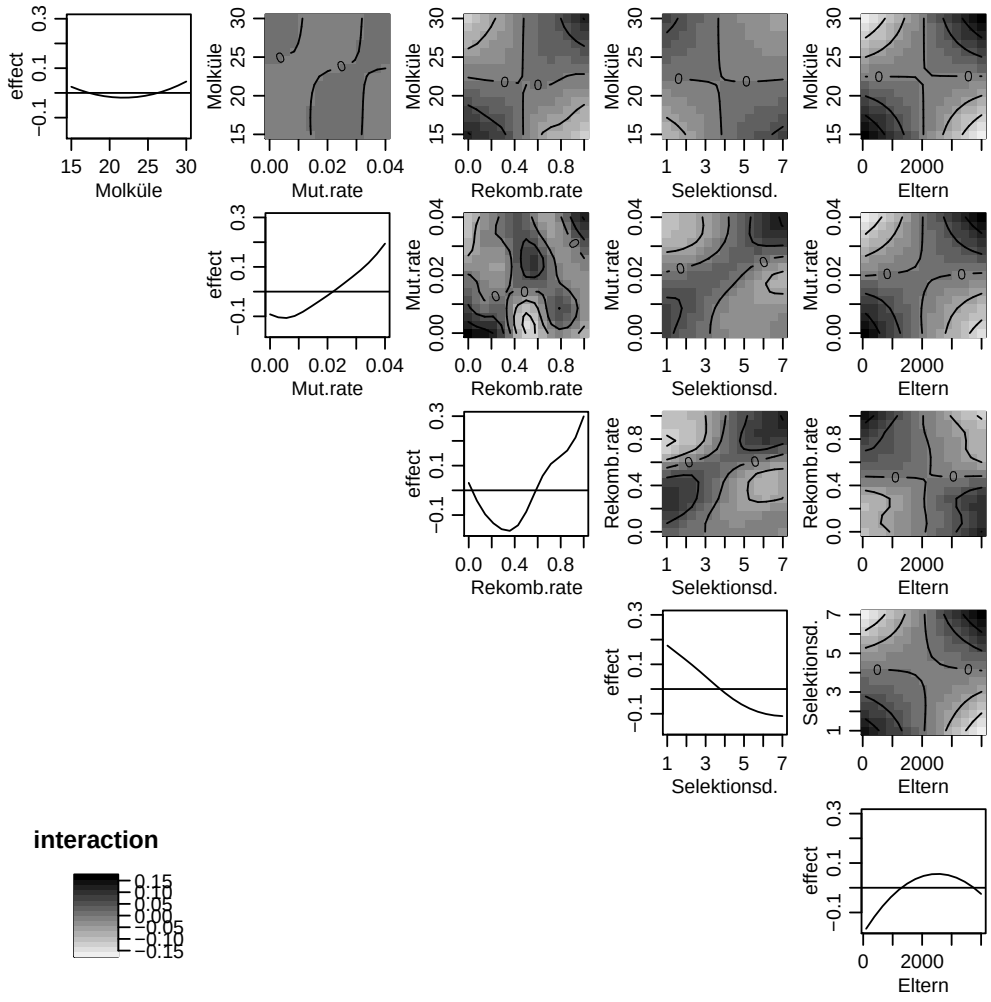


Abbildung A.6.: Effekt- und Interaktionsplots für die Parameter des AC-Systems unter Verwendung von homologer Rekombination auf dem Sinusproblem.

Literaturverzeichnis

[1] Abelson, Harold ; Allen, Don ; Coore, Daniel ; Hanson, Chris ; Homsy, George ; Knight, Thomas F. ; Nagpal, Radhika ; Rauch, Erik ; Sussman, Gerald J. ; Weiss, Ron: Amorphous Computing. In: *Communications of the ACM* 43 (2000), Nr. 5, S. 74–82

[2] Adleman, Leonard M.: Molecular Computation of Solutions to Combinatorial Problems. In: *Science* 266 (1994), November, S. 1021–1024

[3] Altenberg, Lee: The Evolution of Evolvability in Genetic Programming. In: Kinnear, Kenneth E. (Hrsg.): *Advances in Genetic Programming*. Cambridge : MIT, 1994 (Complex Adaptive Systems), S. 47–74

[4] Angeline, Peter J.: Subtree Crossover: Building Block Engine or Macromutation? In: Koza, John R. (Hrsg.) ; Deb, Kalyanmoy (Hrsg.) ; Dorigo, Marco (Hrsg.) ; Fogel, David B. (Hrsg.) ; Garzon, Max (Hrsg.) ; Iba, Hitoshi (Hrsg.) ; Riolo, Rick L. (Hrsg.): *Genetic Programming 1997: Proceedings of the Second Annual Conference*. San Francisco, CA, USA : Morgan Kaufmann, 1997, S. 9–17

[5] Angeline, Peter J.: Multiple Interacting Programs: A Representation for Evolving Complex Behaviors. In: *Cybernetics and Systems* 29 (1998), November, Nr. 8, S. 779–806

[6] Backus, John: Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. In: *Communications of the ACM* 21 (1978), August, Nr. 8, S. 613–641

[7] Banâtre, Jean-Pierre ; Fradet, Pascal ; Le Métayer, Daniel: Gamma and the Chemical Reaction Model: Fifteen Years After. In: Calude, Cristian S. (Hrsg.) ; Păun, Gheorghe (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.) ; Salomaa, Arto (Hrsg.): *Multiset Processing: Mathematical, Computer Science, and Molecular Computing Points of View* Bd. 2235. Springer, Berlin, 2001, S. 17–44

[8] Banzhaf, Wolfgang: Self-Replicating Sequences of Binary Numbers. In: *Computers and Mathematics, with Applications* 26 (1993), Nr. 7, S. 1–8

[9] Banzhaf, Wolfgang: Artificial Selection in a System of Self-Replicating Strings. In: *Proceedings of First IEEE Conference on Evolutionary Computation (ICEC'94) within First IEEE World Congress on Computational Intelligence (WCCI'94)* Bd. 2. Piscataway NJ : IEEE, Juni 1994, S. 651–655

[10] Banzhaf, Wolfgang: Genotype-Phenotype-Mapping and Neutral Variation – A case study in Genetic Programming. In: Davidor, Yuval (Hrsg.) ; Schwefel, Hans-Paul (Hrsg.) ; Männer, Reinhard (Hrsg.): *Parallel Problem Solving from Nature III*. Jerusalem : Springer, Berlin, 9-14 Oktober 1994 (LNCS 866), S. 322–332

[11] Banzhaf, Wolfgang: Self-Replicating Sequences of Binary Numbers – The Build-Up of Complexity. In: *Complex Systems* 8 (1994), Nr. 3, S. 205–215

[12] Banzhaf, Wolfgang: Artificial chemistries – Toward Constructive Dynamical Systems. In: *Solid State Phenomena* 97/98 (2004), S. 43–50

[13] Banzhaf, Wolfgang ; Dittrich, Peter ; Eller, Burkart: Self-Organization in a System of Binary Strings with Spatial Interactions. In: *Physica D* 125 (1999), Nr. 1–2, S. 85–104

[14] Banzhaf, Wolfgang ; Dittrich, Peter ; Rauhe, Hilmar: Emergent Computation by Catalytic Reactions. In: *Nanotechnology* (1996), Nr. 7, S. 307 – 314

[15] Banzhaf, Wolfgang ; Langdon, William B.: Some Considerations on the Reason for Bloat. In: *Genetic Programming and Evolvable Machines* 3 (2002), Nr. 1, S. 81–91

[16] Banzhaf, Wolfgang ; Lasarczyk, Christian W.G.: Genetic programming of an algorithmic chemistry. In: O'Reilly, U.-M. (Hrsg.) ; Yu, T. (Hrsg.) ; Riolo, R. (Hrsg.) ; Worzel, B. (Hrsg.): *Genetic Programming Theory and Practice II* Bd. 8. Boston MA : Kluwer/Springer, 2005, S. 175–190

[17] Banzhaf, Wolfgang ; Nordin, Peter ; Keller, Robert E. ; Francone, Frank D.: *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt, Januar 1998

[18] Bartz-Beielstein, Thomas: *Experimental Research in Evolutionary Computation - The New Experimentalism*. Berlin : Springer, Berlin, 2006 (Natural Computing Series). – ISBN 3-540-32026-1

[19] Bartz-Beielstein, Thomas ; Markon, Sandor: Tuning search algorithms for real-world applications: A regression tree based approach. In: Greenwood, G. W. (Hrsg.): *Proc. 2004 Congress on Evolutionary Computation (CEC'04), Portland* Bd. 1. Piscataway NJ : IEEE, 2004, S. 1111–1118. – ISBN 0-7803-8515-2

[20] Bechhofer, Robert E. ; Santner, Thomas J. ; Goldsman, David M.: *Design and Analysis of Experiments for Statistical Selection, Screening and Multiple Comparisons*. New York : Wiley, 1995

[21] Beyer, Hans-Georg ; Schwefel, Hans-Paul: Evolution strategies - A comprehensive introduction. In: *Natural Computing* 1 (2002), Nr. 1, S. 3–52

[22] Bleuler, Stefan ; Brack, Martin ; Thiele, Lothar ; Zitzler, Eckart: Multiobjective Genetic Programming: Reducing Bloat Using SPEA2. In: *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, IEEE Press, Piscataway, NJ, 27-30 Mai 2001, S. 536–543. – ISBN 0-7803-6658-1

[23] Blickle, Tobias ; Thiele, Lothar: Genetic Programming and Redundancy. In: Hopf, J. (Hrsg.): *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*. Saarbrücken : Max-Planc-k-Institut für Informatik (MPI-I-94-241), 1994, S. 33–38

[24] Bosman, Peter A. N. ; de Jong, Edwin D.: Learning Probabilistic Tree Grammars for Genetic Programming. In: Yao, Xin (Hrsg.) ; Burke, Edmund (Hrsg.) ; Lozano, Jose A. (Hrsg.) ; Smith, Jim (Hrsg.) ; Merelo-Guervós, Juan J. (Hrsg.) ; Bullinaria, John A. (Hrsg.) ; Rowe, Jonathan (Hrsg.) ; Kabán, Peter Tiňo A. (Hrsg.) ; Schwefel, Hans-Paul (Hrsg.): *Parallel Problem Solving from Nature - PPSN VIII* Bd. 3242. Birmingham, UK : Springer, Berlin, 18-22 September 2004, S. 192–201

[25] Brameier, Markus ; Banzhaf, Wolfgang: A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining. In: *IEEE Transactions on Evolutionary Computation* 5 (2001), Februar, Nr. 1, S. 17–26

[26] Bray, Dennis: Protein Molecules as Computational Elements in Living Cells. In: *Nature* 376 (1995), S. 307–312

[27] Busch, Jens: *RESAC: Eine resolutionsbasierte Künstliche Chemie und deren Anwendungen*, Universität Dortmund, Dissertation, 2004

[28] Busch, Jens ; Banzhaf, Wolfgang: Multi-Agent Systems Inspired by Artificial Chemistries: A Case Study in Automated Theorem Proving. In: *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS 2000)*, IEEE Computer Society, 2000, S. 371–372

[29] Chellapilla, Kumar: Evolving Computer Programs Without Subtree Crossover. In: *IEEE Trans. on Evolutionary Computation* 1 (1997), Nr. 3, S. 209–216

[30] Chen, Chun-Hung ; Lin, Jianwu ; Yücesan, Enver ; Chick, Stephen E.: Simulation Budget Allocation for Further Enhancing the Efficiency of Ordinal Optimization. In: *Discrete Event Dynamic Systems* 10 (2000), Jul, Nr. 3, S. 251–270

[31] Chen, E. J. ; Chen, Chun-Hung ; Kelton, W. D.: *Optimal Computing Budget Allocation of Indifference-Zone-Selection Procedures*. – URL

`http://www.cba.uc.edu/faculty/keltonwd/`. – Zugriffsdatum: 8.2.2006. – Working Paper

[32] Conrad, Michael: On design principles for a molecular computer. In: *Communications of the ACM* 28 (1985), Nr. 5, S. 464–480

[33] Conrad, Michael: The Price of Programmability. In: Herkin, Rolf (Hrsg.): *The Universal Turing Machine: A Half-Century Survey*. Oxford : Oxford University Press, 1988, S. 285–307

[34] Conrad, Michael: The geometry of evolution. In: *BioSystems* 24 (1990), S. 61–81

[35] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L.: *Introduction to Algorithms*. Cambridge, MA : MIT Press, 1990

[36] Cramer, Nichael L.: A representation for the Adaptive Generation of Simple Sequential Programs. In: Grefenstette, John J. (Hrsg.): *Proceedings of an International Conference on Genetic Algorithms and the Applications*. Hillsdale, New Jersey : Lawrence Erlbaum Associates, 24-26 Juli 1985, S. 183–187

[37] de Jong, Edwin D. ; Watson, Richard A. ; Pollack, Jordan B.: Reducing Bloat and Promoting Diversity using Multi-Objective Methods. In: Spector, Lee (Hrsg.) ; Goodman, Erik D. (Hrsg.) ; Wu, Annie (Hrsg.) ; Langdon, W. B. (Hrsg.) ; Voigt, Hans-Michael (Hrsg.) ; Gen, Mitsuo (Hrsg.) ; Sen, Sandip (Hrsg.) ; Dorigo, Marco (Hrsg.) ; Pezeshk, Shahram (Hrsg.) ; Garzon, Max H. (Hrsg.) ; Burke, Edmund (Hrsg.): *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. San Francisco, California, USA : Morgan Kaufmann, 7-11 Juli 2001, S. 11–18. – ISBN 1-55860-774-9

[38] Dewdney, Alexander K.: Computer Recreations: In the game called Core War hostile programs engage in a battle of bits. In: *Scientific American* 250 (1984), Nr. 5, S. 14–22

[39] Dittrich, Peter ; Ziegler, Jens ; Banzhaf, Wolfgang: Artificial Chemistries – A Review. In: *Artificial Life* 7 (2001), Nr. 3, S. 225–275

[40] Eiben, Agoston E. ; Smith, James E.: *Introduction to Evolutionary Computing*. Berlin : Springer, 2003 (Natural Computing Series)

[41] Emmerich, M. ; Giannakoglou, K. ; Naujoks, B.: Single- and Multi-objective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels. In: *IEEE Transactions on Evolutionary Computation* (2006). – (akzeptiert zur Veröffentlichung)

[42] Emmerich, Michael ; Giotis, Alexios ; Özdemir, Mutlu ; Bäck, Thomas ; Giannakoglou, Kyriakos: Metamodel-assisted evolution strategies. In: Guervós, J. J. M. (Hrsg.) ; Adamidis, P. (Hrsg.) ; Beyer, H.-G. (Hrsg.) ;

Fernández-Villacañas, J. L. (Hrsg.) ; Schwefel, H.-P. (Hrsg.): *Parallel Problem Solving from Nature – PPSN VII, Proceedings Seventh International Conference, Granada*. Berlin : Springer, 2002, S. 361–370. – ISBN 3-540-44139-5

[43] Feldkamp, Udo ; Rauhe, Hilmar ; Banzhaf, Wolfgang: Software Tools for DNA Sequence Design. In: *Genetic Programming and Evolvable Machines* 4 (2003), Juni, Nr. 2, S. 153–171

[44] Fontana, Walter: Algorithmic Chemistry. In: Langton, Christopher G. (Hrsg.) ; Taylor, Charles (Hrsg.) ; Farmer, J. D. (Hrsg.) ; Rasmussen, Steen (Hrsg.): *Artificial Life* Bd. II. Redwood City, CA : Addison-Wesley, 1992, S. 159–209

[45] Fontana, Walter ; Buss, Leo W.: The arrival of the fittest: toward a theory of biological organization. In: *Bulletin of Mathematical Biology* 56 (1994), S. 1–64

[46] Francone, Frank D. ; Conrads, Markus ; Banzhaf, Wolfgang ; Nordin, Peter: Homologous Crossover in Genetic Programming. In: Banzhaf, Wolfgang (Hrsg.) ; Daida, Jason (Hrsg.) ; Eiben, Agoston E. (Hrsg.) ; Garzon, Max H. (Hrsg.) ; Honavar, Vasant (Hrsg.) ; Jakiela, Mark (Hrsg.) ; Smith, Robert E. (Hrsg.): *Proceedings of the Genetic and Evolutionary Computation Conference* Bd. 2. San Francisco, CA 94104, USA : Morgan Kaufmann, 1999, S. 1021–1026

[47] Garzon, Max H. ; Deaton, Russell J.: Biomolecular Computing and Programming. In: *IEEE Transactions on Evolutionary Computation* 3 (1999), September, Nr. 3, S. 236–250

[48] Gathercole, Chris: *An Investigation of Supervised Learning in Genetic Programming*, University of Edinburgh, Dissertation, 1998

[49] Goldsman, David ; Nelson, Barry L.: Statistical selection of the best system. In: *Proceedings of the 2001 Winter Simulation Conference*, 2001, S. 139–146

[50] Gruau, Frédéric ; Lhuillier, Yves ; Reitz, Philippe ; Temam, Olivier: Blob Computing. In: *Computing Frontiers* ACM (Veranst.), SIGMicro, 2004, S. 125–139

[51] Harik, Georges: Linkage Learning via Probabilistic Modeling in the ECGA / University of Illinois at Urbana-Champaign. 1999 (99010). – Forschungsbericht. IlliGAL Report

[52] Holm, S.: A simple sequentially rejective multiple test procedure. In: *Scandinavian Journal of Statistics* (1979), Nr. 6, S. 65–70

[53] Jansen, Thomas ; Wegener, Ingo: On the Analysis of Evolutionary Algorithms - A Proof That Crossover Really Can Help. In: Nesetril, Jaroslav (Hrsg.): *Algorithms - ESA '99, 7th Annual European Symposium, Prague, Czech*

Republic, July 16-18, 1999, Proceedings Bd. 1643, Springer, 1999, S. 184–193. – ISBN 3-540-66251-0

[54] Jones, D.R. ; Schonlau, M. ; Welch, W.J.: Efficient Global Optimization of Expensive Black-Box Functions. In: *Journal of Global Optimization* 13 (1998), S. 455–492

[55] Kammeyer, Thomas ; Belew, R. K.: Stochastic Context-Free Grammar Induction with a Genetic Algorithm Using Local Search. In: Belew, Richard K. (Hrsg.) ; Vose, Michael (Hrsg.): *Foundations of Genetic Algorithms IV*. University of San Diego, CA, USA : Morgan Kaufmann, 3–5 August 1996, S. 409–436

[56] Kantschik, Wolfgang ; Banzhaf, Wolfgang: Linear-Tree GP and its comparison with other GP structures. In: Miller, Julian F. (Hrsg.) ; Tomassini, Marco (Hrsg.) ; Lanzi, Pier L. (Hrsg.) ; Ryan, Conor (Hrsg.) ; Tettamanzi, Andrea G. B. (Hrsg.) ; Langdon, William B. (Hrsg.): *Genetic Programming, Proceedings of EuroGP'2001* Bd. 2038. Lake Como, Italy : Springer, Berlin, 18-20 April 2001, S. 302–312

[57] Kantschik, Wolfgang ; Banzhaf, Wolfgang: Linear-Graph GP – A new GP Structure. In: Foster, James A. (Hrsg.) ; Lutton, Evelyne (Hrsg.) ; Miller, Julian (Hrsg.) ; Ryan, Conor (Hrsg.) ; Tettamanzi, Andrea G. B. (Hrsg.): *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002* Bd. 2278. Kinsale, Ireland : Springer, Berlin, 3-5 April 2002, S. 83–92

[58] Keller, Robert E. ; Banzhaf, Wolfgang: Genetic Programming using Genotype-Phenotype Mapping from Linear Genomes into Linear Phenotypes. In: Koza, John R. (Hrsg.) ; Goldberg, David E. (Hrsg.) ; Fogel, David B. (Hrsg.) ; Riolo, Rick L. (Hrsg.): *Genetic Programming 1996: Proceedings of the First Annual Conference*. Stanford University, CA, USA : MIT, 28–31 Juli 1996, S. 116–122

[59] Keller, Robert E. ; Banzhaf, Wolfgang: The Evolution of Genetic Code in Genetic Programming. In: Banzhaf, Wolfgang (Hrsg.) ; Daida, Jason (Hrsg.) ; Eiben, Agoston E. (Hrsg.) ; Garzon, Max H. (Hrsg.) ; Honavar, Vasant (Hrsg.) ; Jakiela, Mark (Hrsg.) ; Smith, Robert E. (Hrsg.): *Proceedings of the Genetic and Evolutionary Computation Conference* Bd. 2. Orlando, Florida, USA : Morgan Kaufmann, 13-17 Juli 1999, S. 1077–1082

[60] Kim, Seong-Hee ; Nelson, Barry L.: Selecting the Best System: Theory and Methods. In: *Proceedings of the 2003 Winter Simulation Conference*, ACM, 2003, S. 101–112

[61] Kirschner, Marc ; Gerhart, John: Evolvability. In: *Proc. Natl. Acad. Sci.* 95 (1998), S. 8420–8427

[62] Koza, John R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA : MIT Press, 1992

[63] Koza, John R.: *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge Massachusetts : MIT Press, Mai 1994

[64] Langdon, W. B. ; Poli, R.: Why "Building Blocks" Don't Work on Parity Problems / University of Birmingham, School of Computer Science. 13 Juli 1998 (CSRP-98-17). – Forschungsbericht

[65] Langdon, William B.: Scaling of Program Tree Fitness Spaces. In: *Evolutionary Computation* 7 (1999), Winter, Nr. 4, S. 399–428. – ISSN 1063-6560

[66] Langdon, William B.: Size Fair and Homologous Tree Genetic Programming Crossovers. In: Banzhaf, Wolfgang (Hrsg.) ; Daida, Jason (Hrsg.) ; Eiben, Agoston E. (Hrsg.) ; Garzon, Max H. (Hrsg.) ; Honavar, Vasant (Hrsg.) ; Jakiela, Mark (Hrsg.) ; Smith, Robert E. (Hrsg.): *Proceedings of the Genetic and Evolutionary Computation Conference* Bd. 2. San Francisco, CA 94104, USA : Morgan Kaufmann, 1999, S. 1092–1097

[67] Langdon, William B.: Size Fair and Homologous Tree Genetic Programming Crossovers. In: *Genetic Programming and Evolvable Machines* 1 (2000), April, Nr. 1/2, S. 95–119

[68] Langdon, William B. ; Poli, Riccardo: Fitness Causes Bloat / University of Birmingham, School of Computer Science. 24 Februar 1997 (CSRP-97-09). – Forschungsbericht

[69] Lones, Michael ; Tyrrell, Andy: Crossover and Bloat in the Functionality Model of Enzyme Genetic Programming. In: Fogel, David B. (Hrsg.) ; El-Sharkawi, Mohamed A. (Hrsg.) ; Yao, Xin (Hrsg.) ; Greenwood, Garry (Hrsg.) ; Iba, Hitoshi (Hrsg.) ; Marrow, Paul (Hrsg.) ; Shackleton, Mark (Hrsg.): *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, IEEE, 2002, S. 986–991

[70] Lones, Michael A. ; Tyrrell, Andy M.: Biomimetic Representation with Enzyme Genetic Programming. In: *Genetic Programming and Evolvable Machines* 3 (2002), Juni, Nr. 2, S. 193–217

[71] Lones, Michael A.: *Enzyme Genetic Programming – Modelling Biological Evolvability in Genetic Programming*. Heslington, York, Department of Electronics, University of York, Dissertation, 2004

[72] Lophaven, S.N. ; Nielsen, H.B. ; Søndergaard, J.: DACE – A Matlab Kriging Toolbox / Informatics and Mathematical Modelling, Technical University of Denmark, Copenhagen, Denmark. 2002 (IMM-REP-2002-12). – Forschungsbericht

[73] Luke, Sean ; Spector, Lee: A Revised Comparison of Crossover and Mutation in Genetic Programming. In: Koza, John R. (Hrsg.) ; Banzhaf, Wolfgang (Hrsg.) ; Chellapilla, Kumar (Hrsg.) ; Deb, Kalyanmoy (Hrsg.) ; Dorigo, Marco (Hrsg.) ; Fogel, David B. (Hrsg.) ; Garzon, Max H. (Hrsg.) ; Goldberg, David E. (Hrsg.) ; Iba, Hitoshi (Hrsg.) ; Riolo, Rick (Hrsg.): *Genetic Programming 1998: Proceedings of the Third Annual Conference*. San Francisco, CA, USA : Morgan Kaufmann, 1998, S. 208–213

[74] Mayr, Ernst: *Das ist Evolution*. 3. München : C. Bertelsmann, 2003

[75] McKay, M. D. ; Beckman, R. J. ; Conover, W. J.: A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. In: *Technometrics* 21 (1979), Nr. 2, S. 239–245

[76] Miller, Julian F.: An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach. In: Banzhaf, Wolfgang (Hrsg.) ; Daida, Jason (Hrsg.) ; Eiben, Agoston E. (Hrsg.) ; Garzon, Max H. (Hrsg.) ; Honavar, Vasant (Hrsg.) ; Jakiela, Mark (Hrsg.) ; Smith, Robert E. (Hrsg.): *Proceedings of the Genetic and Evolutionary Computation Conference* Bd. 2. Orlando, Florida, USA : Morgan Kaufmann, 13-17 Juli 1999, S. 1135–1142

[77] Miller, Julian F. ; Job, Dominic ; Vassilev, Vesselin K.: Principles in the Evolutionary Design of Digital Circuits-Part I. In: *Genetic Programming and Evolvable Machines* 1 (2000), April, Nr. 1/2, S. 7–35. – ISSN 1389-2576

[78] Miller, Julian F. ; Thomson, Peter: Cartesian Genetic Programming. In: Poli, Riccardo (Hrsg.) ; Banzhaf, Wolfgang (Hrsg.) ; Langdon, William B. (Hrsg.) ; Miller, Julian F. (Hrsg.) ; Nordin, Peter (Hrsg.) ; Fogarty, Terence C. (Hrsg.): *Genetic Programming, Proceedings of EuroGP'2000* Bd. 1802. Edinburgh : Springer, Berlin, 15-16 April 2000, S. 121–132

[79] Montgomery, Douglas C. ; Runger, George C.: *Applied statistics and probability for engineers*. Wiley, 2003

[80] Neumann, John von: Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. In: Shannon, Claude E. (Hrsg.) ; McCarthy, John (Hrsg.): *Automata Studies*. Princeton, NJ : Princeton University Press, 1956, S. 43–98

[81] Newman, D.J. ; Hettich, S. ; Blake, C.L. ; Merz, C.J.: *UCI Repository of machine learning databases*. 1998. – URL `http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html`

[82] Nordin, Peter: A Compiling Genetic Programming System that Directly Manipulates the Machine Code. In: Kinnear, Kenneth E. (Hrsg.): *Advances in Genetic Programming*. Cambridge : MIT Press, 1994 (Complex Adaptive Systems), S. 311–332

[83] Nordin, Peter ; Banzhaf, Wolfgang: Complexity Compression and Evolution. In: Eshelman, Larry J. (Hrsg.): *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*. San Francisco, CA : Morgan Kaufmann, 1995, S. 310–317

[84] Nordin, Peter ; Banzhaf, Wolfgang: Evolving Turing-Complete Programs for a Register Machine with Self-modifying Code. In: Eshelman, Larry J. (Hrsg.): *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*. Pittsburgh, PA, USA : Morgan Kaufmann, 15-19 Juli 1995, S. 318–325

[85] Nordin, Peter ; Francone, Frank ; Banzhaf, Wolfgang: Explicitly Defined Introns and Destructive Crossover in Genetic Programming. In: Angeline, Peter J. (Hrsg.) ; Kinnear, Jr., K. E. (Hrsg.): *Advances in Genetic Programming 2*. Cambridge, MA : MIT, 1996, Kap. 6, S. 111–134

[86] O'Neill, Michael ; Ryan, Conor: Grammatical Evolution. In: *IEEE Transactions on Evolutionary Computation* 5 (2001), Nr. 4, S. 349–358

[87] O'Neill, Michael ; Ryan, Conor: *Genetic programming*. Bd. 4: *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*. Kluwer Academic Publishers, 2003

[88] O'Reilly, Una-May ; Oppacher, Franz: The Troubling Aspects of a Building Block Hypothesis for Genetic Programming. In: Whitley, L. D. (Hrsg.) ; Vose, Michael D. (Hrsg.): *Foundations of Genetic Algorithms 3*. San Francisco, CA : Morgan Kaufmann, 1995, S. 73–88

[89] Palem, Krishna V.: Energy Aware Computing through Probabilistic Switching: A Study of Limits. In: *IEEE Transactions on Computers* 54 (2005), Nr. 9, S. 1123–1137

[90] Poli, Riccardo: Evolution of Graph-like Programs with Parallel Distributed Genetic Programming. In: Back, Thomas (Hrsg.): *Genetic Algorithms: Proceedings of the Seventh International Conference*. Michigan State University, East Lansing, MI, USA : Morgan Kaufmann, 19-23 Juli 1997, S. 346–353

[91] Port, Otis: *Chips That Thrive on Uncertainty*. BuisnessWeek online. Februar 2005

[92] Păun, Gheorghe: Computing with Membranes. In: *Journal of Computer and System Sciences* 61 (2000), Nr. 1, S. 108–143

[93] Păun, Gheorghe: Membrane Systems: A Quick Introduction. In: *International Workshop on Unconventional Programming Paradigms (UPP), LNCS*, 2004, S. 188–195

[94] Păun, Gheorghe: Introduction to Membrane Computing. In: Ciobanu, Gabriel (Hrsg.) ; Pérez-Jiménez, Mario J. (Hrsg.) ; Păun, Gheorghe (Hrsg.): *Applications of Membrane Computing*. Springer, 2006 (Natural Computing Series), Kap. 1, S. 1–42

[95] Păun, Gheorghe ; Rozenberg, Grzegorz: A guide to membrane computing. In: *Theor. Comput. Sci.* 287 (2002), Nr. 1, S. 73–100

[96] Rasmussen, Steen ; Knudsen, Carsten ; Feldberg, Pasmus ; Hindsholm, Morten: The coreworld: emergence and evolution of cooperative structures in a computational chemistry. In: *Physica D* 42 (1990), Nr. 1-3, S. 111–134

[97] Ratle, Alain ; Sebag, Michele: A novel approach to Machine Discovery: Genetic Programming and Stochastic Grammars. In: Matwin, S. (Hrsg.) ; Sammut, C. (Hrsg.): *Proceedings of Twelfth International Conference on Inductive Logic Programming* Bd. 2583. Sydney, Australia : Springer, Berlin, Juli 9-11, 2002 2003, S. 207–222

[98] Ray, Thomas S.: An Evolutionary Approach to Synthetic Biology: Zen and the Art of Creating Life. In: *Artificial Life* 1/2 (1994), Nr. 1, S. 195–226.

[99] Ruben, Adam J. ; Landweber, Laura F.: The past, present and future of molecular computing. In: *Nature Reviews Molecular Cell Biology* 1 (2000), S. 69–72

[100] Sacks, J. ; Welch, W. J. ; Mitchell, T. J. ; Wynn, H. P.: Design and analysis of computer experiments. In: *Statistical Science* 4 (1989), Nr. 4, S. 409–435

[101] Salustowicz, R. P. ; Schmidhuber, J.: Probabilistic Incremental Program Evolution. In: *Evolutionary Computation* 5 (1997), Nr. 2, S. 123–141

[102] Sastry, Kumara ; Goldberg, David E.: Probabilistic Model Building and Competent Genetic Programming. In: Riolo, Rick L. (Hrsg.) ; Worzel, Bill (Hrsg.): *Genetic Programming Theory and Practise*. Kluwer, 2003, Kap. 13, S. 205–220

[103] Schiffmann, W. ; Joost, M. ; Werner, R.: Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons. In: Albrecht, Rudolf F. (Hrsg.) ; Steele, Nigel C. (Hrsg.) ; Reeves, Colin R. (Hrsg.): *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, 1993, S. 675–682

[104] Schonlau, M. ; Welch, W.J. ; Jones, R.D.: Global versus local search in constrained optimization of computer models. In: Flournoy, N. (Hrsg.) ; Rosenberger, W.F. (Hrsg.) ; Wong, W.K. (Hrsg.): *New developments and applications in experimental design* Bd. 34. Beachwood OH : Institute of Mathematical Statistics, 1998, S. 11–25

[105] Shor, P. W.: Fault-tolerant quantum computation. In: IEEE (Hrsg.): *37th Annual Symposium on Foundations of Computer Science*. Burlington, Vermont : IEEE Computer Society Press, 1996, S. 56–65

[106] Skusa, Andre ; Banzhaf, Wolfgang ; Busch, Jens ; Dittrich, Peter ; Ziegler, Jens: Künstliche Chemie. In: *Künstliche Intelligenz* (2000), S. 12–19

[107] Soule, Terence: *Code Growth in Genetic Programming*. Moscow, Idaho, USA, University of Idaho, Dissertation, 1998

[108] Soule, Terence ; Foster, James A.: Code Size and Depth Flows in Genetic Programming. In: Koza, John R. (Hrsg.) ; Deb, Kalyanmoy (Hrsg.) ; Dorigo, Marco (Hrsg.) ; Fogel, David B. (Hrsg.) ; Garzon, Max (Hrsg.) ; Iba, Hitoshi (Hrsg.) ; Riolo, Rick L. (Hrsg.): *Genetic Programming 1997: Proceedings of the Second Annual Conference*. San Francisco, CA : Morgan Kaufmann, 1997, S. 313–320

[109] Soule, Terence ; Foster, James A.: Removal Bias: a New Cause of Code Growth in Tree Based Evolutionary Programming. In: *1998 IEEE International Conference on Evolutionary Computation*, IEEE, 1998, S. 781–186

[110] Soule, Terence ; Foster, James A.: Effects of Code Growth and Parsimony Presure on Populations in Genetic Programming. In: *Evolutionary Computation* 6 (1999), Nr. 4, S. 293–309

[111] *Computing Media and Languages for Space-Oriented Computation*. `http://www.dagstuhl.de/de/programm/kalender/semhp/?semid=27127`. September 2006. – Dagstuhl-Seminar 6361

[112] Swanson, Steven ; Michelson, Ken ; Schwerin, Andrew ; Oskin, Mark: WaveScalar. In: *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, ACM/IEEE, 2003, S. 291–302

[113] Teller, Astro ; Veloso, Manuela: PADO: A New Learning Architecture for Object Recognition. In: Ikeuchi, Katsushi (Hrsg.) ; Veloso, Manuela (Hrsg.): *Symbolic Visual Learning*. Oxford University Press, 1996, S. 81–116

[114] Thearling, Kurt: Evolution, entropy, and parallel computation. In: *Workshop on Physics and Computation, 1994. PhysComp '94*, 1994, S. 246–254

[115] Trosset, M.W. ; Padula, A.D.: Designing and Analyzing Computational Experiments for Global Optimization / Department of Computational and Applied Mathematics, Rice University, Houston TX. 2000 (00-25). – Forschungsbericht

[116] Weiss, Sholom M. ; Kapouleas, Ioannis: An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods. In: Sridharan, N. S. (Hrsg.): *Proceedings of the 11th International*

Joint Conference on Artificial Intelligence. Detroit, MI, USA : Morgan Kaufmann, August 1989, S. 781–787

[117] Welch, W. J. ; Buck, R. J. ; Sacks, J. ; Wynn, H. P. ; Mitchell, T. J. ; Morris, M. D.: Screening, predicting, and computer experiments. In: *Technometrics* 34 (1992), S. 15–25

[118] Zauner, Klaus-Peter: *Biomolekulare Rechner*. Telepolis – Heise Zeitschriften Verlag. August 1998. – URL `http://www.telepolis.de/r4/artikel/2/2423/1.html`

[119] Zhang, Byoung-Tak ; Mühlenbein, Heinz: Balancing Accuracy and Parsimony in Genetic Programming. In: *Evolutionary Computation* 3 (1995), Nr. 1, S. 17–38

[120] Ziegler, Jens ; Dittrich, Peter ; Banzhaf, Wolfgang: Towards a metabolic robot control system. In: Holcombe, Mike (Hrsg.) ; Paton, Ray (Hrsg.): *Proceedings of the International Workshop on Information Processing in Cells and Tissues (IPCAT'97)*. New York : Plenum, 1998, S. 305–317

Index