Diplomarbeit

Verbesserung eines hierarchischen evolutionären Algorithmus mit Anwendungen in der Optimierung und dem maschinellen Lernen

Improving a Hierarchical Evolutionary Algorithm with Applications in Optimization and Machine Learning

Michael Neumann

31.03.2016

Karlsruher Institut für Technologie
Fakultät für Informatik
Institut für Theoretische Informatik

Bearbeitungszeitraum
09. 09. 2015 – 31. 03. 2016

Gutachter
Juniorprof. Dr.rer.nat. H. Meyerhenke
Prof. Dr. Marc Weber

Betreuer
Dr. Fridtjof Feldbusch

Erklärung

Hiermit erkläre ich, dass ich diese Diplomarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichen Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Diplomarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Karlsruhe, den 31.03.2016

Michael Neumann

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Geht man von der Richtigkeit der Darwinschen Evolutionstheorie aus, so ist es die *Evolution*, die seit Millionen von Jahren erfolgreich dafür sorgt, dass sich eine Vielzahl unterschiedlichster Lebensformen entwickelten, die sich fortwährend den veränderten Umweltbedingungen und Widrigkeiten unseres Planetens auf scheinbar optimale Art und Weise angepasst haben. Es verwundert daher nicht, dass sich der Mensch ähnliche Konzepte, zum Beispiel in Form der *Evolutionären* oder *Genetischen Algorithmen* in der Informatik zunutze macht, um Lösungen für komplizierte Probleme heuristisch zu finden, für die sich entweder aufgrund der Komplexität oder von Nichtlinearitäten in der Problemstellung keine exakten Algorithmen entwerfen lassen[1], oder sich diese aufgrund hoher Laufzeiten[2] mit den heutigen, uns zur Verfügung stehenden Rechnermodellen, nicht effizient[3] lösen lassen.

So werden *Evolutionäre Algorithmen* erfolgreich in einem breiten Spektrum unterschiedlichster Anwendungsgebiete zur Lösung von Optimierungsproblemen eingesetzt, beispielsweise im automatisierten Entwurf elektromechanischer Systeme, etwa zur Entwicklung von Antennen für Miniaturweltraumfahrzeuge [Hor+06], in der Finanzmathematik zur Vorhersage von Börsenkursen [Che02], in der Molekularbiologie zum Auffinden der Signalsequenzen von Proteinen [Pon+06], oder im Bereich der Medizin zur Klassifikation von Brustkrebs [Dol+06], um nur einige zu nennen.

Ein weiterer Bereich, in dem sich der Mensch Konzepte aus der Natur zu Nutze macht, ist das *Maschinelle Lernen* bzw. die Künstliche Intelligenz. Hier hat sich der Mensch die Funktionsweise von biologischen neuronalen Netzen abgeschaut und daraus mathematische Modelle wie die *künstlichen neuronalen Netze*, basierend auf dem Perzeptron, oder zeitbehaftete Neuronenmodelle wie die *gepulsten neuronalen Netze*, die ihrem biologischen Vorbild sehr viel näher kommen, entwickelt. Während sich die Forschung in den letzten Jahrzehnten, aufgrund der besseren Beherrschbarkeit und der Existenz von Lernverfahren, größtenteils auf den Einsatz künstlicher neuronaler Netze konzentrierte, so erleben die gepulsten neuronalen Netze, abgesehen vom ständigen Interesse innheralb des Bereiches der Neurowissenschaften, jüngst neue Aufmerksamkeit, unter anderem auch aufgrund neuer Erkenntnisse wie z.B.

[1] bzw. nur unter sehr hohem menschlichen Aufwand
[2] NP-vollständige Probleme
[3] auch nicht in absehbarer Zeit

neue prädiktive Lernverfahren [Güt16]. Eingesetzt werden künstliche bzw. gepulste neuronale Netze meisst zur Klassifikation von hochdimensionalen, unscharfen und unvollständigen Daten, bei denen zudem die zur Klassifikation benötigten Parameter nicht oder nur unzureichend bekannt sind. Die Bild- oder Spracherkennung sind zwei Beispiele hierfür.

Die *Verbindung* zwischen evolutionären Algorithmen und künstlichen bzw. gepulsten neuronalen Netzen ergibt sich derweils aus der Schwierigkeit heraus, die Netze der Problemstellung entsprechend geeignet zu konfigurieren. Dies betrifft insbesondere die Wahl der Topologie, aber auch die Einstellung der Gewichtung solcher Netze. Während für die Einstellung der Gewichtung bei künstlichen neuronalen Netzen Verfahren zur Minimierung des Klassifikationsfehlers beispielsweise in Form von Backpropagation existieren, so ist dies aufgrund des nichtlinearen Verhaltens bei gepulsten Neuronenmodellen nur schwer möglich. Zwar existieren Ansätze zum Einlernen der Gewichtung auch für gepulste neuronale Netze[4], allerdings sind diese meisst auf einzelne Aspekte wie die synaptische Effizienz[5] beschränkt und lassen andere Aspekte, wie die Verzögerungszeit von Synapsen, ausser Acht. Da zudem ein Teil der Berechnungsmächtigkeit von gepulsten neuronalen Netzen indirekt durch deren synaptische Verzögerungszeiten gegeben ist, kann hier nicht auf reguläre Topologien, wie etwa Vollverdrahtung zwischen zwei Schichten, zurückgegriffen werden[6], ohne einen Verlust an Performanz und in der Berechnungsmächtigkeit in Kauf zu nehmen. Aus diesem Grunde wurde bereits in den 90er Jahren des letzten Jahrtausends der Versuch unternommen, die Topologie von gepulsten (und künstlichen) neuronalen Netzen, sowie Teile der Gewichtung, wie beispielsweise die Verzögerungszeit, durch den Einsatz von evolutionären Algorithmen zu entwickeln und zu optimieren.

Im Folgenden betrachten wir aussschliesslich die Optimierung von *gepulsten neuronalen Netzen*. Nicht nur ist die Berechnungsmächtigkeit dieser Netze bei Verwendung von Zeitkodierung erwiesenermaßen [Maa99] (streng) grösser als die von sigmoiden künstlichen neuronalen Netze derselben Grösse. Auch lassen sich bestimmte Phänomene des menschlichen Cortex nur durch Zeitkodierung erklären und nicht durch Ratenkodierung wie sie bei künstlichen neuronalen Netzen zum Einsatz kommt. Ein Beispiel hierfür ist die Analyse und Erkennung von Mustern, deren Dauer beim Menschen typischerweise zwischen 100 und 140 $msec$ beträgt [TI89], obgleich das Bildsignal von der Retina bis zum Temporallappen mindestens zehn synaptische Bereiche durchlaufen muss[7]. Die Schlussfolgerung die sich daraus ergibt ist, dass jedes Neuron auf Basis von maximal zwei Pulsen der vorherigen Schicht eine Entscheidung treffen muss, womit sowohl Ratenkodierung (und damit künstliche neuronale Netze) als auch rekurrente Netze auszuschliessen sind.

[4] SpikeProp [BKL02], Tempotron [GS06], ReSuMe [Pon06], Chronotron [Flo12], SPAN [Moh+12]

[5] Gewichtung der Eingänge

[6] anders ist dies hingegen bei künstlichen neuronalen Netzen, da bei diesen keine Verzögerungszeiten existieren

[7] Photorezeptor $\rightarrow_1$ bipolare Zellen der Retina $\rightarrow_1$ Ganglion Zellen $\rightarrow_1$ Lateral Geniculate Nucleus $\rightarrow_1$ Layer IV Striate Cortex $\rightarrow_{1,2}$ Ausgangszellen des Striate Cortex $\rightarrow_{2,3}$ V2 $\rightarrow_{2,3}$ V4 $\rightarrow_{1+}$ inferotemporaler Cortex.

1.2. Aufgabenstellung

In dieser Arbeit gehen wir der Fragestellung nach, wie sich gleichermassen die *Topologie* und die *Gewichtung* von gepulsten neuronalen Netzen evolutionär entwickeln lässt. Unser Fokus liegt hierbei auf der Erzeugung komplexer Netze, also Netze, die bestimmte Strukturen wiederholt enthalten und somit eine gewisse Modularität aufweisen. Einsatzgebiet der so erzeugten Netze soll das maschinelle Lernen sein.

Ausgangspunkt ist die Arbeit von Johannes Schmidt [Sch09], in der eine hierarchische Beschreibung der strukturellen Ausprägung gewählt wurde um komplexe Netze beherrschbar zu machen. Hierarchie stellt für uns jedoch ein zutiefst menschliches Konzept dar, um Komplexität beherrschbar zu machen. Im Gegensatz dazu bilden sich viele Regelmässigkeiten in der Natur nur durch das emergente Zusammenwirken vieler kleiner einzelner Teile, ohne dass dabei ein genauer Plan des *Grossen Ganzen* erkennbar wird. Somit sehen wir in der Hierarchie, die sich auch im Titel dieser Arbeit wiederfindet, vielmehr ein Synonym für die Beherrschbarkeit von Komplexität, als eine genaue Vorgehensweise der strukturellen Beschreibung. Das Ziel dieser Arbeit sehen wir daher vielmehr in der Entwicklung eines Verfahrens, dass komplexe Netze beherrschbar macht, als in der Notwendigkeit der Verwendung von Hierachie in diesem Verfahren.

Weiterhin wollen wir der Frage nachgehen, ob sich prinzipiell Verfahren wie *Space Colonization* [RLP07] oder *Diffusionsbegrenztes Wachstum* [WS81] zur Ausprägung von Verbindungsstrukturen von gepulsten neuronalen Netzen eignen. Zudem soll der Einsatz von *Lindenmayer-Systemen* untersucht werden.

1.3. Gliederung der Arbeit

Zum besseren Verständnis der nachfolgenden Kapitel geben wir in Kapitel 2 eine kurze Einführung in die Themengebiete *Gepulste Neuronale Netze*, *Evolutionäre Algorithmen* und *Lindenmayer-Systeme*.

Im Anschluss daran betrachten wir in Kapitel 3 die Herausforderungen, die bei der evolutionären Erzeugung von neuronalen Netzen entstehen und beschreiben einige ausgesuchte Algorithmen für die evolutionäre Optimierung von Topologie und Gewichtung neuronaler Netze. Hierbei gehen wir ausschliesslich auf den Aspekt der Netzgenerierung ein und arbeiten am Ende des Kapitels die jeweiligen Vor- und Nachteil der vorgestellten Algorithmen heraus.

Einige alternative Ansätze und Ideen, wie zum Beispiel der Einsatz von *Diffusionsbegrenztem Wachstum* oder *Space Colonization* für die Ausbildung von Verbindungsstrukturen bei neuronalen Netzen werden in Kapitel 4 besprochen und auf ihre Eignung hin untersucht.

Aufbauend auf den in den vorangegangenen Kapiteln gewonnenen Erfahrungen, erarbeiten wir in Kapitel 5 unseren eigenen Ansatz für die Erzeugung und Optimierung

von gepulsten neuronalen Netzen, begründen die von uns getroffenen Entscheidungen und beschreiben diese im Detail. Eine genaue Evaluation unseres Verfahrens findet im Anschluss daran in Kapitel 6 statt.

Den Abschluss dieser Arbeit bildet Kapitel 7. Hier fassen wir unsere Ergebnisse nochmals zusammen und geben einen Ausblick auf zukünftige und weiterhin notwendige Arbeiten in diesem Bereich.

2. Grundlagen

2.1. Neuronale Netze

Im Folgenden geben wir eine kurze Einführung in den Aufbau und die Funktionsweise von biologischen Neuronen und beschreiben einige mathematische Modelle die zur Simulation eingesetzt werden können. An dieser Stelle verzichten wir auf die Beschreibung der *künstlichen neuronalen Netze*, basierend auf dem Perzeptron als Neuronenmodell, da diese gänzlich auf den zeitlichen Aspekt verzichten, und somit als Modell von biologischen Neuronen ungeeignet sind[1]. Auch gehen wir nicht auf die Frage ein, wie *gepulste neuronale Netze* effizient simuliert werden können. Dies war Thema einer vorherigen Arbeit und kann in [Neu08] nachgelesen werden.

2.1.1. Biologisches Modell

Abbildung 2.1 zeigt den schematischen Aufbau von biologischen Neuronen. Die vom Zellkörper (Soma) des Neurons auswachsenden Dendriten dienen vorwiegend der Reizaufnahme, während Axone der Weiterleitung ausgehender Reize dienen. Interneuronale Verbindungen entstehen durch eine synaptische Brücke (Synapse) zwischen Dendrit und Axon verschiedener Neuronen.

Überschreitet das Zellpotential eines Neurons eine durch mehrere Faktoren bestimmte Schwelle, so depolarisiert sich dieses (das Neuron *feuert*) indem es Ionenkanäle öffnet und so ein Aktionspotential entlang seines Axons an nachfolgende, postsynaptische Neuronen aussendet. Die Ausbreitung des Aktionspotentials im Axon erfolgt hierbei mit bis zu 100 Meter pro Sekunde, abhängig von Durchmesser und Isolierung.

Während die meisten Neuronen genau ein Axon ausbilden, mit Ausnahme einiger Nervenzellen denen keine Axone auswachsen, so ist die Anzahl der Dendriten und Synapsen sehr viel größer. Beispielsweise bildet eine einzelne Purkinje-Zelle bis zu 200 000 synaptische Kontakte aus (Abbildung 2.2). Axone und Dendriten unterscheiden sich zudem in ihrer Länge. Während Axone beim Menschen eine Länge von über 1 Meter erreichen können, so sind Dendriten mit nur einigen hundert Mikrometern vergleichsweise kurz.

[1] Ihr praktischer Nutzen ist dennoch unbestritten.

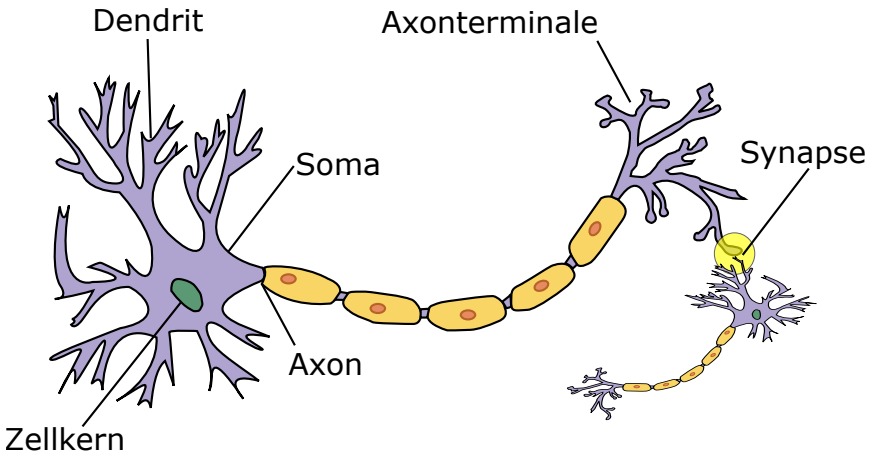


Abbildung 2.1.: Schematischer Aufbau biologischer Neuronen. Ein Neuron empfängt präsynaptische Signale durch seine vom Zellkörper baumförmig auswachsenden Dendriten, die mit Axonen anderer Neuronen durch Synapsen in Kontakt stehen. Überschreitet das Zellpotential eines Neurons eine durch mehrere Faktoren bestimmte Schwelle, so *feuert* dieses und leitet ein Aktionspotential in Form von elektrischen Impulsen entlang seines Axons an nachfolgende Neuronen weiter. Quelle: Wikipedia (CC BY-SA 3.0) modifiziert

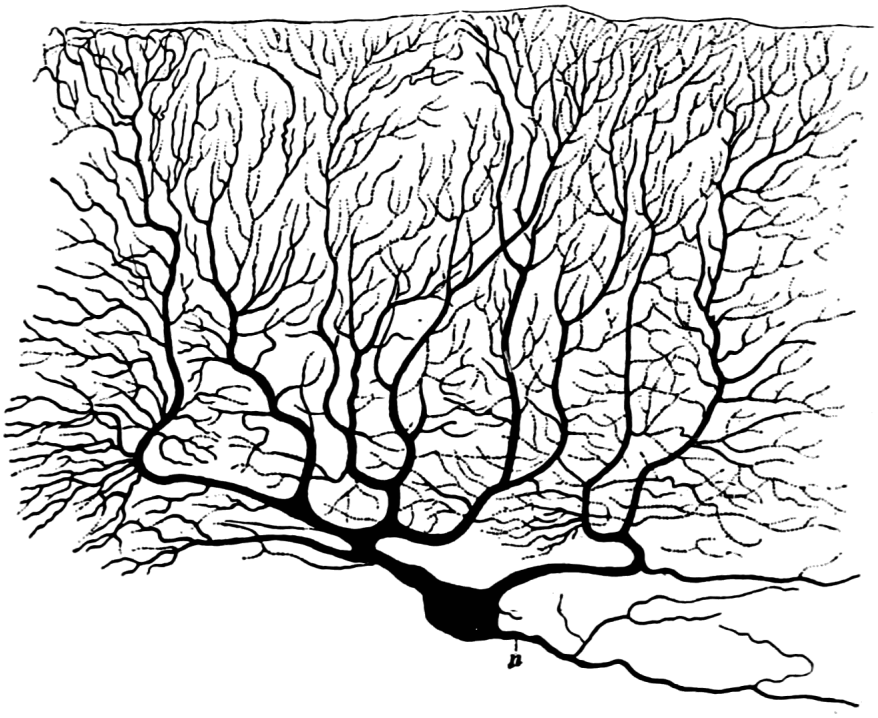


Abbildung 2.2.: Purkinje-Zelle des menschlichen Cerebellum. Quelle: Popular Science Monthly Volume 71/1907 (public domain)

2.1.2. Gepulste Neuronen Modelle

Es existieren mehrere verschiedene Modelle um das Verhalten von biologischen Neuronen nachzubilden. Diese unterscheiden sich im Wesentlichen in den folgenden Aspekten:

- Biologische Plausibilität des Modells
- Qualität des Modells
- Rechenaufwand

Die Qualität eines Modells lässt sich daran bewerten, ob sich mit diesem alle an realen Neuronen beobachteten Messungen beschreiben lassen können. Eine Auswahl einiger solcher Beobachtungen, die sich im dynamischen Antwortverhalten verschiedener Neuronentypen ausdrückt, ist in Abbildung 2.4 dargestellt. Im Folgenden gehen wir auf einige Modelle ein, die auf der numerischen Berechnung von Differentialgleichungen basieren. Weiterführende Details sind [IM08] zu entnehmen. Auf das von Gerstner beschriebene *Spike Response Modell* [GK02] gehen wir hier nicht ein.

2.1.3. Hodgkin-Huxley-Modell

Als Ergebnis der Forschungen am Riesenaxon des Tintenfisches, entstand bereits 1952 das Hodgkin-Huxley-Modell [HH52]. Dieses beschreibt detailliert die physiologischen Vorgänge an der Zellmembran eines Neurons durch den Zu- und Abfluss positiv geladener Kaliumionen (Ka^+) und negativ geladener Natriumionen (Na^-), sowie der Leckströme und des von aussen zugeführten Stromes. Es ist somit biologisch plausibel und kann alle Beobachtungen realer Neuronen nachbilden. Allerdings ist die Geschwindigkeit der Simulation dieses Modells aufgrund der Verwendung eines gekoppelten, nichtlinearen Differentialgleichungssystems vergleichsweise langsam. Izhikevich gibt diese in [Izh04] mit 1200 FLOPS[2] an. Im Vergleich dazu 5 FLOPS bei dem Integrate and Fire Modell, 10 FLOPS bei Resonate and Fire sowie 13 FLOPS bei dem Modell von Izhikevich.

2.1.4. Leaky Integrate and Fire

Das *Leaky Integrate and Fire* Modell beschreibt die Dynamik des Membranpotentials in Form einer einfachen RC-Schaltung (siehe Abbildung 2.3).

Der synaptische Eingangsstrom I teilt sich in einen resistiven und kapazitiven Anteil auf: $I = I_R + I_C$. Der resistive Teil ist gegeben durch das Ohmsche Gesetz $I_R = V/R_{leak}$, wobei V das Membranpotential (Spannung) darstellt. Der kapazitive Strom I_C ist gegeben durch die Kapazität $C = Q/V$ (mit der Ladung Q). Da $dQ/dt = I_C$ gilt, ergibt sich $CdV/dt = C\dot{V} = I_C$. Somit erhält man $I = I_R + I_C = V/R_{leak} + C\dot{V}$.

[2] Anzahl benötigter Fliesskommaoperationen zur Simulation von $1ms$

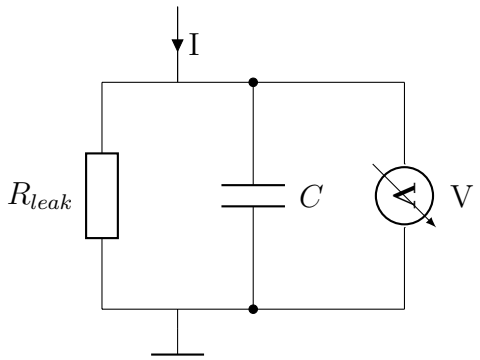


Abbildung 2.3.: Elektrisches Modell des Leaky Integrate and Fire Neurons in Form eines RC-Schwingkreises. I: Synaptischer Eingangsstrom, R_{leak}: Ohmscher Verlust, C: Membrankapazität, V: Membranpotential.

Teil man den Ohmschen Verlust in einen spannungsunabhängigen Verlust E_{leak} und einen spannungsabhängiggen Leitwert g_{leak} so ergibt sich daraus die in [IM08] beschrieben Form:

$$C\dot{V} = I - \overbrace{g_{leak}(V - E_{leak})}^{\text{Ohmscher Verlust}} \tag{2.1}$$

Ein Neuron feuert sobald das Membranpotential V den Schwellwert $E_{threshold}$ übersteigt, woraufhin V auf das Potential E_k zurückgesetzt wird.

Eine vereinfachte, abstraktere Form von Gleichung 2.1 erhält man durch entsprechende Skalierung der Parameter:

$$\dot{v} = b - v, \qquad \text{wenn } v = 1, \text{ dann } v \leftarrow 0 \tag{2.2}$$

Der Parameter v beschreibt hierbei das Membranpotential, b die äussere Erregung. Das Neuron ist erregbar bei $b < 1$ und feuert periodische Pulse bei $b > 1$. Ruhezustand ist bei $v = b$ (Nullstelle) gegeben. Der Schwellwert ist $v = 1$, das Rücksetzpotential $v = 0$.

2.1.5. Quadratic Integrate and Fire

Ersetzt man in Gleichung 2.2 $-v$ durch v^2 so erhält man das *Quadratic Integrate and Fire* Modell:

$$\dot{v} = b + v^2, \qquad \text{wenn } v = v_{peak}, \text{ dann } v \leftarrow v_{reset} \tag{2.3}$$

Ein Neuron feuert bei Überschreiten des Gipfelpunktes v_{peak}. Im Unterschied zum Integrate and Fire Modell handelt es sich bei v_{peak} allerdings um keinen echten Schwellwert. Es wird lediglich das Membranpontial an diesem Punkt zurückgesetzt. Würde man v weiter berechnen, so würde dieser unweigerlich gegen $+\infty$ streben und somit einen echten Impuls darstellen.

2.1.6. Resonate and Fire

Resonate and Fire [Izh01] ist eine zweidimensionale Erweiterung des *Integrate and Fire* Modells. Es basiert auf einem elektrischen LC-Schwingkreis:

$$C\dot{V} = I - g_{leak}(V - E_{leak}) - W \tag{2.4}$$
$$\dot{W} = (V - V_{1/2})/k - W \tag{2.5}$$

Die zweite Variable W beschreibt hierbei den Effekt, den der Membranstrom auf das Potential ausübt. Mit dem Resonate and Fire Modell lassen sich bestimmte Eigenschaften von biologischen Neuronen, wie gedämpfte Oszillationen, Resonanzfrequenzen oder postinhibitorische Rebound-Spikes nachbilden.

2.1.7. Modell von Izhikevich

Das Neuronenmodell von Izhikevich [Izh00; Izh+03] kann als Vereinigung des *Quadratic Integrate and Fire* und *Resonate and Fire* Modells angesehen werden. Es verwendet ein differenzielles Gleichungssystem mit zwei Variablen, dem Membranpotential v (in mV) und der Erholungsvariablen u, gegeben durch:

$$\dot{v} = 0.04v^2 + 5.0v + 140.0 - u + I_{syn} \tag{2.6}$$
$$\dot{u} = a(bv - u) \tag{2.7}$$

Und dem Rücksetzen nach einem Spike:

$$\text{wenn } v >= 30.0\,mV, \text{ dann } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.8}$$

Die Parameter a, b, c und d haben dabei folgende Bedeutung:

- a Erholungsrate.
- b Sensitivität der Erholungsvariable u in Bezug auf das Membranpotential v.
- c Rücksetzwert des Membranpotentials v nach einem Spike.
- d Rücksetzwert der Erholungsvariable u nach einem Spike.

Eine Reduktion auf nur einen Parameter $r \in [0, 1]$ beschreibt Izhikevich in [IGE04]. Erregende (exzitative) Neuronen lassen sich somit vereinfacht wie folgt darstellen:

$$a = 0.02, \quad b = 0.2, \quad c = -65.0 + 15.0\,r^2, \quad d = 8.0 - 6.0\,r^2 \tag{2.9}$$

Für hemmende (inhibitorische) Neuronen hingegen gelten folgende Werte:

$$a = 0.02 + 0.08\,r, \quad b = 0.25 - 0.05\,r, \quad c = -65.0, \quad d = 2.0 \tag{2.10}$$

Die Startwerte sind durch $v = -70.0\,mV$ und $u = -14.0$ gegeben, lassen sich jedoch je nach Neuronentyp auch variieren. Alle Parameter und Variablen wurden bereits entsprechend skaliert, sodass die Werte des Membranpotentials in mV mit Beobachtungen biologischer Neuronen übereinstimmen. Die Einheit der Zeit t ist hierbei in Millisekunden gegeben, der Eingangstrom I_{syn} in pA.

Mit dem Modell von Izhikevich können alle in [Izh04] aufgeführten Beobachtungen kortikaler Neuronen simuliert werden (siehe Abbildung 2.4). Zudem ist es ungefähr 100-mal schneller als die Simulation des Hodgkin-Huxley-Modells. Allerdings beschreibt es nicht die physiologischen Abläufe und ist somit nicht biologisch plausibel. Dies erkennt Izhikevich jedoch als Vorteil an, da die genauen chemischen Abläufe im Inneren eines Neurons, die für die Bildung und Parameterisierung eines biologisch plausiblen Modells notwendig sind, nur sehr ungenau zu messen sind, wohingegen der äussere Effekt, der durch diese inneren Prozesse entsteht, sehr gut messbar ist.

2.1.8. Gepulste Neuronale Netze

Erst die Verschaltung einzelner Neuronen zu neuronalen Netzen erlaubt die Berechnung komplexer Funktionen. Bei biologischen neuronalen Netzen entstehen Verbindungen zwischen den Dendriten und Axonen. Der Punkt an dem sich Dendrit und Axon berühren wird Synapse genannt. Dendriten dienen nicht nur der Reizaufnahme, ihre Morphologie bestimmt zudem auch einen Teil der Funktionalität des Neurons [CRT13].

Während Modelle existieren, die die genaue Morphologie der Dendriten berücksichtigt (Kabelmodell), so verwenden wir zur schnellen Simulation eine vereinfachte Darstellung. Unser Modell unterscheidet zudem nicht zwischen Dendriten, Axone und Synapsen, sondern vereinigt die Eigenschaften dieser drei Bestandteile eines Neurons in einer sogenannten *künstlichen Synapse*. Da wir im Folgenden nur künstliche Synapsen betrachen, und nicht biologische Synapsen, nennen wir diese einfach nur *Synapsen*.

Neuronen unseres Modells kommunizieren somit über Synapsen mit anderen Neuronen. Eigenschaften dieser Synapsen sind:

- Verzögerungszeit
- Effizienz

Die Verzögerungszeit einer (künstlichen) Synapse geht hauptsächlich auf die Länge des biologischen Axons, sowie dessen Ummantelung und Durchmesser zurück. Die Effizienz einer (künstlichen) Synapse bestimmt, wie stark das vom Axon ausgehende Aktionspotential gedämpft wird, bis es das nachfolgende Neuron erreicht hat. Biologische Synapsen übermitteln das Signal entweder chemisch oder elektrisch. Ein Teil der Effizienz geht somit auf die Art der biologischen Synapse zurück.

Ein (künstliches) Neuron kann in diesem Modell mit beliebig vielen nachfolgenden Neuronen verbunden sein. Jede dieser Verbindungen wird durch eine eigene Synapse

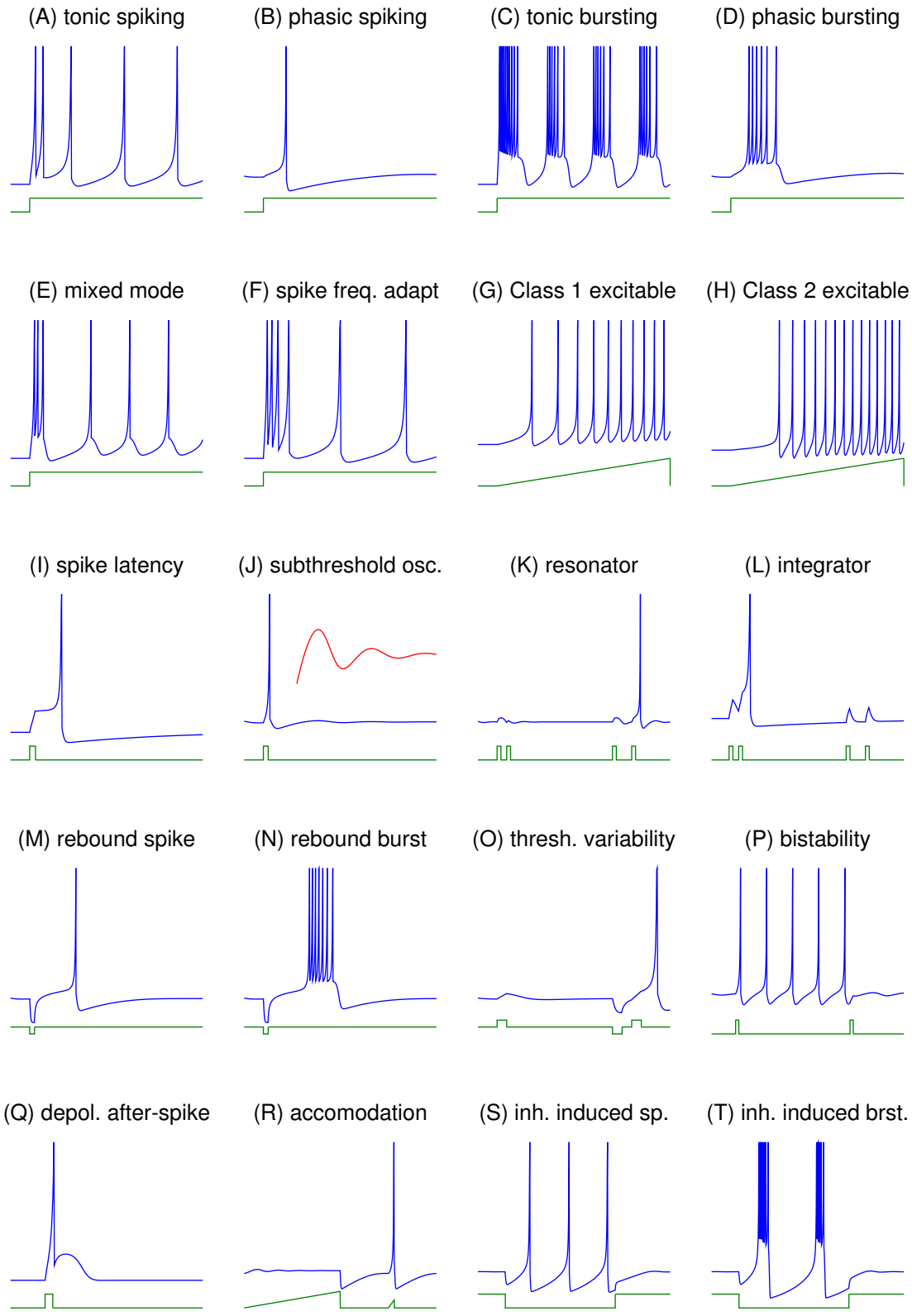


Abbildung 2.4.: Simulation des dynamischen Antwortverhaltens verschiedener kortikaler Neuronentypen mit dem Neuronenmodell von Izhikevich. Blau: Membranpontial (mV). Grün: Synaptischer Eingangsstrom. Rot: Vergrösserung. MATLAB Code siehe `www.izhikevich.com`.

repräsentiert, d.h. mit getrennten Werten für die Verzögerungszeit und Effizienz. Es kann zudem auch mit sich selbst verbunden sein (rekurrent). Dies ist möglich, da die Synapse eine Verzögerungszeit > 0 hat.

Um ein solches Modell zu simulieren, muss bei dem Feuern eines Neurons für jedes nachfolgende Neuron festgehalten werden, wann das Aktionspotential dieses erreicht, und wie stark das Signal abgeschwächt wird. Am einfachsten lässt sich dies durch eine Prioritätswarteschlange realisieren. Bei der Berechnung des neuen internen Zustands eines Neurons muss nun zusätzlich überprüft werden, ob ein synaptischer Strom I_{sys}, ausgelöst durch das Feuern eines oder mehrerer präsynaptischen Neuronen, im aktuell simulierten Zeitpunkt anliegt.

In biologischen neuronalen Netzes konnte beobachtet werden, dass sich die Effizienz einer Synapse erhöht, wenn das Feuern des präsynpatischen Neurons A dazu führt, dass das nachfolgende (postsynaptische) Neuron B innerhalb eines begrenzten Zeitraumes auch feuert. Diesen Effekt nennt man Verstärkungslernen, Hebbsches Lernen oder auch *Spike-Timing Dependent Plasticity* (STDP)[3].

Dieser Effekt lässt sich simulieren und zum Einlernen der Synapsengewichte verwenden. Eine genauere Beschreibung findet sich beispielsweise in [IGE04]. Weitere Effekte, wie z.B. *Long-Term Potentiation* (LTP), treten in biologischen neuronalen Netzen auf, und spielen darüber hinaus eine wichtige Rolle für den Lernvorgang.

[3] Diese Kausalität, auf die ein Teil unseres Gehirns basiert, kann auch als Erklärung für unsere logische Denkweise herangenommen werden.

2.2. Evolutionäre Algorithmen

Evolutionäre Algorithmen sind Heuristiken, die im Bereich der Optimierung eingesetzt werden. Ihr charakteristisches Merkmal ist die Verwendung von Operationen zur *Selektion* und zur *Reproduktion*. Während die Selektion den Verbleib einzelner Lösungen in der Lösungsmenge regelt, erlaubt die Reproduktion die Bildung neuer Lösungen. Dieses Konzept beruht auf dem Prinzip der natürlichen Auslese, die in der Darwin'schen Evolutionstheorie als *Survivial of the Fittest* beschrieben wird.

Zu den im Laufe der Jahre entstandenen evolutionären Algorithmen gehören u.a. die Evolutionsstrategie [Sch65; BS02], der Genetische Algorithmus [Hol62; Hol75] oder das Genetische Programmieren [Koz92]. Die Differenzierung verschiedener evolutionärer Algorithmen kann derweil anhand folgender Merkmale vorgenommen werden:

- Selektionsoperation
- Repräsentation bzw. Kodierung der Lösung
- Reproduktionsoperatoren
- Auswahlmechansimus für Reproduktion

Den allgemeinen Ablauf eines evolutionären Algorithmus zeigt Abbildung 2.5. Die erste Iteration stellt einen Sonderfall dar. Gehen wir daher davon aus, dass bereits aus einer vorherigen Iteration eine bewertete Population aus $\mu + \lambda$ Lösungen (bzw. Individuen oder Genome) vorliegt. Mit μ bezeichnen wir die gewünschte Populationsgrösse, während λ die Anzahl der erzeugten Nachkommen repräsentiert. Ziel der Selektion ist es, die gewünschte Populationsgrösse aufrechtzuerhalten. Der Selektionsmechanismus wählt nun anhand der Bewertung der Individuen und möglicher weiterer Kriterien (häufig unterstützt durch Zufall) μ der $\mu + \lambda$ Individuen aus, die in die nächste Generation übernommen werden. Aus dieser neuen Generation werden durch Reproduktion wieder λ Nachkommen erzeugt. Hierbei findet eine weitere Form der Selektion statt. Die sogenannte *sexuelle Selektion* bestimmt, aus welchen Individuen anhand der Reproduktionsoperatoren Nachkommen erzeugt werden. Dies geschieht meisst eingeschlechtlich durch *Mutation* oder zweigeschlechtlich durch *Kreuzung* (Crossover). Ergebnis der Reproduktion sind λ Nachkommen. Einige evolutionäre Algorithmen vereinigen die Menge der Nachkommen mit der Elternpopulation, bezeichnet durch $(\mu + \lambda)$, während bei anderen die Nachkommen die Elternpopulation komplett ersetzen, abgekürzt (μ, λ). Wir haben oben den Fall eines $(\mu + \lambda)$ Algorithmus beschrieben. Algorithmen die (μ, λ) einsetzen, kopieren meisst einen Teil der elitären Individuen aus der Elternpopulation in die Population der Nachkommen, um gute bereits gefundene Lösungen nicht zu verlieren. Bei $(\mu+\lambda)$ ist dies aufgrund der Vereinigung nicht nötig. Die neu erzeugten Nachkommen müssen anschliessend bewertet werden. Nun liegt wieder eine vollständig bewertete Population vor und es kann geprüft werden, ob darin eine Lösung existiert, die dem Zielkriterium entspricht, d.h. *gut genug* ist. In diesem Fall kann der evolutionäre Algorithm an dieser Stelle abgebrochen werden. Andernfalls wird eine weitere Iteration durchgeführt.

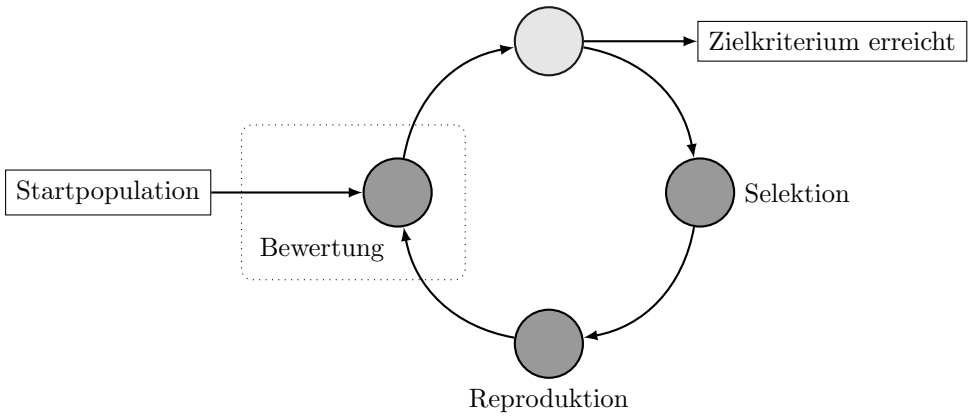


Abbildung 2.5.: Allgemeiner Ablauf eines evolutionären Algorithmus. Erzeugen einer Startpopulation. Bestimmen der Fitness (Bewertung). Optimierungsziel erreicht? Selektion guter Individuen. Erzeugen von Nachkommen. usw.

2.2.1. Genetischer Algorithmus

Der Genetische Algorithmus von John Holland zeichnet sich durch eine binäre Repräsentation der Lösung aus. In diesem Kontext verwenden wir oft den Begriff Genom oder Genotyp, wenn wir von der Repräsentation der Lösung sprechen, sowie Phenotyp, als Synonym für die Lösung.

Die binäre Repräsentation macht in der Regel eine umkehrbare Abbildung zwischen Genotyp und Phenotyp notwendig. Zudem ist die Länge des Genoms meisst unveränderlich.

Als Reproduktionsoperatoren kommen Mutation und Kreuzung (Crossover) zum Einsatz. Bei der Mutation wird jedes Bit mit einer bestimmten Wahrscheinlichkeit umgedreht. Zur Kreuzung stehen mehrere Operatoren zur Auswahl. Einige davon sind:

- Lineares 1-Punkt Crossover
- Lineares 2-Punkt Crossover
- Lineares n-Punkt Crossover
- Uniformes Crossover

Bei dem linearen 1-Punkt Crossover werden zwei Genome $(x_1, \ldots, x_n)$ und $(y_1, \ldots, y_n)$ der Länge n an einer zufällig gewählten Position $1 < p < n$ geschnitten und daraus zwei Nachkommen $(x_1, \ldots, x_p, y_{p+1}, \ldots, y_n)$ und $(y_1, \ldots, y_p, x_{p+1}, \ldots, x_n)$ erzeugt. Analog dazu werden bei 2-Punkt Crossover die Genome an 2 Schnittpunkten geschnitten (bzw. bei Mehrpunkt-Crossover an mehreren).

Uniformes Crossover verknüpft zwei Genome stellenweise mit einer Funktion $f(x_i, y_i)$. Daraus resultieren ein oder mehrere Genome der Form $(f(x_1, y_1), \ldots, f(x_n, y_n))$. Die Funktion f kann hierbei probabilistisch sein, d.h. zufällig eines der Elemente x_n oder y_n zurückliefern.

2.2.2. Genetisches Programmieren

Das Genetische Programmieren unterscheidet sich im Wesentlichen vom Genetischen Algorithmus in der zugrundeliegenden Repräsentation des Genoms. John R. Koza erkennt als einer der Ersten die Signifikanz die von der Kodierung ausgeht, '... because it severly limits the window by which the system observes its world' ([KR91]). Koza verwendet anstelle einer binären Repräsentation eine Strukturierung in Form von Listen oder Bäumen. Bei der Repräsentation als Liste können die bekannten genetischen Operatoren des Genetischen Algorithmus übernommen werden. Für die Kreuzung von Genomen, die durch einen Baum repräsentiert werden, verwendet Koza das Austauschen von Teilbäumen zwischen den Genomen.

2.2.3. Selektionsverfahren

Selektion kommt an zwei verschiedene Stellen eines evolutionären Algorithmus zum Einsatz. Zum einen zur Begrenzung der Populationsgrösse, zum anderen zur Auswahl der für die Reproduktion zugelassener Individuen bzw. der Partnerwahl bei mehrgeschlechtlicher Reproduktion. Einige Verfahren hierfür sind:

- Selektion durch Abschneiden
- Selektion proportional zur Fitness
- Rang-basierte Selektion
- Elitäre Selektion
- Tournament Selektion

2.2.3.1. Selektion durch Abschneiden

Die Selektion durch Abschneiden (Truncation Selection) stellt die einfachste Form der Selektion dar. Hierbei wird die Population anhand der Fitness sortiert und die besten k Individuen in die neue Generation übernommen.

2.2.3.2. Selektion proportional zur Fitness

Bei der Fitness-proportionalen Selektion ist die Selektionswahrscheinlichkeit p_i eines Individuums i mit Fitness f_i proportional zur Gesamtsumme der Fitnesswerte aller Individuen der Population, d.h. $p_i = f_i / \sum_j f_j$. Nachteile dieser Methode sind: Sogenannte Superindividuen, d.h. Individuen, deren Fitnesswert größer ist als die aller anderer Individuen, jedoch mit gemessen am globalen Optimum niedriger Fitness, werden mit hoher Wahrscheinlichkeit ausgewählt. Dies führt dazu, dass Superindividuen nach einigen Iterationen viele andere initial vorhandene Individuen dominieren. Dies hat meisst die vorzeitige Konvergenz aufgrund des Mangels an Diversität zur

Folge. Ein weiterer Nachteil tritt auf, wenn Individuen sehr ähnliche Fitnesswerte aufweisen. Damit ist die Selektionswahrscheinlichkeit dieser Individuen sehr ähnlich und es wird schwer die besten Individuen auszuwählen. Mögliche Lösung hierfür ist die Änderung der Fitnessfunktion oder Wahl einer anderen Selektionsmethode.

2.2.3.3. Rang-basierte Selektion

Bei der Rang-basierten Selektion wird den Individuen ein Rang anhand ihrer Fitness zugeordnet. Die Wahrscheinlichkeit zur Selektion ist nun nicht mehr abhängig vom Fitnesswert, sondern vom Rang. Dadurch wird das Problem der Superindividuen gelöst, welches bei der Fitness-proportionalen Selektion auftritt.

2.2.3.4. Elitäre Selektion

Bei der Elitären Selektion werden immer $k\,\%$ der besten Individuen in die neue Generation übernommen. Der Rest der Population wird durch ein anderes Selektionsverfahren bestimmt. Durch Elitäre Selektion kann gewährleistet werden, dass die besten Lösungen nicht verloren gehen.

2.2.3.5. Tournament Selektion

Bei der Tournament Selektion wird durch k-maliges Würfeln das beste Individum ermittelt und ausgewählt. Tournament Selektion kann sowohl für die Reduktion der Populationsgrösse verwendet werden, als auch zur Selektion der Individuen bei der Reproduktion. Der Selektionsdruck kann durch Wahl von k angepasst werden. Für $k = 1$ enstpricht die Tournament Selektion der Fitness-proportionalen Selektion.

2.2.4. Mehrwertige Optimierung

Setzt sich der Fitnesswert f einer Lösungsmenge $\mathcal{P}$ aus mehreren unabhängigen Werten zusammen, d.h. $f : \mathcal{P} \to (x_1, \ldots, x_n)$, so spricht man von mehrwertiger, multimodaler oder Pareto-Optimierung (engl. multiobjective optimization).

Zwei bekannte Verfahren aus diesem Bereich sind NSGA-II [Deb+02] (Non-Dominated Sorting Genetic Algorithm) und SPEA-II [Zit+01] (Strength Pareto EA). Im Folgenden wollen wir NSGA-II exemplarisch betrachten.

2.2.4.1. NSGA-II

Anstelle des einfachen Vergleichs zweier numerischer skalarer Fitnesswerte, wie bei der einwertigen Optimierung, definiert NSGA-II auf den mehrwertigen Fitnesswerten eine Dominanz-Relation (Definition 1). Die Individuen werden anhand dieser

Dominanz-Relation in nicht-dominierende Pareto-Fronten $\mathcal{F}_i$ eingeteilt, sodass folgendes gilt: $\forall i < j : \forall x \in \mathcal{F}_i : \forall y \in \mathcal{F}_j : x \prec y$. Dies bedeutet, $\mathcal{F}_1$ enthält alle Individuen, deren Fitness nicht dominiert wird. $\mathcal{F}_2$ hingegen wird ausser von den Individuen aus $\mathcal{F}_1$ von keinen weiteren Individuen dominiert, usw. Abbildung 2.6 veranschaulicht die Pareto-Fronten bei einer zweiwertigen Fitness.

Um aus den $\mu+\lambda$ Individuen der Population $\mathcal{P}_1$ die nächste Generation $\mathcal{P}_2$, bestehend aus μ Individuen zu bilden, geht NSGA-II wie in Algorithmus 1 beschrieben vor. Die Crowindg-Distanz einer Population $\mathcal{P}$, bestehend aus Individuen mit mehrwertigen Fitnesswerten $x = (x_1, \ldots, x_k)$, wird hierbei wie in Algorithmus 2 beschrieben berechnet.

Definition 1 (Dominanz-Relation von NSGA-II) *Seien $x = (x_1, \ldots, x_n)$ und $y = (y_1, \ldots, y_n)$ zwei multimodale Fitnesswerte mit jeweils n Kriterien (Objektiven). x dominiert y (abk. $x \prec y$) $\iff (\forall_{i \in \{1..n\}} x_i \leq y_i) \wedge (\exists_{i \in \{1..n\}} x_i < y_i)$. Wir nehmen an, dass die Minimierung der Kriterien das Ziel ist, d.h. ein niedrigerer Wert in dem jeweiligen Kriterium bedeutet eine bessere Fitness.*

Algorithmus 1 Selektion bei NSGA-II

```
1: berechne Pareto-Fronten F_i von P_1
2: P_2 ← ∅
3: for i ← 1, |F| do                          ▷ Für jede Pareto-Front
4:     if |P_2 ∪ F_i| ≤ μ then                ▷ Wenn F_i vollständig in P_2 passt
5:         P_2 ← P_2 ∪ F_i                    ▷ dann füge F_i zu P_2
6:     else
7:         Sortiere F_i anhand der Crowding-Distanz
8:         k = μ − |P_2|                      ▷ Anzahl noch ausbleibender Individuen
9:         P_2 ← P_2 ∪ F_i[1..k]              ▷ Füge die ersten k Individuen aus F_i zu P_2
10:        break                              ▷ Ende. P_2 enthält nun exakt μ Individuen.
11:    end if
12: end for
```

Algorithmus 2 Crowding-Distanz $\mathcal{C}$ der Individuen aus $\mathcal{P}$ bei NSGA-II

procedure CROWDINGDISTANCE($\mathcal{P}$)
  Sei k die Kardinalität der Fitnesswerte $x = (x_1, \ldots, x_k)$
  Sei $n = |\mathcal{P}|$ die Anzahl der Individuen in $\mathcal{P}$
  for all $p \in \mathcal{P}$ **do**
    $\mathcal{C}(p) \leftarrow 0.0$
  end for
  for $i \leftarrow 1, k$ **do** ▷ Für jedes Kriterium i
    Sortiere $\mathcal{P}$ anhand des Fitnesswertes x_i
    $\mathcal{C}(\mathcal{P}[1]) \leftarrow \infty, \quad \mathcal{C}(\mathcal{P}[n]) \leftarrow \infty$ ▷ Randpunkte
    $norm \leftarrow k \cdot |\mathcal{P}[n] - \mathcal{P}[1]|$ ▷ Normierte Distanz zw. Minima und Maxima
    if $norm > 0.0$ **then**
      for $j \leftarrow 2, n-1$ **do**
        $dist \leftarrow |\mathcal{C}(\mathcal{P}[j+1]) - \mathcal{C}(\mathcal{P}[j-1])|$ ▷ Distanz benachbarter Punkte
        $\mathcal{C}(\mathcal{P}[j]) \leftarrow \mathcal{C}(\mathcal{P}[j]) + dist/norm$
      end for
    end if
  end for
end procedure

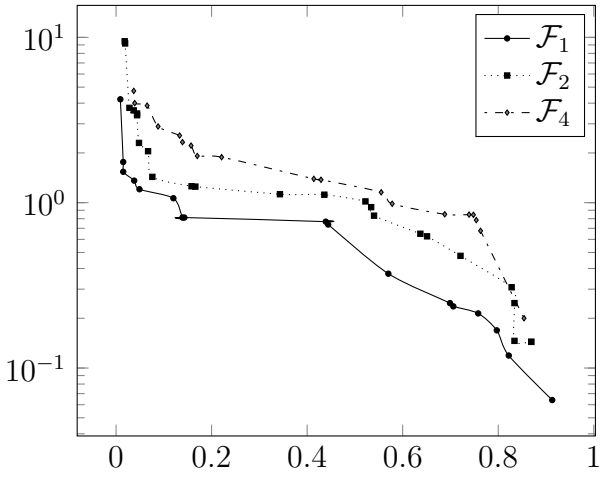


Abbildung 2.6.: Pareto-Fronten $\mathcal{F}_1$, $\mathcal{F}_2$ und $\mathcal{F}_4$ bei der Optimierung des ZDT1 Problems 2. Ordnung. ZDT wird als Standardbenchmark zum Vergleich multimodaler evolutionärer Algorithmen eingesetzt.

2.3. Lindenmayer Systeme

Ein Lindenmayer-System (L-System) ist ein paralleles Textersetzungssystem. Es wurde 1968 [Lin68] von dem ungarischen theoretischen Biologen Aristid Lindenmayer als mathematischer Formalismus zur Beschreibung von biologischen Wachstumsprozessen filamentöser Organismen vorgestellt. Einen weiten Verbreitungsgrad haben L-Systeme daher hauptsächlich im Bereich der Modellierung von biologischen Organismen erlangt.

Im Unterschied zu Chomsky-Grammatiken werden bei einem L-System die Ersetzungsregeln nicht sequentiell, sondern parallel auf allen Zeichen des Wortes simultan angewendet.

Über den Rahmen der folgenden kurzen Einführung in die verschiedenen Typen von L-Systemen hinausgehende Beispiele, Grafiken und Informationen finden sich in dem sehr zu empfehlenden Buch *The Algorithmic Beauty of Plants* [PL96].

Definition 2 (Kontextfreies 0L-System) *Sei Σ ein Alphabet, Σ^* die Menge aller Wörter über Σ, und $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$ die Menge der nicht-leeren Wörter über Σ. Ein kontextfreies (String-) 0L-System ist definiert durch $G = \langle \Sigma, \omega, P \rangle$, mit dem Alphabet Σ des Systems, dem nicht-leeren Wort $\omega \in \Sigma^+$ als Axiom, und $P \subset \Sigma \times \Sigma^*$ einer endlichen Menge an Produktionsregeln. Eine Produktion $(\alpha, \chi) \in P$ wird auch geschrieben als $\alpha \to \chi$, wobei α Vorgänger und χ Nachfolger genannt wird. Es wird angenommen, dass für jedes $\alpha \in \Sigma$ ein Nachfolger χ existiert. Ist dies nicht der Fall, so wird implizit angenommen, dass hierfür die Identität $id : \alpha \to \alpha$ zu P hinzugefügt wird.*

Definition 3 (Deterministisches D0L-System) *Ein 0L-System ist deterministisch (D0L) $\Leftrightarrow$ für jeden Vorgänger α einer Produktion P genau ein Nachfolger χ existiert.*

L-Systeme werden schrittweise, ausgehend von dem Axiom (oder Startwort) ω, entwickelt. Das L-System in folgendem Beispiel 1 beschreibt die fraktale Koch-Kurve aus 2.7.

Beispiel 1 (L-System für Kochkurve)

$$
\begin{aligned}
G &= \langle \Sigma, \omega, P \rangle \\
\Sigma &= \{F, +, -\} \\
\omega &= F \\
p_0 &= F \to F - F + +F - F \\
P &= \{p_0\}
\end{aligned}
$$

Die schrittweise Entwicklung ($S_n \subset \Sigma^*$) startet mit dem Axiom ($\omega = F$). Durch paralleles Anwenden der Produktionsregeln P auf S_1 erhält man im zweiten Entwicklungsschritt S_2. Durch Wiederholen des gleichen Vorganges erhält man S_3 usw. Die parallelen Ersetzungen in S_3 sind dabei der Übersichtlichkeit halber geklammert dargestellt; sie entsprechen jeweils dem Nachfolger der Regel p_0.

$$
\begin{aligned}
S_1 &= F \\
S_2 &= F - F + +F - F \\
S_3 &= \overbrace{F - F + +F - F} - \overbrace{F - F + +F - F} + + \overbrace{F - F + +F - F} - \overbrace{F - F + +F - F}
\end{aligned}
$$

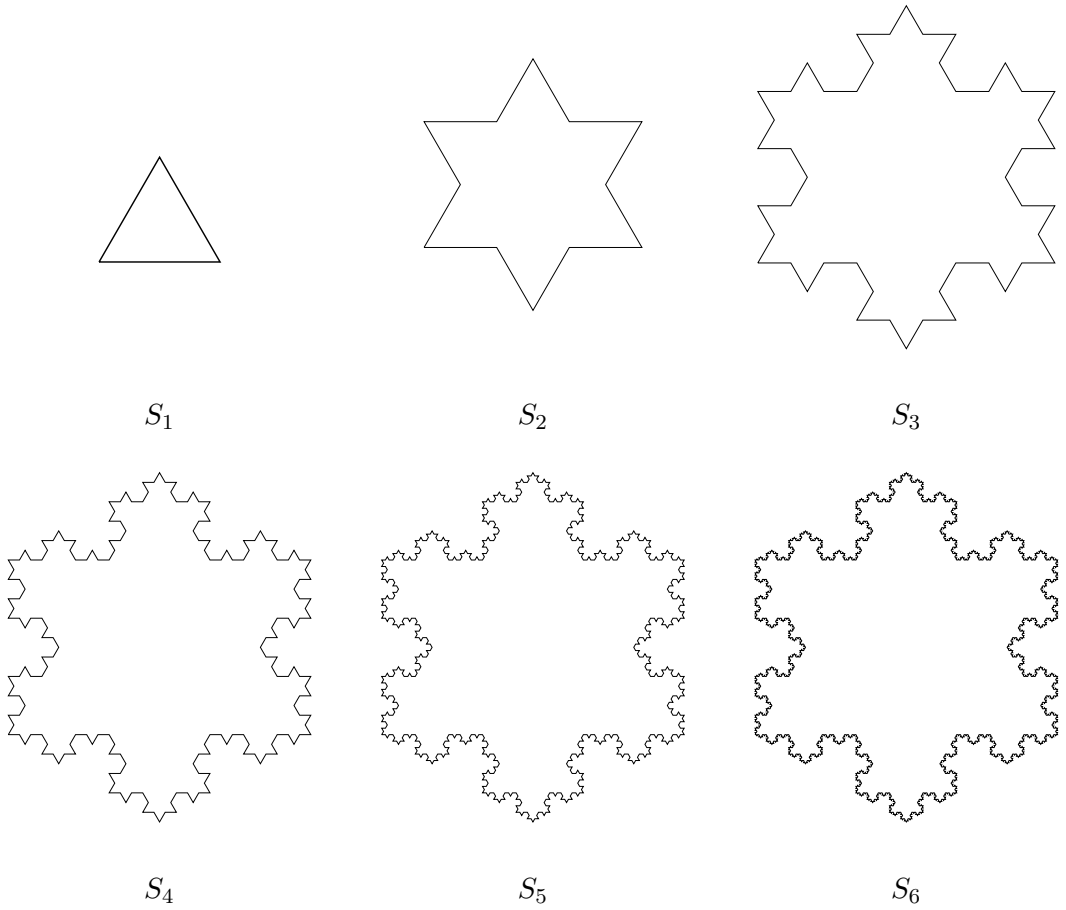


Abbildung 2.7.: Erste 6 Iterationen der Koch-Kurve mit $\phi = 25°$ und $\delta = 10$

2.3.1. Grafische Interpretation von L-Systemen

Eine beliebte Interpretation der erzeugten Wörter eines L-Systems verwendet die sogenannte Turtle-Grafik, die ihren Ursprung in der Programmiersprache LOGO

hat. In der einfachsten (nicht-parametrischen) Form, werden die Zeichen als Befehle eines Zeichenstiftes (in LOGO dargestellt als kleines, schildkrötenähnliches Dreieck, daher der Name *Turtle*) interpretiert, der bewegt, rotiert und mit dem gemalt werden kann. Dabei rotiert der Befehl + und − die Richtung des Zeichenstiftes um einen fest definierten Winkel ϕ nach links bzw. rechts. Der Befehl `F` malt mit dem Stift einen Strich der Länge δ in die eingestellte Richtung. `f` bewegt den Stift um δ Einheiten in die eingestellte Richtung ohne zu zeichnen. Alle weiteren Zeichen werden von dem Grafik-Interpreter ignoriert, sodass auch nicht-terminale Hilfzeichen im L-System verwendet werden können.

Mit $\phi = 25°$ und $\delta = 10$ erhält man eine Kochkurve wie in Abbildung 2.7. Dabei lässt sich sehr schön die fraktale Selbstähnlichkeit erkennen, denn das Dreieckmuster aus S_1 wiederholt sich in gedrehter Form in jedem Seitenelement.

Entsprechend der Interpretation im 2-dimensionalen Raum, existieren Erweiterungen für den 3-dimensionalen Raum. *Geklammerte L-Systeme* (*Bracketed L-system*) interpretieren zudem die eckigen Klammern `[` und `]`. Hierbei wird der aktuelle Zeichenzustand (Richtung und Position) entweder gespeichert oder wiederhergestellt. Dies ermöglicht es auch verzweigende Strukturen wie z.B. Bäume mit einem L-System darzustellen (siehe Beispiel 2 und Abbildung 2.8).

Beispiel 2 (L-System für fraktale Pflanze)

$$
\begin{aligned}
G &= \langle \Sigma, \omega, P \rangle \\
\Sigma &= \{F, X, +, -, [,]\} \\
\omega &= X \\
p_0 &= X \to F - [[X] + X] + F[+FX] - X \\
p_1 &= F \to FF \\
P &= \{p_0, p_1\}
\end{aligned}
$$

2.3.2. Kontextsensitive L-Systeme

Definition 4 (Kontextsensitives IL-System (1L oder 2L)) *Sei $G = \langle \Sigma, \omega, P \rangle$ ein 0L-System. Bei einem kontextsensitiven IL-System wird zusätzlich zum Vorgänger α einer Regel $(\alpha, \chi) \in P$ der linke ($\alpha_l \in \Sigma^*$) bzw. rechte Kontext ($\alpha_r \in \Sigma^*$) betrachtet. Eine Produktionregel hat dabei die Form $\alpha_l < \alpha > \alpha_r \to \chi$. Ist der Kontext einseitig (links oder rechts), dann spricht man von einem 1L-System, bei beidseitigem Kontext von einem 2L-System. Allgemein spricht man von einem (k, l)-System wenn der linke Kontext ein Wort der Länge k ist, sowie der rechte Kontext ein Wort der Länge l. Eine Produktionsregel wird nur bei übereinstimmendem Kontext angewendet.*

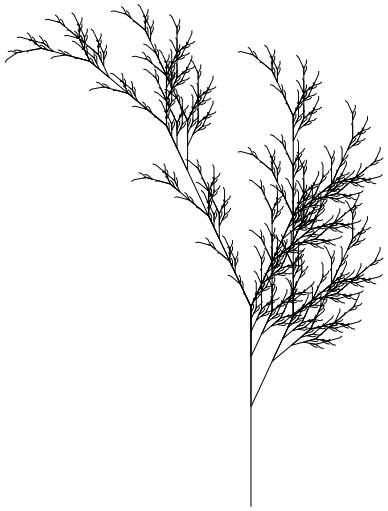


Abbildung 2.8.: Fraktale Pflanze nach 6 Iterationen (Beispiel 2)

Damit kontextsensitive L-Systeme deterministisch sind, ist entweder eine Reihenfolge auf den Produktionsregeln definiert oder die Regel mit der größten Übereinstimmung wird verwendet. Bei einem *geklammerten*, kontextsensitiven L-System wird zudem auf korrekte Klammerung im Kontext bzw. über Kontextgrenzen hinweg geachtet.

Im Unterschied zu 0L-Systemen ist es mit IL-Systemen möglich, Signale durch das Wort *wandern* zu lassen (siehe Beispiel 3). IL-Systeme (auf Zeichenketten) sind damit äquivalent zu eindimensionalen, unendlichen Zellularautomaten mit entsprechend grosser Nachbarschaft.

Beispiel 3 (Signal-Übertragung mit 1L-System)

$$\begin{aligned} \omega &= baaaaaaaa \\ p_0 &= b < a \rightarrow b \\ p_1 &= b \rightarrow a \end{aligned}$$

Die ersten Ableitungen sind dabei wie folgt:

baaaaaaaa
abaaaaaaa
aabaaaaaa
aaabaaaaa
...

2.3.3. Stochastische L-Systeme

In einem stochastischen L-System kann es zu einem gegebenen Zeichen α mehrere Produktionsregeln $(\alpha, \chi) \in P$ geben, d.h. der Nachfolger χ ist nicht eindeutig. Jeder dieser Regeln ist eine Wahrscheinlichkeit $\pi \in [0, 1)$ zugeordnet, mit der Eigenschaft dass ihre Summe genau 1 beträgt. Die Wahl der anzuwendenden Regel erfolgt damit nicht-deterministisch anhand der so definierten Zufallsverteilung.

2.3.4. Parametrische L-Systeme

Bei parametrischen L-Systemen [Han92] sind die Zeichen eines Wortes zusätzlich mit Parameterwerten oder Ausdrücken behaftet. Dabei ist die Anzahl der Parameter vom Zeichen abhängig. In Beispiel 4 haben die Zeichen aus Σ_0 keine Parameter, die aus Σ_1 genau einen. Die Produktionsregeln eines parametrischen L-Systems werden zudem um freie Variablen erweitert. Bei dem Matching einer Regel werden diese dann an einen konkreten Wert gebunden und im Nachfolger durch den errechneten Wert ersetzt. Im Beispiel 4 wird bei der Ersetzung von $A(300.0)$ die Regel $A(x)$ aktiv und damit der Wert 300.0 an die freie Variable x gebunden. Im Nachfolger wird x entsprechend durch 300.0 ersetzt. Diese führt nach der ersten Iteration zu folgendem Term:

$$F(300.0)[+A(206.04)][-A(206.04)]$$

Zudem sind den Produktionsregeln Bedingungen zugeordnet, die erfüllt sein müssen damit die Regel angewendet werden kann. Im gezeigten Beispiel ist dies $\{true\}$, d.h. die Bedingung ist immer erfüllt. Nimmt man in der Bedingung auf die freie Variable Bezug, so lässt sich ein Abbruchkriterium definieren (z.B. $\{a > 0\}$). Ausserdem ist es bei parametrischen L-Systemen erlaubt, mehrere Nachfolger für ein Zeichen zu definieren, mit jedoch unterschiedlichen Bedingungen, z.B.

$$A(x) : \{x > 0\} \rightarrow F(x) \qquad\qquad A(x) : \{x < 0\} \rightarrow f(x)$$

Im Falle einer grafischen Interpretation des parametrischen Wortes durch Turtle-Grafik werden auch die Parameterwerte verwendet. Beispielsweise beschreibt der Parameter x in $F(x)$ die Länge der Zeichenbewegung. Dies führt für das Beispiel 4 zu Abbildung 2.9.

Beispiel 4 (Parametrisches L-System)

$$\begin{aligned}
\Sigma_n &= \{\Sigma_0, \Sigma_1\} \\
\Sigma_0 &= \{+, -, [,]\} \\
\Sigma_1 &= \{A, F\} \\
\omega &= A(300.0) \\
p_0 &= A(x) : \{true\} \rightarrow F(x)[+A(\frac{x}{1.456})][-A(\frac{x}{1.456})] \\
P &= \{p_0\}
\end{aligned}$$

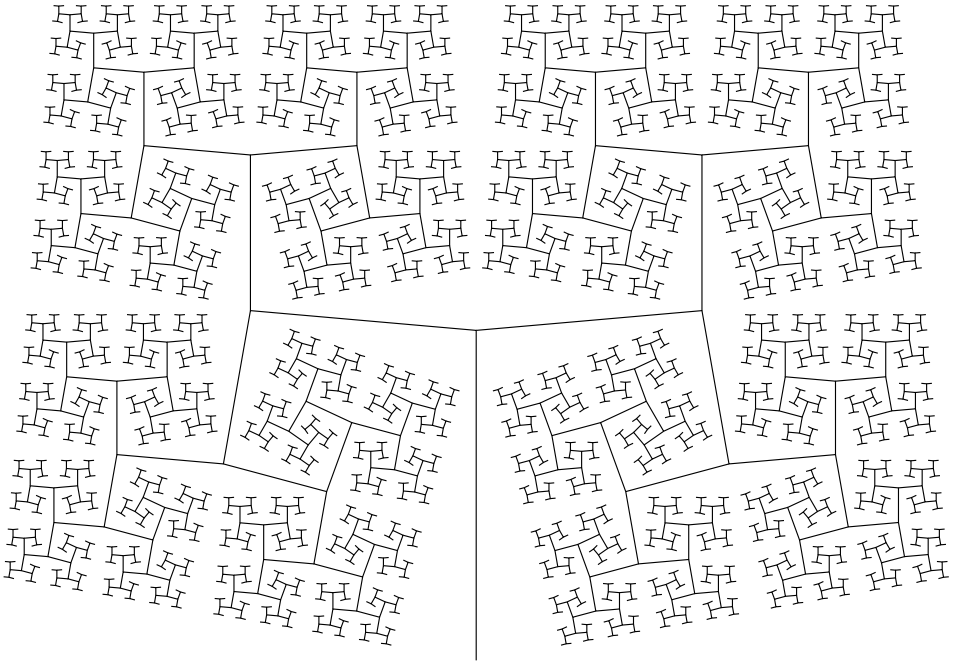


Abbildung 2.9.: Fraktale Zeichnung des parametrischen L-Systems aus Beispiel 4 (12. Iteration)

2.3.5. Map L-Systeme

Viele botanische Strukturen oder auch neuronale Netze haben eine komplexe Topologie die sich nur durch Graphen mit Zyklen beschreiben lassen und nicht durch Bäume. Mit den oben genannten L-Systemen unter Verwendung von Turtle-Kommandos ist dies allerdings nicht möglich, da diese lediglich zwei Relationen zwischen Objekten ermöglichen: Es handelt sich entweder um einen direkten Nachfolger oder um eine Verzweigung. Zyklen können demnach nicht erzeugt werden.

Map L-Systeme ermöglichen hingegen beliebige Relationen. Aufgrund der Komplexität und Vielfältigkeit des Themas sei der Leser auf [FP90], [Lin87] und [LR79] verwiesen.

3. Evolutionäre Erzeugungsalgorithmen für Neuronale Netze

In diesem Kapitel betrachten wir einige ausgesuchte Algorithmen zur evolutionären Optimierung von Topologie und Gewichtung neuronaler Netze und arbeiten ihre Vor- und Nachteile heraus. Dabei gehen wir ausschliesslich auf den Aspekt der Netzgenerierung ein. Details, die das zugrundeliegende Neuronenmodell betreffen, das Lernverfahren oder die genaue Durchführung der Klassifikation und Bewertung von Eingabemustern, interessieren uns hierbei nicht. Kurz: Wir möchten wissen, zum einen, wie gut die Algorithmen in der Lage sind, gerichtete Graphen mit Mehrfachkanten und Gewichtung zu erzeugen, zum anderen, welche Prinzipien hierbei von Vorteil sind.

Auf die Herausforderungen, die bei der Verwendung eines evolutionären Algorithmus zur gleichzeitigen Optimierung von Topologie und Gewichtung auftreten, gehen wir in Abschnitt 3.1 ein. In Abschnitt 3.2 folgt ein kurzer Überblick über die verschiedenen Kodierungsverfahren. Eine ausführliche Beschreibung der ausgesuchten Algorithmen findet im Anschluss daran, in Abschnitt 3.3 statt. Diese unterziehen wir in Abschnitt 3.4 einer genauen Analyse, wobei wir die besonderen Merkmale nochmals herauskristallisieren. Eine kurze Zusammenfassung in Abschnitt 3.5 bildet das Ende dieses Kapitels.

3.1. Herausforderungen

3.1.1. Wahl der geeigneten Kodierung

Eine der grössten Herausforderungen bei der evolutionären Optimierung von Neuronalen Netzen besteht darin, eine geeignete Wahl für die Kodierung der Genome zu treffen. Die Kodierung hat hierbei weitreichende Auswirkungen auf:

- die Qualität und Funktionsweise der genetischen Operatoren,
- die damit verknüpften Probleme, wie etwa das Permutationsproblem oder das Problem der inkompatiblen Repräsentation,

- die Skalierbarkeit des zu entwickelnden Netzes (z.B. direkte Kodierung vs. indirekte Kodierung),
- und auf die Effizienz des Verfahrens.

Abschnitt 3.2 verschafft einen Überblick über die verschiedene Arten der Kodierung. Abschnitt 3.3 erklärt im Detail einige ausgesuchte Kodierungen und Erzeugungsalgorithmen.

3.1.2. Das Permutationsproblem

Das *Permutationsproblem* [Rad90] tritt immer dann auf, wenn ein Suchalgorithmus der Repräsentation einer Lösung eine gewisse Ordnung aufzwängt, der Lösungsraum hingegen keiner solchen Ordnung unterliegt. Es wird somit jede mögliche Permutation ($n!$) einer Lösung auf denselben Punkt im Lösungsraum abgebildet. Dieses Problem wird in der Literatur oft auch als *Competing Conventions Problem* [LEM92] bezeichnet, oder das Abbildungsproblem von Struktur und Funktion[1] [WSB90] genannt. Bei Graphen entspricht dies der Isomorphie zweier oder mehrerer Graphen.

Im Kontext der Evolutionären Optimierung von Neuronalen Netzen tritt das Permutationsproblem genau dann auf, wenn unterschiedliche Genome auf ein identisches Netz, oder ein Netz mit gleicher Fitness, abbilden. Die Abbildung $Genom \rightarrow Netz$ bzw. $Genom \rightarrow Fitness$ stellt in diesem Fall also keine injektive Funktion dar.

Als eine der Folgen erhöht sich die Komplexität des Suchraums, den der Evolutionäre Algorithmus durchlaufen muss. Dies hat im allgemeinen negative Auswirkungen auf die Performanz des Algorithmus.

Ein weitaus grösseres Problem entsteht allerdings bei der Kreuzung zweier permutierter Genome. Da die Genome (in Teilen) äquivalent zueinander sind, kann die unsachgemässe Kreuzung zu einem Verlust an Vielfalt führen. Im folgendem Beispiel werden die Genome ABC und CBA durch 1-Punkt Crossover an der zweiten Stelle miteinander rekombiniert. Beide Genome repräsentieren hierbei dasselbe Netz, jedoch in permutierter Form. Den erzeugten Nachkommen (ABA und CBC) fehlt es entsprechend an Diversität.

```
Genom 1: A B C
Genom 2: C B A
Kind 1:  A B A
Kind 2:  C B C
```

Verhindert wird dieses Problem in einigen Algorithmen durch den vollständigen Verzicht auf die Crossover-Operation. Alternativ wird eine grosse Anfangspopulation zufällig erzeugter Genome einer bestimmten Komplexität (Länge) gewählt, um die Wahrscheinlichkeit, dass während der Evolution Diversität verloren geht,

[1] Structural/Functional Mapping Problem

entsprechend zu minimieren. Beide Methoden bekämpfen allerdings nur die Symptome dieses Problems, nicht die Ursache, und schaffen dabei neue Probleme[2]. In Abschnitt 3.3.5 werden wir ein Verfahren kennenlernen, dass eine grundsätzliche Lösung des Problems beschreibt.

Eine weiterführende wissenschaftliche Studie, die sich unter anderem mit der Signifikanz des Permutationsproblems beschäftigt, findet sich in [Haf10].

3.1.3. Das Problem der Inkompatiblen Repräsentation

Das Problem der inkompatiblen Repräsentation ist eng verknüpft mit der Problematik, die bei der Kreuzung von (teilweise) permutierten Genomen auftritt. Grundsätzlich besteht das Problem darin, dass durch Kreuzung von Genomen mit inkompatibler Repräsentation, Nachkommen entstehen können, die eine sehr viel niedrigere Fitness aufweissen als ihre Eltern.

Doch was bedeutet inkompatible Repräsentation überhaupt? Betrachten wir hierzu als Beispiel eine Liste bestehend aus ganzen Zahlen zwischen 0 und 15 und nehmen an, dass wir diese im Genom als Folge von Binärziffern kodieren.

```
A: [7, 15, 0] => 0111 1111 0000
B: [1, 0, 7]  => 0001 0000 0111
```

Dies entspricht einer Kodierung wie sie früher häufig in binärkodierten genetischen Algorithmen zur Anwendung kam. Betrachten wir nun was passiert, wenn wir durch lineares 1-Punkt Crossover einen Nachfolger bilden. Schneiden wir an Position vier (bzw. an einer durch vier teilbaren Position), so erhalten wir eine Kombination und Permutation aus Werten beider Listen:

```
A: 0111|1111 0000 => [7, 15, 0]
B: 0001|0000 0111 => [1, 0, 7]
X: 0111|0000 0111 => [7, 0, 7]
```

Schneiden wir hingegen an einer anderen Stelle, beispielsweise an Position sechs, so erhalten wir einen Wert, der zuvor nicht in den beiden Listen enthalten war (12):

```
A: 0111 11|11 0000 => [7, 15, 0]
B: 0001 00|00 0111 => [1, 0, 7]
X: 0111 11|00 0111 => [7, 12, 7]
```

Bei uniformen Crossover verhält es sich ähnlich. Das Ergebnis ist abhängig von der gewählten Blockgrösse. Tauschen wir beispielsweise immer Blöcke zu je vier Bits zwischen den Genomen hin und her, so bleibt die ursprüngliche Menge der Zahlen erhalten:

[2] Ein rein auf Mutation basierendes Verfahren hat eine langsamere Konvergenz. Eine komplexere Anfangspopulation erhöht den Suchraum, und erschwert das Finden von Lösungen geringer Komplexität.

```
A: 0111 1111 0000 => [7, 15, 0]
B: 0001 0000 0111 => [1, 0, 7]
X: 0111 0000 0000 => [7, 0, 0]
```

Allerdings macht uns das sehr anfällig für das Permutationsproblem. Wie dem obigen Beispiel zu entnehmen ist, geht dabei sehr schnell Diversität verloren.

Werden zusätzlich noch unterschiedliche Datentypen im Genom binär kodiert, beispielsweise Fliesskommazahlen oder Strings, die sich zudem in ihrer Kodierungslänge unterscheiden, so verschärft sich das Problem. Bei naiver Anwendung der genetischen Operatoren auf Bitebene führt dies dazu, dass möglicherweise eine neue Fliesskommazahl aus Bits einer Ganzzahl und den Bits eines Strings gebildet werden, obwohl deren Bedeutung komplett unterschiedlich (inkompatibel) ist.

Eine Repräsentation auf höherer Ebene, wie etwa bei dem Genetischen Programmieren von John R. Koza in Form von Listen oder Bäumen, erhält die innere Struktur der einzelnen Werte, indem die genetischen Operatoren, statt auf der Bitebene, auf der Ebene einzelner Elemente einer Liste oder eines Baumes operieren. Dennoch tritt auch hier die gleiche Problematik auf, wenn auch auf einer höheren Ebene. Dies liegt an der Nichtbeachtung der durch die Struktur der Liste oder des Baumes implizit gegebenen Ordnung, die bei der genetischen Operation, unabhängig davon ob es sich nun um eine Liste von Bits (Binärstring) oder eine Liste bestehend aus anderen Elementen, teilweise verloren geht.

Betrachten wir nun wie sich die inkompatible Repräsentation bei der Kreuzung von Graphen bemerkbar macht (Abbildung 3.1). Genom B (siehe unten) repräsentiert hierbei einen isomorphen Teilgraph des durch Genom A beschriebenen Graphen, verwendet jedoch eine unterschiedliche Nummerierung der Knoten. Kreuzen wir beide Genome durch lineares 1-Punkt Crossover an Position eins, so erhalten die in der Abbildung als Kind 1 und Kind 2 bezeichneten Graphen.

```
Genom A: (1,2) (2,3) (2,4)
Genom B: (3,1) (3,2)
Kind  1: (1,2) (3,2)
Kind  2: (3,1) (2,3) (2,4)
```

Die Struktur von Kind 1 besitzt hierbei keine Ähnlichkeiten zu beiden Elterngenomen (ausgenommen sind einzelne Kanten). Kind 2 hingegen lässt eine gewisse strukturelle Ähnlichkeit zu beiden Genomen vermuten. Allerdings sieht man, dass die Verbindung $3 \rightarrow 1$ aus Genom B an den Knoten 3 aus Genom A hinzugefügt wurde. Dabei wird ausser Acht gelassen, dass Verbindung $3 \rightarrow 1$ (Genom B) topologisch ähnlich zu $2 \rightarrow 3$ (Genom A) ist, d.h. durch Hinzufügen dieser Kante zu Genom A, sollte eigentlich keine neue Kante entstehen, sondern diese mit Kante $2 \rightarrow 3$ aus Genom A vereinigt werden. Folglich ist auch hier eine inkompatible Repräsentation festzustellen.

Das Problem der inkompatiblen Repräsentation tritt demnach immer dann auf, wenn es bei der Kreuzung nicht gelingt eine Zuordnung der einzelnen Teile (Gene)

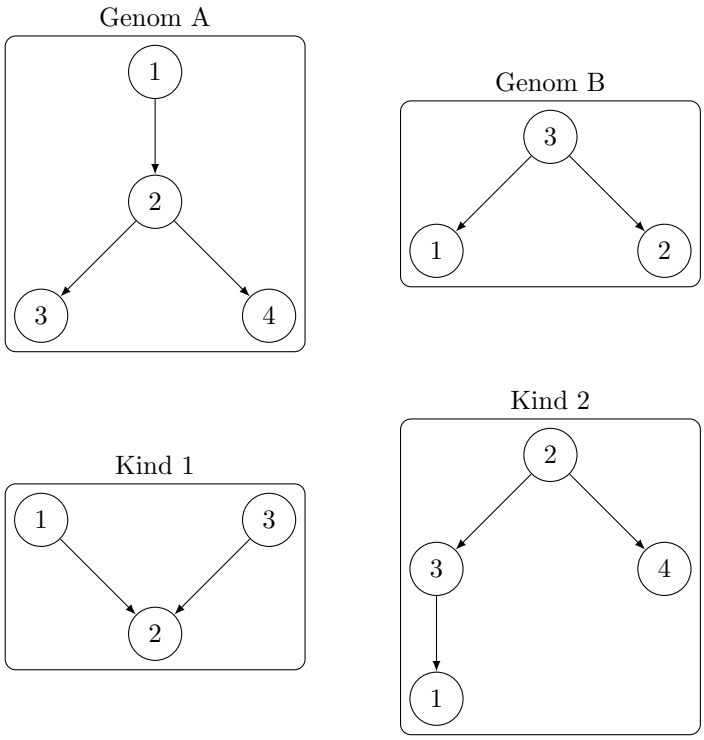


Abbildung 3.1.: Inkompatible Repräsentation bei der knotenorientierten Kodierung von Graphen und punktweiser Kreuzung.

des einen Genoms zu denen des anderen Genoms herzustellen (*Gene Alignment*). Im Beispiel oben könnte diese Zuordnung etwa durch eine topologische Analyse der Graphen erfolgen[3]. Wagt man einen Blick auf die Prozesse, die bei der Rekombination von DNA stattfinden, so erkennt man, dass hierbei bereits in der Kodierung zueinander passende Gene derart markiert sind, sodass sich diese leicht finden lassen, ohne dass dafür eine komplizierte Berechnung notwenig wird[4].

Bei der Wahl der Kodierung ist daher in besonderer Weise darauf zu achten, dass diese eine geeignete Zuordnung der Gene ermöglicht und somit eine effiziente Kreuzung topologisch unterschiedlicher Genome erlaubt. Stanley [SM02] beschreibt dies als *Gene Alignment of Disparate Topologies* und präsentiert eine Lösung dieses Problems in Form eines direkt kodierenden Verfahren (siehe Abschnitt 3.3.5). Das Fehlen dieser Zuordnung erkennt man häufig an grossen Abweichungen zwischen den Fitnesswerten der Eltern zu ihren Nachkommen.

[3]Dies ist jedoch langsam.

[4]Die Natur kann allerdings auf quantenähnliche Computer zurückgreifen, sodass selbst eine Berechnung der Isomorphie von Graphen wohl kein Problem darstellen würde.

3.1.4. Schutz neuer Innovationen

Bei der gleichzeitigen Evolution von Topologie und Gewichtung Neuronaler Netze kann die Situation auftreten, dass sich Änderungen am Genom erst nach mehreren Generationen positiv auf die Fitness auswirken. Dies ist insbesondere dann der Fall, wenn Änderungen am Genom möglichst klein gehalten werden (siehe inkompatible Repräsentation). Es muss daher Vorsorge getroffen werden, dass diesen Genomen genügend Zeit gegeben wird sich zu entwicklen, ohne dass diese durch andere Genome, die beispielsweise lokale Maxima repräsentieren, dominiert und vorzeitig eleminiert werden.

Hierfür bietet sich die Unterteilung der Gesamtpopulation in Nischen bzw. Unterpopulationen an, wobei fortan nur noch Genome der gleichen Nische, d.h. Genome von ähnlicher Struktur, gegeneinander konkurrieren. Die Schwierigkeit liegt allerdings darin, eine geeignete Partitionierung zu finden, mit der die Genome auf die Nischen verteilt werden. Eine Möglichkeit hierfür, die Stanley in [SM02] beschreibt, ist die Unterteilung der Genome in Nischen anhand ihrer genetischen Ähnlichkeit, und damit anhand ihrer topologischen Ähnlichkeit.

3.1.5. Beherrschung der Evolution komplexer Netze

Philipp Koehn fasst die Problematik wie folgt zusammen: 'It is a largely observed fact that GANN[5] systems have difficulties with large networks, see for instance [Kit90], [WDD91] and [Whi93]. While functions like XOR have been successfully applied to nearly all approaches, reports of complex tasks with many input and output nodes are rather rare' ([Koe94], Seite 18).

Ein entscheidender Aspekt bei der Beherrschung der Evolution komplexer Netze ist die durch den Algorithmus bestimmte Kodierung. Die direkte Kodierung ermöglicht beispielsweise keine effiziente Kodierung wiederkehrender Strukturen und ist somit nicht geeignet grosse Netze zu erzeugen[6]. Die indirekte Kodierung eines Netzes hingegen erlaubt es, wiederkehrende Muster kompakt zu repräsentieren. Bietet die Kodierung allerdings zuviele Freiheitsgrade, so droht dadurch eine Explosion des Suchraumes. Desweiteren sind die genetischen Operatoren von entscheidender Bedeutung für die Beherrschung komplexer Netze (siehe inkompatible Repräsentation). Allerdings zeigt es sich, dass sich topologieerhaltende genetische Operatoren wesentlich besser für direktkodierende Verfahren entwerfen lassen als dies für indirekte Verfahren der Fall ist.

Einen Ausweg aus diesem Dilemma wäre eine Kodierung auf Abstraktionsebenen von unterschiedlicher Komplexität. Dies könnte beispielsweise erreicht werden, indem ein Netz von geringer Komplexität direkt kodiert wird, unter der Annahme,

[5] GANN: Genetic Algorithm Neural Networks. Gemeint ist die Evolutionäre Optimierung von Neuronalen Netzen.

[6] Wir machen hierbei die Annahme, dass sich grosse und komplexe Netze dadurch auszeichnen, dass sie wiederkehrende Strukturen aufweisen.

dass hierfür geeignete genetische Verknüpfungsoperatoren zur Verfügung stehen[7]. Dieses Netz erster Ordnung würde dafür verwendet werden, ein komplexeres Netz zweiter Ordnung zu erzeugen. Ähnliches könnte erreicht werden, indem das emergente Zusammenspiel einiger weniger Regeln genutzt wird um komplexe Strukturen zu erzeugen[8].

3.2. Verschiedene Arten der Kodierung

3.2.1. Direkte Kodierung

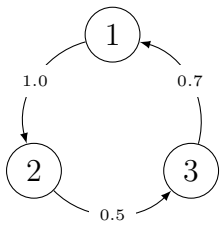
Bei der direkten Kodierung wird die Struktur und Gewichtung direkt im Genom kodiert, ohne dass dabei eine weitere (nicht-triviale) Interpretation auf den im Genom gespeicherten Information stattfindet.

3.2.1.1. Verbindungsbasierte Kodierung

Bei der verbindungsbasierten Kodierung wird ausschliesslich die Verbindungsinformation des Netzes im Genom kodiert. Dies kann dabei entweder in Form der zugehörigen Adjazenzmatrix[9] oder in Form einer Adjazenzliste[10] erfolgen. In der Regel ist bei dieser Kodierung die Anzahl der Neuronen eine fest vorgegebene Grösse, die im Laufe einer Evolution nicht verändert werden kann. Die verbindungsbasierte Kodierung findet erstmals in Innervator [MTH89] und GENITOR [Whi89] Anwendung. Sie stellt die einfachste Form der direkten Kodierung dar.

Folgendes Beispiel zeigt die verbindungsbasierte Kodierung eines Kreisgraphen der Länge 3 in Form einer gewichteten Adjazenzliste:

```
(1, 2, 1.0)
(2, 3, 0.5)
(3, 1, 0.7)
```



Verbindungsbasierte Kodierung

[7] Eine Möglichkeit ist die Verwendung von NEAT (Abschnitt 3.3.5)

[8] Siehe hierzu Abschnitte 3.3.4 und 3.3.8.

[9] Bei ungerichteten Graphen genügt die Kodierung der oberen Hälfte. Bei gewichteten Graphen muss statt einer binären Adjazenzmatrix eine entsprechende Matrix mit reelen, die Gewichtung beschreibenden, Werten verwendet werden. Ungewichtete Graphen mit Mehrfachkanten können durch eine ganzzahlige Adjazenzmatrix repräsentiert werden. Die Repräsentation von gewichtete Graphen mit Mehfachkanten ist auf diese Weise jedoch nicht möglich.

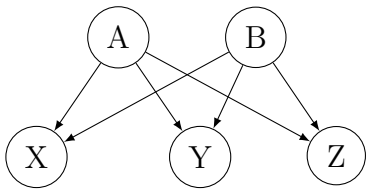
[10] Dies ist vorallem bei spärlich besetzen Adjazenzmatrizen von Vorteil

3.2.1.2. Knotenbasierte Kodierung

Bei der knotenbasierten Kodierung [KR91; Whi93; SJW91; SJW93] werden zudem die Knoten des Netzes explizit mitkodiert. Dies hat den Vorteil, dass die Anzahl der Knoten nicht mehr wie bei der verbindungsbasierten Kodierung fest vorgegeben werden muss, sondern sich im Laufe der evolutionären Entwicklung an die Anforderungen der Problemstellung anpassen kann.

Das folgende Beispiel zeigt die Kodierung eines 2-3 Jeffress Netzes:

```
A: [X, Y, Z]
B: [X, Y, Z]
X: []
Y: []
Z: []
```



Knotenbasierte Kodierung

3.2.1.3. Pfadbasierte Kodierung

Bei der pfadbasierten Kodierung werden anstelle einzelner Verbindungen ganze Pfade durch das Netz kodiert.

3.2.1.4. Schichtenbasierte Kodierung

Bei der schichtenbasierten Kodierung werden die Parameter der Schichten eines mehrlagigen neuronalen Netzes kodiert, wie etwa die Anzahl der Schichten des Netzes, die Anzahl der Neuronen einer bestimmten Schicht oder der Verbindungsgrad mit der nachfolgenden Schicht.

3.2.2. Indirekte Kodierung

Im Unterschied zur direkten Kodierung erfolgt bei der indirekten Kodierung eine Interpretation der im Genom gespeicherten Information. Resultat der Interpretation (oder Abbildung) ist ein konkretes Netz. Im Idealfall ermöglicht dies eine kompaktere Beschreibung eines Netzes mit der auch wiederkehrende Muster effizient kodiert werden können. Eine Vielzahl unterschiedlicher indirekter Kodierungsformen hat sich im Laufe der Jahre entwickelt. Einige grundlegende Verfahren wollen wir im Folgenden kurz beschreiben.

3.2.2.1. Grammar Encoding

Kitano [Kit90] beschreibt als einer der Ersten ein indirektes Kodierungsverfahren zur evolutionären Entwicklung der Topologie von neuronalen Netzen. Für die Entwicklung der binären Verbindungsmatrix des Netzes verwendet er hierbei ein kontextfreies L-System. Seine Experimente zeigen, dass sein *Grammar Encoding* genanntes Verfahren gegenüber der direkten Kodierung[11] sowohl eine signifikant höhere Skalierbarkeit[12] als auch eine wesentlich höhere Konvergenz[13] aufweist.

3.2.2.2. Koza

Wenig später wendet John R. Koza [KR91] das von ihm ursprünglich zur Funktionsoptimierung entwickelte *Genetische Programmieren* auf die evolutionäre Optimierung neuronaler Netze an. Sein Verfahren kann dabei erstmals sowohl die Topologie eines Netzes, als auch dessen Gewichtung optimieren. Die Topologie der mit seinem Verfahren erzeugten Netze ist allerdings auf die eines Baumes beschränkt. Der Grund hierfür liegt in der direkten Abbildung von der Repräsentation des Genoms, welches in Form von LISP S-Expressions vorliegt, auf die zu entwickelnde Netzstruktur. Siehe dazu Abbildung 3.4.

Neben dieser Einschränkung, die sich darin bemerkbar macht, dass die erzeugten Netze lediglich einen Ausgang[14] aufweisen können, besteht ein weiterer Nachteil seines Verfahrens darin, dass damit Redundanzen nicht (effizient) kodiert werden können.

Allerdings erkennt er als einer der Ersten die Signifikanz die von der Kodierung ausgeht, '... because it severly limits the window by which the system observes its world' ([KR91]). Er spielt damit auf die Probleme der inkompatiblen Repräsentation an, die sich insbesondere bei einem binären Genetischen Algorithmus bemerkbar machen[15].

3.2.2.3. Zellulare Kodierung

Beeinflusst durch das Genetische Programmieren von Koza entwickelt Gruau [Gru92] seine sehr einflussreiche *Zellulare Kodierung*. Diese spezielle Form der Kodierung ermöglicht erstmals, durch den Einsatz einer Graphgrammatik, sowohl die Gewichtung als auch die Topologie *beliebiger* Netze evolutionär zu entwickeln *und* dabei wiederkehrende Muster effizient zu kodieren. Eine genaue Beschreibung des Verfahrens findet sich in Abschnitt 3.3.2.

[11] Er verwendet hierfür einen binären genetischen Algorithmus.
[12] Skalierbarkeit in Bezug auf die Anzahl der Knoten des Netzes.
[13] gemessen an der Anzahl benötigter Generationen.
[14] Der Ausgang wird hierbei durch die Wurzel des Baumes repräsentiert.
[15] Siehe Abschnitt 3.1.3.

```
(P (W (+ 1.1 0.74) (P (W 1.66 D0) (W -1.38 D1)))
   (W (* 1.2 1.58) (P (W 1.19 D1) (W -0.98 D0))))
```

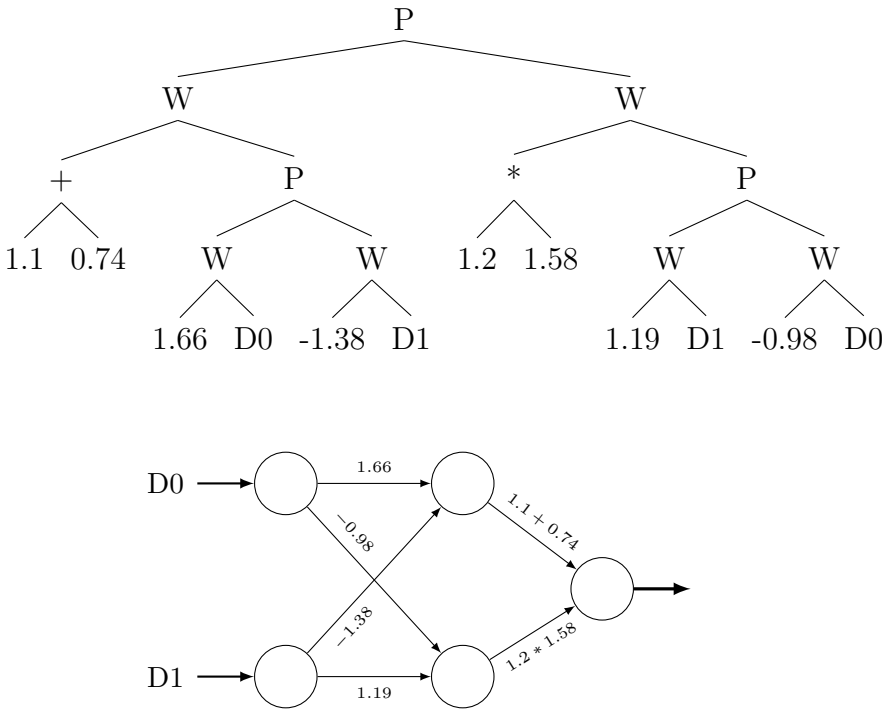


Abbildung 3.4.: Kodierung von Koza. Oben: LISP S-Expression. Mitte: Repräsentation als Baum. Unten: Erzeugtes Neuronales Netz. D0 und D1 sind Eingänge des Netzes. P repräsentiert ein Neuron (processing node), W dient der Einstellung der Kantengewichte.

3.2.2.4. Kodierung durch L-System

Während Kitano [Kit90] ein L-System zur Entwicklung der quadratischen Verbindungsmatrix eines Netzes verwendet, gelingt Boers [Boe95] die Verknüpfung eines L-Systems mit einer Graphgrammatik zur evolutionären Optimierung neuronaler Netze. Sein kontextsensitives L-System mit Klammerung erlaubt hierbei die Beschreibung der Topologie eines Netzes, die Klammerung ermöglich indes die effiziente (vollständige) Verdrahtung von Teilnetzen. Allerdings berücksichtig sein Verfahen noch nicht die Gewichtung der Kanten.

Pollack und Hornby [H+01] hingegen ermöglichen auch die Einstellung der Gewichtung, indem sie ein parametrisches L-System zur Kodierung der Netze verwenden. Eine genaue Beschreibung des letzteren Verfahrens findet sich in Abschnitt 3.3.3.

3.2.2.5. Kodierung durch Zellulare Automaten

Isto Aho [Aho+97] verwendet ein 2-dimensionales L-System, welches starke Ähnlichkeiten zu einem 2-dimensionalen Zellularen Automaten mit Moorscher Nachbarschaft aufweist. Anhand einfacher, emergenter Regeln bestimmt er damit die Positionierung und Parameterisierung von Neuronen in einer Zellmatrix. Die Bildung der Verbindungen erfolgt separat und ist durch die Positionierung der Neuronen in der Zellmatrix und durch deren Geruchsorientierung bestimmt. Eine nähere Beschreibung des Verfahrens findet sich in Abschnitt 3.3.8.

Gers et al. beschreibt in [GGK98] ein auf Zellulare Automaten basierendes Modell von gepulsten neuronalen Netzen. Anders als Isto Aho verwendet Gers den Zellularen Automat nicht nur zur Netzgenerierung, sondern dieser dient auch gleichzeitig der Simulation der gepulsten neuronalen Netze. Dies macht es möglich, das Netz während der Simulation noch zu verändern. Das verwendete Modell ist hierbei einfach genug, um dieses effizient in Hardware zu realisieren.

3.2.2.6. Hierarchische Kodierung

Schmidt entwickelt in [Sch09] eine hierarchische, modulare Kodierung. In seinem Verfahren beschreiben Moduldefinitionen die Struktur und Parameterisierung einzelner Teilnetze. Module können sich hierbei wechselseitig instanzieren. Eine ausführliche Beschreibung des Verfahrens findet in Abschnitt 3.3.7 statt.

3.2.2.7. Netzwerkgesteuerte Kodierung

Lobo [LV10] verwendet ein Genregulationsnetzwerk in Form eines einfachen Boolschen Netzwerkes und steuert damit die Anwendung von Operationen einer Graphgrammatik, ähnlich wie Gruau das mit seiner Zellularen Kodierung macht, allerdings mit dem Unterschied dass Gruau die Operationen in Bäumen kodiert. Näheres hierzu siehe Abschnitt 3.3.4.

Stanley [Sta07] verwendet ein Compositional Pattern Producing Network, d.h. eine spezielle Form eines künstlichen neuronalen Netzes. Damit erreicht er die Komposition und Ausprägung 4-dimensionaler geometrischer Muster, deren Bildwert er als Verbindungsgewichtung zwischen je zwei Neuronen an Position (x_i, y_i) und (x_j, y_j) deutet. An die Eingänge des Netzes legt er hierzu das Koordinatenpaar der zu bestimmenden Neuronen an. Näheres hierzu siehe Abschnitt 3.3.6.

3.3. Untersuchte Erzeugungsalgorithmen

In diesem Abschnitt untersuchen wir einige existierende Algorithmen, die zur Erzeugung von neuronalen Netzen eingesetzt werden.

3.3.1. Allgemeiner Aufbau

Die in diesem Kapitel beschriebenen Erzeugungsalgorithmen für neuronale Netze basieren alle auf dem gleichen Grundprinzip eines evolutionären Algorithmus. Den typischen Ablauf zeigt Abbildung 2.5. Der wesentliche Unterschied zwischen den einzelnen Erzeugungsalgorithmen liegt in der verwendeten Kodierung und in den daraus resultierenden Unterschieden zwischen den genetischen Operatoren. Allen Verfahren ist gemeinsam, dass die Genome eine Netzbeschreibung enthalten, die durch die Fitnessfunktion entwickelt und bewertet wird. Im Falle der evolutionären Entwicklung von neuronalen Netzen gliedert sich die Fitnessfunktion (siehe Inlet in Abbildung 2.5 sowie Abbildung 3.5) in zwei separate Teilabschnitte:

1. Zunächst wird aus der Netzbeschreibung des Genoms unter Anwendung des Erzeugungsalgorithmus ein neuronales Netz gewonnen (Genom → Netz).
2. Anschliessend wird das resultierende Netz in geeigneter Weise bewertet. Dies geschieht meisst durch Simulation, wobei die beobachtete Fehlerrate bei der Klassifikation von Testmustern als Fitness gewertet wird (Netz → Fitnesswert).

Da der zweite Teil der Bewertungsfunktion abhängig von der zu lösenden Problemstellung ist, konzentrieren wir uns in diesem Kapitel ausschliesslich auf den ersten, die Netzerzeugung betreffenden Teil der Bewertungsfunktion.

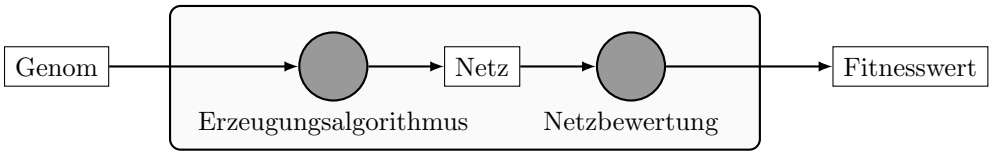


Abbildung 3.5.: Bewertungsfunktion eines evolutionären Algorithmus als Abbildung des Genoms auf seinen Fitnesswert. Bei der evolutionären Optimierung von neuronalen Netzen gliedert sich die Bewertungsfunktion in zwei Teile: Netzerzeugung und Netzbewertung.

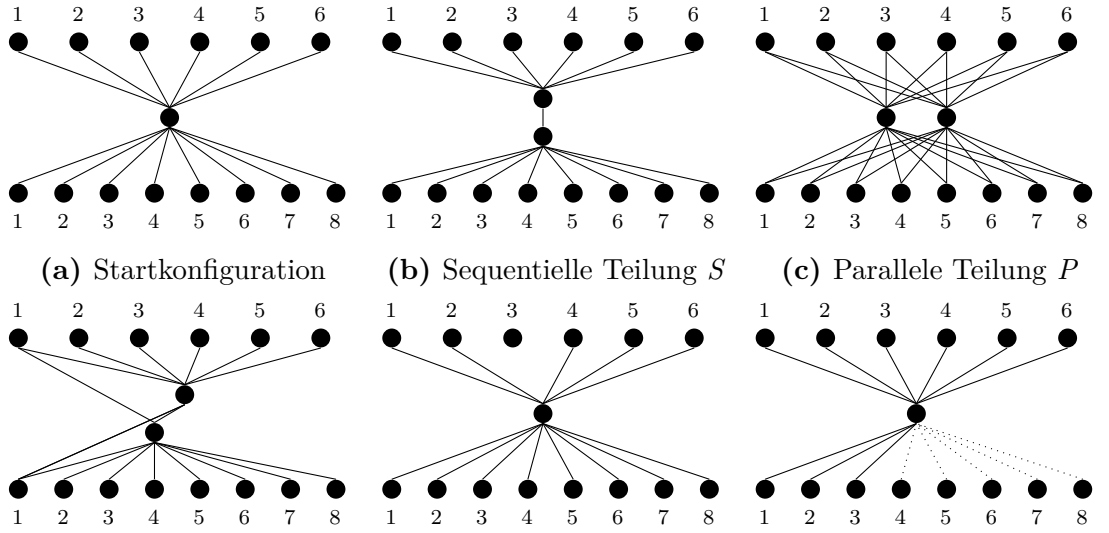
3.3.2. Zellulare Kodierung nach Gruau

Frédéric Gruau beschreibt unter dem Namen *Cellular Encoding* [Gru92; Gru+94; Gru95] ein Verfahren zur Erzeugung von neuronalen Netzen basierend auf dem biologischen Konzept der Zellteilung.

Sein Verfahren kennzeichnet sich dadurch, dass ausgehend von einer einzelnen *Keimzelle*, durch wiederholte Zellteilung und Transformation der Nachbarschaftsbeziehungen zwischen den Zellen, ein *Zellgraph* iterativ entwickelt wird. Dieser Zellgraph entspricht der späteren Topologie des neuronalen Netzes, allerdings erst nachdem alle Zellen ihre Entwicklung abgeschlossen haben und somit die Form eines Neurons annehmen.

Die Steuerung des Zellteilungsprozesses und der Transformationsoperationen übernimmt ein globales *Zellprogramm*, welches in Form eines Instruktionsbaumes vorliegt. Um ein differenziertes Zellwachstum zu ermöglichen, besitzt jede Zelle einen eigenen Befehlszeiger. Dieser bestimmt, welche Operation des globalen Zellprogramms die Zelle im nächsten Entwicklungsschritt auszuführen hat. Die Ausführung der Operationen aller Zellen geschieht hierbei simultan[16]. Führt eine Zelle eine Zellteilungsoperation durch, so entsteht eine neue Tochterzelle. Enthält der Instruktionsbaum an dieser Stelle eine Verzweigung, so folgen beide Zellen unterschiedliche Teilbäume[17]. Dies ermöglicht ein differenziertes Fortentwickeln der Zellen.

Desweiteren besitzt jede Zelle einen internen Zustand. Dieser bestimmt spätere Eigenschaften des ausgeprägten Neurons, wie etwa die Schwellwertfunktion oder die Gewichtung der eingehenden Synapsen, und kann durch entsprechende Operationen verändert werden.



(a) Startkonfiguration **(b)** Sequentielle Teilung S **(c)** Parallele Teilung P

(d) Teilung T, Var. von S **(e)** Unterdrückung $C\ 3$ **(f)** Änderung Gewichte $K\ 4$

Abbildung 3.6.: Zellulare Graphenoperationen nach F. Gruau

[16] genauso wie dies bei Zellularen Automaten der Fall ist, allerdings mit dem Unterschied, dass die Menge der Zellen in einem zellularen Automaten konstant ist.

[17] Der Instruktionsbaum ist ein binärer Baum. Die Mutterzelle folgt dem linken Teilbaum, die Tochterzelle dem rechten

Operationen der Graphgrammatik Abbildung 3.6 zeigt eine Auswahl der von Gruau verwendeten Operationen. Die als schwarze Punkte dargestellte Knoten symbolisieren die einzelne Zellen, die Kanten die Verbindungen zwischen diesen. Ausgangszustand für alle Operationen ist die in 3.6a gezeigte Startkonfiguration. Hierbei wird die entsprechende Operation auf die im Inneren dargestellte Zelle angewendet. Die Verbindungen zu den oberen Knoten symbolisieren die Eingangsverbindungen, die Verbindungen zu den unteren entsprechen den Ausgangsverbindungen. Die Anzahl der Ein- und Ausgangsverbindungen ist dabei variabel, anders als dies vielleicht die Abbildung zu suggerieren scheint. Operationen können hierbei:

- die Nachbarschaft bzw. Topologie der Zelle verändern (Abbildungen 3.6b bis 3.6e)
- die Teilung der Zelle bewirken (Abbildungen 3.6b bis 3.6d)
- die Gewichtungen verändern (Abbildung 3.6f)
- oder den Befehlsfluss steuern.

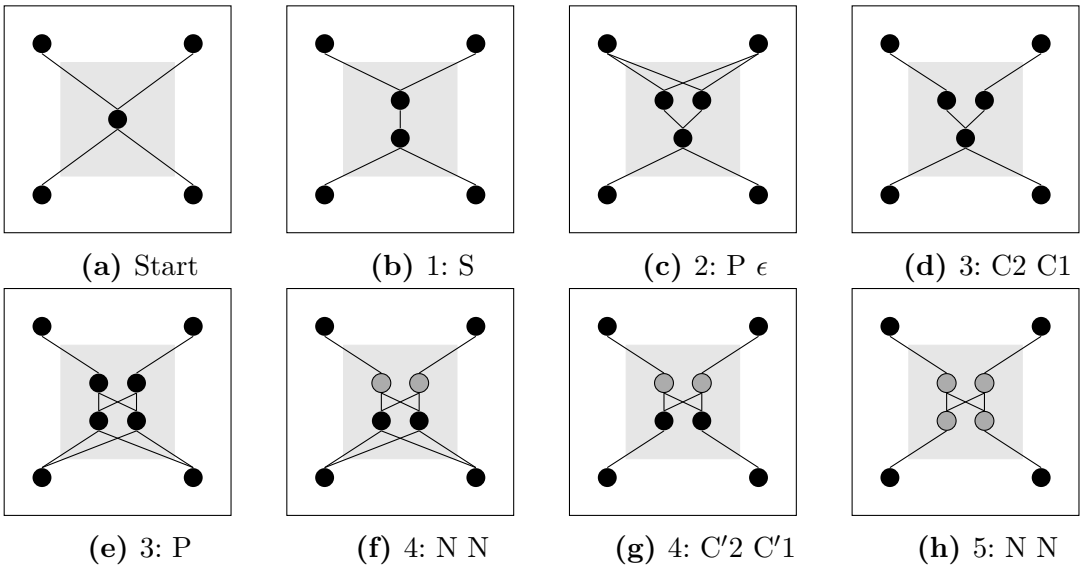
Gruau definiert weitere topologische Transformationen die hier allerdings nicht dargestellt sind. Dabei ist die Gemeinsamkeit aller Operationen seines Verfahrens, dass sie Produktionen einer *knotenorientierten* Graphgrammatik darstellen, d.h. sie operieren aus der Sicht eines einzelnen Knotens und verändern dessen Ein- und Ausgangsverbindungen bzw. fügen einen neuen Knoten zum Graphen hinzu. Da der Ein- und Ausgangsgrad eines Knotens variabel ist, lassen sich somit unendlich viele verschiedene Transformationen definieren. Dies ist einer der Hauptkritikpunkte seines Verfahrens. Weiterhin ist jede Transformation abhängig von seiner Umgebung. Dies macht es schwierig die genaue Auswirkung einer oder mehrerer Operation auf den Graphen vorherzusehen. Andererseits lassen sich sehr mächtige Operationen realisieren, die bei *kantenorientierten* Graphgrammatiken nur durch viele Einzeloperationen realisiert werden können.

Zellteilung Bewirkt eine Operation die Teilung der Zelle, so verfolgen beide Zellen fortan unterschiedliche Teilbäume des Befehlsstromes (Abbildung 3.8), sofern dieser an der aktuellen Position eine Verzweigung enthält. Dies ermöglich ein differenziertes Wachstum der Zellen und die Ausprägung unterschiedlicher Eigenschaften.

Beispiel Die Entwicklung eines 2-2 Jeffress Moduls ist in Abbildung 3.7 dargestellt. Der zugehörige Instruktionsbaum ist in Abbildung 3.8 gegeben. Die Zahlen im Untertitel der Bilder kennzeichnen den jeweiligen Entwicklungsschritt.

- Ausgegend vom Startzustand (Abbildung 3.7a) mit zwei Eingängen und zwei Ausgängen und der Keimzelle in der Mitte wird zuerst eine sequentielle Teilung S durchgeführt (3.7b). Dies führt zur Differenzierung beider Zellen, wobei die ursprüngliche Zelle den linken Teilbaum des Instruktionsbaumes weiter verfolgt, die neu erzeugte Zelle den rechten.

- Im zweiten Entwicklungsschritt (3.7c) führt die linke Zelle eine parallele Teilung P durch, während die rechte Zelle wartet (ϵ). Dieser Wartevorgang ist wichtig, da ansonsten die nächste parallele Teilung gleichzeitig erfolgen würde und somit die Eingangsnachbarschaft der rechten Zelle nicht vererbt werden könnte.
- Der dritte Schritt gliedert sich in Abbildungen 3.7d und 3.7e. In 3.7d löscht die linke Zelle den rechten Eingangslink ($C2$), während die rechte Zelle den linken löscht ($C1$). Zeitgleich führt die untere Zelle eine parallele Teilung P durch (3.7e).
- Im vierten Schritt (Abbildungen 3.7f und 3.7g) entwickeln sich die zwei oberen Zellen zu Neuronen (N), während die zwei unteren Zellen jeweils eine ihrer Ausgansverbindungen trennen ($C'2$ und $C'1$).
- Im letzten Schritt (3.7h) ensteht durch Ausführen der N Operation aus den beiden unteren Zellen ebenfalls Neuronen.



(a) Start **(b)** 1: S **(c)** 2: P ϵ **(d)** 3: C2 C1

(e) 3: P **(f)** 4: N N **(g)** 4: C′2 C′1 **(h)** 5: N N

Abbildung 3.7.: Entwicklung eines 2-2 Jeffress-Modul

Hierarchische Beschreibung Durch den Einsatz mehrerer Instruktionsbäume erreicht Gruau zudem die Kodierung von modularen Teilnetzen. Hierzu führt er Instruktionen ein, die bei Ausführung in einen anderen Instruktionsbaum wechseln. Um Schleifen zu vermeiden, und somit eine Terminierung des Verfahrens zu garantieren, müssen allerdings entsprechende Vorkehrungen getroffen werden. Dies kann entweder bei der Erzeugung der Instruktionsbäume geschehen oder während der Ausführung erfolgen.

In [Gru94] zeigt er, dass für ein einfaches Steuerungsproblem eines sechs-beinigen Roboters, unter Verwendung von mehreren Instruktionsbäumen, eine Lösung mehr

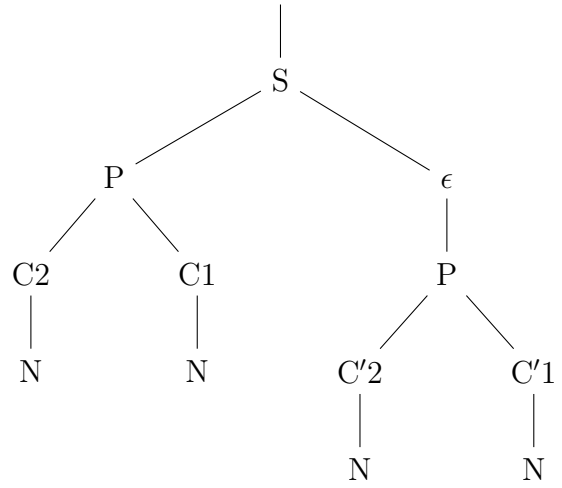


Abbildung 3.8.: Instruktionsbaum zur Entwicklung des Graphen aus Abbildung 3.7

als sechsmal so schnell gefunden wird als mit nur einem Instruktionsbaum. Darüber hinaus ist die Anzahl der benötigten inneren Neuronen des so gefundenen Netzes mehr als viermal so gering.

Evolutionäre Optimierung Gruau verwendet das von John R. Koza [KR91] eingeführte *Genetische Programmieren*. Hierbei bleibt die Repräsentation seines Genoms in Form eines oder mehrerer Instruktionsbäume erhalten, und die genetischen Operatoren arbeiten direkt auf dieser Repräsentation. Der Crossover-Operator tauscht dabei ganze Teilebäume zwischen den beiden Genomen aus, während der Mutations-Operator auf den einzelnen Knoten des Baumes Veränderungen bewirkt.

Bewertung

- Eine hierarchische Modularisierung wird ermöglicht, indem mehrere Instruktionsbäume verwendet werden. Dies führt, wie Gruau zeigt, zu einer kompakteren Kodierung, schnelleren Konvergenz und effizienteren Lösung.
- Die einzelnen graphtransformierenden Instruktionen arbeiten auf den Knoten des Graphen. Knoten haben dabei einen variierenden Eingangs- und Ausgangsgrad. Dies erlaubt eine beliebige Selektion oder Modifikation der für die Zelle (bzw. Neuron) benötigten Verbindungen. Verbindungen müssen also nicht erst explizit erzeugt werden, wie dies z.B. bei dem kantenorientierten Verfahren aus Abschnitt 3.3.3 der Fall ist.
- Die Verwendung einer *knotenorientierten* Graphgrammatik führt auf der anderen Seite zu der Problematik, dass sich grundsätzlich unendlich viele mögliche Operationen zur Transformation des Graphen definieren lassen. Welche Operationen notwendig oder besonders geeignet sind um jeden beliebigen Graphen

erzeugen zu können ist nicht leicht ersichtlich. Eine minimale Konfiguration muss hierbei Operationen enthalten die die Ein- und Ausgänge eines Knotens permutieren und somit durch wiederholte Anwendung jede beliebige Transformation realisieren können.

- Hinzu kommt, dass die Auswirkung einer Graph-Operation auf die Topolgie des Graphen abhängig von der Umgebung des Knotens ist. Dies erschwert die Entwicklung einzelner wiederverwendbarer Module.
- Die von Gruau verwendeten Operationen enthalten zudem Parameter, die zum einen beschreiben welche Verbindungung getrennt oder neu gewichtet werden sollen, zum anderen den Wert der Gewichtung angeben. Hierbei ist insbesondere die Wahl einer Verbindung problematisch, da diese eventuell gar nicht existiert und die Operation somit ungültig oder aber ohne Auswirkung bleibt.
- Während das Verfahren bei der Rekombination (Crossover) zweier Genome eine gute strukturelle Durchmischung durch den Austausch von Teilbäumen erreicht, müssen Parameterwerte (Kantengewichtung als auch die Topologie betreffende Parameter) mühsam durch Mutation adaptiert werden. Dies hat negative Auswirkungen auf die Konvergenzeigenschaften des evolutionären Algorithmus.
- Die manuelle Kodierung von Graphen mit diesem Verfahren ist bereits bei Graphen von geringer Komplexität nicht trivial und erfordert einiges an Übung.

3.3.3. L-System-gesteuerte Graphgrammatik

Hornby und Pollack [H+01] verwenden ein *Parametrisches L-System*[18] um damit eine Instruktionssequenz zu generieren, die durch anschliessende Interpretation anhand einer gegebenen Graphgrammatik einen Graphen ausbildet. Im Unterschied zu Gruau (Abschnitt 3.3.2) verwenden sie eine *kantenbasierte* Graphgrammatik, d.h. die Graphtransformationen operieren auf den Kanten. Dies ermöglicht die Definition von Modulen, die unabhängig von ihrer Umgebung bei ihrer Ausprägung immer die gleiche Topologie entstehen lassen[19].

Ausgangspunkt des Algorithmus ist eine einzelne virtuelle[20] Kante mit einem einzelnen Knoten, der Start- und Zielknoten der Kante zugleich ist (siehe Abbildung 3.9a). Durch sequentielles Abarbeiten der durch das L-System generierten Graphenoperationen entwickelt sich der Graph schrittweise. Zu jeder Zeit existiert dabei immer genau eine aktive Kante, auf der die aktuelle Operation angewendet wird. Diese

[18] siehe Abschnitt 2.3.4

[19] Dies liegt daran, dass die Umgebung genau durch den Quell- und Zielknoten der Kante gegeben ist. Abgesehen von einigen zustandsändernden Operationen, die es erlauben die Modulgrenze zu verlassen, wie z.B. `Parent`.

[20] Aus einer virtuellen Kante kann durch Veränderung der Gewichtung eine real existierende Kante erzeugt werden.

aktive Kante ist definiert durch einen Startknoten, einen Zielknoten, sowie dem Index der Kante in der Liste der eingehenden Verbindungen des Zielknotens. Letztere Information kann allerdings durch einige Operation[21] ungültig gemacht werden, was zum Entstehen einer sog. virtuellen Kante führt.

Operationen der Graphgrammatik

- `Duplicate(`w`)` erzeugt eine neue Kante mit den gleichen Quell- und Zielknoten wie die aktuelle Kante, jedoch mit Gewicht w. Die so erzeugte neue Kante wird danach zur aktuellen Kante.
- `Reverse` vertauscht Quell- und Zielknoten der aktuellen Kante.
- Die `Split(`x`)` Operation spaltet die aktuelle Kante $e = (v_s, v_t)$ durch Einfügen eines neuen Knotens v_n in zwei Kanten $e_1 = (v_s, v_n)$ und $e_2 = (v_n, v_t)$ auf. Dabei gilt für die Gewichtung der zwei neuen Kanten: $w(e_1) = w(e)$ sowie $w(e_2) = x$. Neue aktuelle Kante wird e_2.
- `Loop(`w`)` erzeugt eine Schleife im Zielknoten der aktuellen Kante mit Gewichtung w. Die Schleife wird neue aktuelle Kante.
- `Next(`n`)` wählt die n-te Nachbarkante (des gleichen Zielknotens) als neue aktive Kante aus.
- `Parent(`n`)` setzt den Quellknoten der aktuellen Kante auf die n-te Eingangskante des Quellknotens. Hierbei wird nur der Zustand der aktuellen Kante verändert, nicht die Topologie des Graphen!
- `Output(`w`)` erzeugt einen neuen Knoten v_n und verbindet diesen vom Zielknoten der aktuellen Kante aus mit Gewichtung w. Die aktuelle Kante bleibt erhalten.
- Durch `Merge(`n`)` wird der Quellknoten der aktuellen Kante mit dem Zielknoten verschmolzen, d.h. alle Eingangskanten (Ausgangskanten) des Quellknoten werden zu Eingangskanten (Ausgangskanten) des Zielknotens. Der Quellknoten, sowie die aktuelle Kante werden anschliessend entfernt. Neue aktuelle Kante wird die n-te Einganskante des ursprünglichen Quellknotens.
- `Increase-Weight(`w`)` erhöht das Gewicht der aktuellen Kante um w. Ist die aktuelle Kante eine virtuelle Kante, so wird eine Kante mit Gewichtung w erzeugt.
- `Decrease-Weight(`w`)` reduziert das Gewicht der aktuellen Kante um w. Ist die aktuelle Kante eine virtuelle Kante, so wird eine Kante mit Gewichtung $-w$ erzeugt.
- `Set-Node-Function(`f`)` setzt den Typ des Zielknotens der aktuellen Kante auf f.

[21] `Push` und `Pop` beispielsweise

- Hinzu kommen die Operationen `Push` und `Pop`, die den aktuellen Zustand auf einem Stapel sichern bzw. wiederherstellen. Diese sind auf die Klammern `[` und `]` des geklammerten L-Systems abgebildet.

Beispiel Da eine manuelle Erzeugung eines Graphen mit diesem Verfahren einiges an Übung erfordert, haben wir dieses Verfahren implementiert und mit einem evolutionären Algorithmus die Beschreibung für ein 2-3 Jeffress-Netz suchen lassen. Dabei haben wir auf die `Push` und `Pop` Funktionen verzichtet, da diese grafisch nicht leicht darzustellen sind. Bei einer Population von 1000 Individuen und einer Tournament-Grösse von 4 mit Neighbor Matching (siehe Abschnitt 5.3.5.1) als Fitness Funktion wurde nach 69 Iterationen[22] folgende Beschreibung gefunden (die mit `!` markierten Operationen sind ungültig bzw. haben keine Auswirkungen auf den Graphen):

```
Split(0.64) Split(0.80) Reverse Split(0.97) Merge(1) Reverse
Parent(0) Parent(!) Merge(!) Split(0.00) Merge(!) Reverse
Split(0.00) Split(0.00) Reverse Split(0.94) Split(0.97) Merge(1)
Reverse Parent(0) Merge(0) Next(0) Split(0.60) Reverse
```

Hierbei fiel auf, dass bei Verzicht auf die `Merge` Operation kein annähernder Graph gefunden werden konnte; alle Versuche stagnierten exakt bei 19% Ähnlichkeit zum Zielgraphen. Dies lässt vermuten, dass diese Operation eine wichtige Bedeutung für die Erzeugung beliebiger Graphen spielt.

Die einzelnen Entwicklungsschritte sind Abbildung 3.9 zu entnehmen. Dem Untertitel ist hierbei die Operation zu entnehmen, die zu dem jeweiligen Bild, ausgehend vom Vorgängerbild, geführt hat. Die aktive Kante ist fett-gedruckt und in rot, eine virtuelle aktive Kante gestrichelt dargestellt.

Parametrisches L-System Die Autoren des Verfahrens verwenden ein Parametrisches L-System. Hierbei sind die Regeln und Symbole des L-Systems zusätzlich mit Parametern bzw. Parameterausdrücken versehen. Bei der Ersetzung eines Symbols (Vorgänger) durch eine Regelproduktion (Nachfolger) werden diese entsprechend ausgewertet. Zudem ist an jede Regel eine Bedingung geknüpft, die wiederum in Bezug zu den Parametern stehen kann. Nur bei wahrem Ausgang der Bedingung wird eine Regel auch angewendet.

Zum besseren Verständnis der Funktionsweise der Parameterisierung sei an dieser Stelle Beispiel 5 gegeben. Regel $L(n)$ produziert hierbei durch einen rekursiven Aufruf eine Folge von n `Split` Operationen. Dies erzeugt einen Liniengraph der Länge n. Um einen geschlossenen Kreisgraph $K(n)$ zu realisieren, genügt es, die initiale virtuelle Kante durch Setzen der Gewichtung auszuprägen, gefolgt von n-maliger Kantenteilung durch die `Split` Operation.

[22] Nach mehrmaligem Ausprobieren

Beispiel 5 (Parametrisches L-System für Liniengraph/Kreisgraph der Länge 50)

$$\begin{aligned} L(n) &: \{n > 0\} \rightarrow Split\ L(n-1) \\ K(n) &: \{n > 0\} \rightarrow IncreaseWeight(0)\ L(n) \\ \omega_L &= L(50) \\ \omega_K &= K(50) \end{aligned}$$

Auf die gleiche Art und Weise lässt sich die Kantengewichtung parameterisieren. Beispiel 6 zeigt, wie sich ein Liniengraph der Länge n erzeugen lässt, dessen Kantengewichtung sich stetig halbiert.

Beispiel 6 (Param. L-System für Liniengraph mit abnehmender Gewichtung)

$$\begin{aligned} L(n,w) &: \{n > 0\} \rightarrow Split(w)\ L(n-1, \frac{w}{2}) \\ \omega &= L(50, 1.0) \end{aligned}$$

Genetische Operationen Für die Rekombination wird lineares 2-Punkt Crossover zwischen zwei zufällig ausgewählten Regeln (des L-Systems) beider Genome eingesetzt. Die Mutationsoperation arbeitet auf einer zufällig ausgewählten Regel und kann eine der folgenden Operationen ausführen:

`InsertSequence` Einfügen einer zufällig erzeugten Instruktionssequenz

`DeleteSequence` Entfernen eines Teiles der Instruktionssequenz

`ReplaceSymbol` Ändern einer Instruktion in eine andere, ohne die Parameter zu verändern

`ModifyParameter` Ändern eines Parameterwertes ohne die Instruktion zu verändern

`ModifyCondition` Modifizieren einer Regelbedingung

Bewertung

- Der Einsatz eines *Parametrischen L-Systems* erlaubt die Definition von komplexen hierarchischen Modulen, die darüber hinaus parameterisiert sein können.
- Durch Verwendung einer *kantenbasierten* Graphgrammatik ist es möglich, klar abgegrenzte Module zu entwickeln. Die manuelle Beschreibung eines Graphen ist allerdings mühsam und erfordert viel Übung.

- Während die Operationen bei Gruau (Abschnitt 3.3.2), ausgehend von einer Vollverdrahtung Verbindungen selektiv trennen, werden bei diesem Verfahren Verbindungen konstruktiv erzeugt. Neben der aktiven Kante, kann eine andere Kante durch `Next` und `Parent` explizit ausgewählt werden.
- Die Graphgrammatik benötigt relativ viele verschiedene Operationen, dabei ist nicht genau klar, welche unbedingt vorhanden sein müssen um beliebige Graphen erzeugen zu können. Unsere Experimente lassen indes vermuten, dass `Merge` eine notwendige Operation ist. Ob allerdings auf `Push` und `Pop` verzichtet werden kann ist nicht klar. Letztere Operationen erschweren die Genetischen Operationen, da sie eine Baumstruktur voraussetzen. Zudem erfordert die Interpretation dieser Operationen die zusätzliche Speicherung des Kontextes.
- Während einige Operationen der Graphgrammatik ganze Zahlen als Parameter erfordern, so benötigen andere reel-wertige. Dies erschwert die Implementierung der genetischen Mutations-Operation. Wird eine Graphenoperation durch Mutation in eine andere umgewandelt, so muss eventuell auch eine Konvertierung des Parameterwertes durchgeführt werden.
- Ein Parametrisches L-System, bei dem die Parameter die Form beliebiger mathematischer Ausdrückn annehmen können, ist zwar gut für die manuelle Beschreibung von komplexen funktionalen Zusammenhängen geeignet, erhöht hingegen auch den Suchraum den der evolutionäre Algorithmus durchlaufen muss. Ausserdem können so kleinste Änderungen an der Zusammensetzung eines mathematischen Terms, grosse Änderungen im Wertebereich hervorrufen (z.B. Umwandlung von x in $1/x$). In unseren Experimenten zeigte sich dies als Problem, insbesondere bei der Annäherung der Gewichtung, da ein einzelner Ausreisser, nach der Normalisierung der Kantengewichte, ein Grossteil der Gewichte gegen Null laufen liess.
- Die Erzeugung eines Graphen erfolgt nicht direkt wie etwa bei Gruau, sondern über den Umweg der Generierung einer Folge von Graphenoperation durch das L-System. Dies erschwert beispielsweise eine mögliche Kopplung der Iterationstiefe des L-Systems an bestimmte Grapheigenschaften, wie etwa die Anzahl der erzeugten Knoten. Die zellulare Kodierung von Gruau scheint hier von Vorteil zu sein.
- Die Iterationstiefe des L-Systems stellt ein zusätzlichen Parameter dar, dessen richtige Wahl grossen Einfluss auf den resultieren Graphen hat.
- Komplexe mathematische Parameter-Terme erscheinen biologisch nicht plausibel, ebensowenig wie die Verwendung eines unbeschränkten Wertebereichs.
- Der Austausch von Information bei der Rekombination (Crossover) zweier Genome findet ausschliesslich auf der Ebene der Operationen statt, Parameterwerte werden hierbei nicht berücksichtigt. Dies hat zur Folge, dass Parameterwerte mühsam durch Mutation verändert werden müssen. Da zudem einige topologische Operationen wie etwa `Next` und `Parent` parameterbehaftet sind, hat

dies auch Auswirkungen auf die Entwicklung der Topologie. Die Einstellung der Kantengewichtung geschieht ausschliesslich durch Mutation, d.h. durch punktuelle, zufällige Veränderung der Parameter. Eine genaue Annäherung der Kantengewichte an eine Zielvorgabe zeigte sich in unseren Experimenten als sehr schwierig.

3.3.4. Genregulationsnetzwerk-gesteuerte Graphgrammatik

Lobo und Vico [LV10] stellen ein biologisch motiviertes Verfahren zur gleichzeitigen Entwicklung von Form und Funktion mehrzelliger Organismen vor. Sie entwicklen dabei künstliche Lebensformen mit einer Gestalt (Form), die den Weg durch eine Kurvenstrecke mithilfe eines Steuerungsnetzwerkes (Funktion) erlernen sollen. Dieses Steuerungsnetzwerk besitzt dabei Ähnlichkeiten zu einem neuronalen Netz. Ihr zugrundeliegender Netzerzeugungsalgorithmus unterscheidet sich gegenüber der Zellularen Kodierung von Gruau[23] in folgenden Aspekten:

- Ein Genregulationsnetzwerk in Form eines Boolschen Netzwerkes steuert die Zellteilung und Spezialisierung. Gruau verwendet hierfür Instruktionsbäume.
- Für die Entwicklung des Zellgraphen verwenden sie eine sehr einfache *kantenorientierte* Graphgrammatik mit nur fünf (parameterlosen) Operationen. Gruau verwendet hingegen eine knotenorientierte Graphgrammatik mit parameterbehafteten Operationen.
- Anders als bei Gruau, entspricht ihr Zellgraph nicht dem späteren Netzwerk. Die Kanten ihres Zellgraphen symbolisieren Zellen, aus denen sich die Bestandteile des Netzwerkes, Neuronen und Synapsen, entwickeln. Die Knoten des Zellgraphen repräsentieren hierbei die Zellnachbarschaften.
- Kanten im Zellgraph (d.h. Zellen) mit gleichem Quell- und Zielknoten konkurrieren gegenseitig. Nur die Kante mit dem höchsten Wert des Typzählers überlebt.

Graphgrammatik Die folgenden fünf parameterlosen Operationen sind definiert:

1. `Split` teilt die Kante (Abbildung 3.10a).
2. `Duplicate` verdoppelt die Kante (Abbildung 3.10b).
3. `Reverse` dreht die Richtung der Kante um (Abbildung 3.10c).
4. `Resize` verändert den Gewichtswert (bzw. Länge) der Kante um 25% (Abbildung 3.10d).
5. `Type` erhöht den zellinternen Typzähler.

[23] Abschnitt 3.3.2

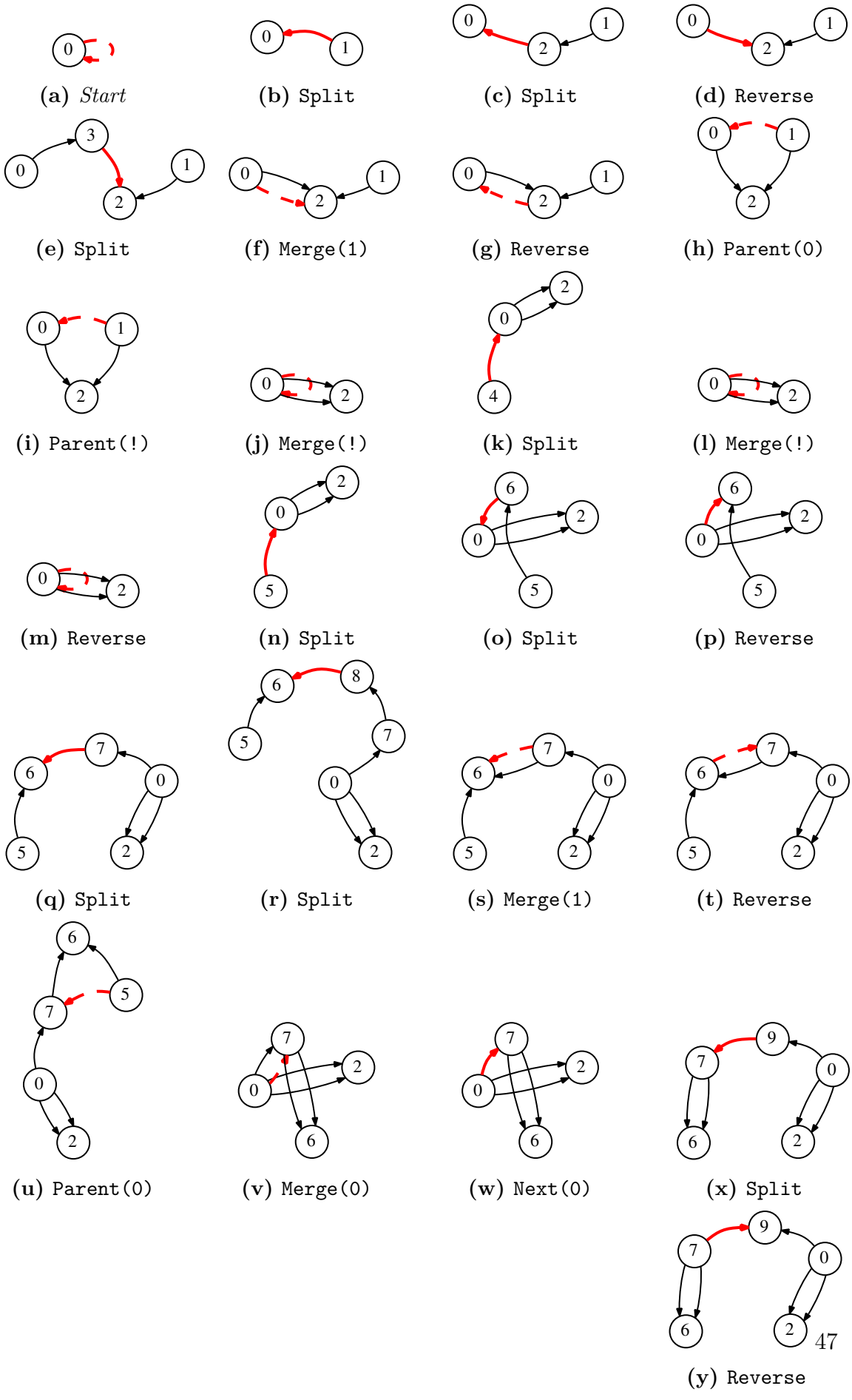
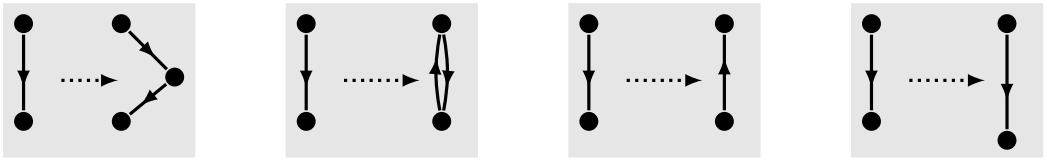


Abbildung 3.9.: Entwicklung eines 2-3 Jeffress Moduls

Anstelle der Parameterisierung tritt die wiederholte Anwendung der Operationen `Resize` und `Type`. Entsprechend dem Wert des Typzählers entwickelt sich im späteren Netzwerk entweder ein Neuron oder eine Synapse (bzw. ein anderes Element des Netzwerkes).



(a) Split **(b)** Duplicate **(c)** Reverse **(d)** Resize

Abbildung 3.10.: Vier Operationen der Graphgrammatik

Boolsches Netzwerk Das Genom kodiert die Wechselwirkungen der Knoten eines Boolschen Netzwerkes. Die Anzahl der Knoten ist dabei variabel und abhängig vom Genom. Ein Beispiel für ein Boolsches Netzwerk ist in Abbildung 3.11 gegeben. Hierbei ist einigen Knoten eine Graphenoperation zugeordnet: **T** steht für `Type`, **D** für `Duplicate`, **R** für `Resize`, **S** für `Split` und **W** für `Reverse`. Der Knoten **X** ist ein zusätzlicher Knoten dem keine spezielle Funktion zugeordnet ist und dient lediglich der Vergrösserung des Zustandsraumes.

Jeder Knoten eines Boolschen Netzwerkes ist entweder aktiv (= 1) oder inaktiv (= 0). Der Gesamtzustand eines Boolschen Netzwerkes mit n Knoten lässt sich somit mit n Bits kodieren. Zustandstransitionen des Boolschen Netzes sind anhand der Verbindungen zwischen den Knoten geregelt. Hierbei gibt es verstärkende Verbindungen (normale Pfeile in der Abbildung), als auch hemmende Verbindungen. Der neue Zustand eines Knotens ergibt sich aus der Summe aller eingehender verstärkender Verbindungen abzüglich der Summe aller eingehender unterdrückender Verbindungen. Ist das Ergebnis > 0 dann wird der Knoten aktiviert (Zustand = 1). Ansonsten ist der neue Zustand = 0. Wird diese Zustandstransition für jeden Knoten parallel durchgeführt, so ergibt sich ein neuer Zustand des gesamten Netzes, kodiert als n-Bit Wert.

Jede Zelle im Zellgraphen beherrbergt seinen eigenen Zustand des Boolschen Netzes. Dies entspricht dem zellinternen Befehlszeiger bei Gruau's Zellularer Kodierung. In jedem Iterationszyklus führt jede Zelle, parallel zu allen Zellen, eine Zustandstransition des Boolschen Netzwerkes anhand seines lokalen Zustandes durch. Wird hierbei ein Knoten aktiv, so wird die verknüpfte Graphenoperation (falls vorhanden) auf die Zelle angewendet. Dabei können auch mehrere Operationen gleichzeitig aktiv werden. Eine Besonderheit betrifft die beiden zellteilende Operationen `Split` und `Duplicate`. Um eine unterschiedliche Fortentwicklung beider Zellen zu ermöglichen, wird ein bestimmtes Bit des Netzzustandes in einer der beiden Zellen gesetzt, während es in der anderen Zelle gelöscht wird. Dies entspricht der Aufgabelung des

Instruktionsbaumen in einen linken und einen rechten Teilbaum bei Gruau's Zellularer Kodierung.

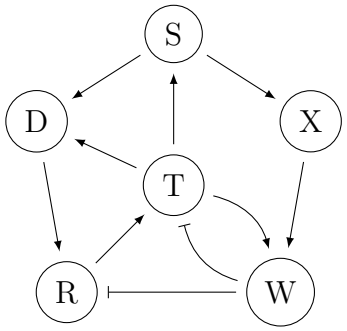


Abbildung 3.11.: GRN in Form eines Boolschen Netzwerkes

Kodierung Die Autoren verwenden die von Reil [Rei99] vorgeschlagene Kodierung des *Artificial Genome*. Ein Genom besteht hierbei aus einer Zeichenkette, die aus den vier Zeichen 0, 1, 2 und 3 gebildet wird. Jedes dieser vier Zeichen repräsentiert eine der vier Nukleinbasen der biologischen DNA, Adenin, Guanin, Cytosin und Thymin. Innerhalb einer solchen Basenkette markiert die Folge 0101 (sogenannte TATA-Box[24]) den Beginn einer Gendefinition. Direkt im Anschluss daran folgt die Definition des Gens, repräsentiert durch die nächsten vier Basen. Ein Gen enspricht dabei einem Knoten des Boolschen Netzwerkes. Die Wechselwirkung dieses Gens mit den anderen Genen der DNA ist beschrieben durch sein Genprodukt. Dieses wird gebildet, indem jede Base stellenweise um eins addiert wird. So ist das Genprodukt von 1230 beispielsweise 2341. Nach dem Genprodukt wird nun in den Bereichen vor der TATA-Box eines jeden Gens gesucht. Wird ein Vorkommen gefunden, so besteht eine Wechselwirkung zwischen den beiden Genen. Ob diese hemmend oder anregend ist, entscheidet das Genprodukt: Ist die letzte Base eine 0, so wirkt das Genprodukt hemmend, ansonsten anregend. Entsprechend den Genen und den Wechselwirkungen zwischen diesen wird ein Boolsches Netzwerk erzeugt.

Evolutionärer Algorithmus Die Autoren verwenden eine zufällig erzeugte Startpopulation mit 200 kurzen, nur aus 256 Basen bestehenden, Genomen. Im Schnitt enthalten dabei 256 Basen genau ein einzelnes Gen.

In jeder Generation werden 25% der Population mutiert. Die folgenden, biologisch-motivierten Mutationoperatoren werden verwendet:

- Ein-Punkt: Ändern eines einzelnen Zeichens (Base).
- Duplikation: Ein Teil des Genoms wird zufällig ausgewählt und an der gleichen Stelle verdoppelt.

[24] Auch Goldberg-Hogness-Box genannt.

- Transposition: Ein Teil des Genoms wird zufällig ausgewählt und an eine andere Stelle verschoben.
- Deletion: Ein Teil des Genoms wird zufällig ausgewählt und gelöscht.
- Inversion: Ein Teil des Genoms wird zufällig ausgewählt und durch seine invertierte Reihenfolge ersetzt.

Die zufällig ausgewählten Teile des Genoms haben hierbei immer eine feste Länge von 256 Basen. Die durch Mutation neu gewonnenen Genome werden der Gesamtpopulation hinzugefügt. Selektion findet durch Tournament-Wahl der Grösse 2 statt.

Fazit

- Die Autoren präsentieren ein biologisch motiviertes Verfahren, welches auch zur Erzeugung von neuronalen Netzen verwendet werden kann. Ein Boolsches Netzwerk übernimmt die Rolle eines Genregulationsnetzwerk (GRN) und steuert die Entwicklung der Zellen. Ebenso ist die Kodierung des Genoms, als auch die Mutationsoperatoren stark an die Biologie angeknüpft.
- Vom Prinzip her ähnelt das Verfahren der Zellularen Kodierung von Gruau. An die Stelle der Instruktionsbäume tritt jedoch ein Boolsches Netzwerk. Zudem unterscheidet sich das Verfahen insbesondere darin, dass eine einfache kantenorientierte Graphgrammatik mit nur fünf parameterlosen Operationen verwendet wird.
- Die Gewichtung der Kanten wird durch wiederholte Anwendung der Kantenoperation **Resize** eingestellt. Ebenso wird der Typ einer Zelle, die ein späteres Element des neuronalen Netzes darstellt, durch wiederholte Anwendung der Kantenoperation **Type** definiert.
- Der Zellgraph entspricht nicht dem späteren Netzwerk, sondern muss erst in dieses umgewandelt werden. Die Kanten des Zellgraphen entsprechen dabei den Knoten in Gruau's Zellularer Kodierung. Die Knoten repräsentieren mögliche Verbindungen zwischen den Zellen. Dies hat zur Folge, dass die Komplexität des zu erzeugenden neuronalen Netzes nicht direkt aus dem Zellgraphen ersichtlich ist. Daher lässt sich kein, in Bezug auf die Komplexität, aussagekräftiges Abbruchkriterium für die Entwicklung des Zellgraphen definieren, anders als dies bei Gruau's Zellularer Kodierung der Fall ist.
- Anders als bei Gruau's Instruktionsbäumen, erschwert das Boolsche Netzwerk eine genaue Analyse der genauen Entwicklungsschritte. Ebenso ist eine manuelle Beschreibung eines Netzes praktisch nicht möglich.

3.3.5. NeuroEvolution augmentierter Topologien

Die Autoren der Veröffentlichung *NeuroEvolution of Augmented Topolgoies* (NEAT)[25] [SM02] identifizieren drei wesentliche Herausforderung an Algorithmen, die sowohl Topologie als auch Gewichtung evolutionär entwickeln:

1. Wie können Genome unterschiedlicher Topologie sinnvoll miteinander gekreuzt werden? Welche genetische Repräsentation ermöglicht dies?
2. Wie kann topologische Innovation, die einige Generationen braucht um sich optimal zu entwickeln, vor dem vorzeitigen Ausscheiden aus der Population geschützt werden?
3. Wie können Topologien während der gesamten Dauer der Evolution minimiert werden, ohne dass hierfür auf eine künstliche, die Komplexität des Genoms bestimmende, Fitnessfunktion zurückgegriffen werden muss?

Für alle drei Herausforderungen haben die Autoren dabei eine geeignete Lösung:

1. Durch die Markierung der Gene entsprechend ihrer historischen Herkunft erreichen sie, dass bei der Kreuzung zweier Genome, übereinstimmende Gene wechselseitig zugeordnet werden können (*Gene Alignment*).
2. Durch *Nischenbildung* (Subpopulationen) erreichen sie den Schutz neu entstehender topologischer Innovation.
3. Indem sie die Evolution, ausgehend von einer minimalen Topologie, nur bei Bedarf schrittweise komplexer werden lassen (*Complexification*), erreichen sie, dass komplexere Topologien nur dann entstehen (können) wenn sie auch sinnvoll sind.

Allerdings verwenden sie in ihrem Ansatz eine direkte Kodierung der Knoten und Verbindungen und können daher wiederkehrende Strukturen nicht effizient kodieren. Somit ist NEAT auf die Evolution von Netzen mit geringer Komplexität beschränkt.

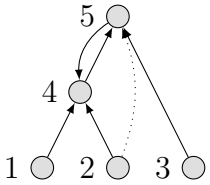
Kodierung des Genoms Das Genom ist in zwei Typen von Genen unterteilt: *Knotengene* beschreiben die Eigenschaften eines jeden Knotens, *Verbindungsgene* die einzelner Verbindungen. Die Attribute eines Verbindungsgens sind: Quellknoten, Zielknoten, Gewichtung der Verbindung, Aktivität des Gens, sowie dessen Innovationsnummer. Die Attribute eines Knotengens sind in diesem Kontext nicht weiter von Bedeutung. Abbildung 3.12a zeigt die genetische Repräsentation des in Abbildung 3.12b dargestellten Netzes.

[25] Die Netz-Konfigurationen der Abbildungen dieses Abschnittes sind dem Paper entnommen.

Node 1	Node 2	Node 3	Node 4	Node 5
Input	Input	Input	Hidden	Output

$1 \rightarrow 4$	$2 \rightarrow 4$	$2 \rightarrow 5$	$3 \rightarrow 5$	$4 \rightarrow 5$	$5 \rightarrow 4$
0.7	0.5	0.5	0.2	0.4	0.6
An	An	Aus	An	An	An
Innov 1	Innov 3	Innov 4	Innov 5	Innov 6	Innov 9

(a) Genom



(b) Netzwerk

Abbildung 3.12.: NEAT: Genotyp und ausgeprägter Phenotyp

Festhalten des historische Ursprungs ermöglicht Gen-Zuordnung Wie oben schon erwähnt wurde besteht eine der grossen Herausforderungen darin, Genome die unterschiedliche Topologien repräsentieren, sinnvoll miteinander zu kreuzen. Um dies zu ermöglichen, muss eine Zuordnung zwischen den Genen beider Genome gefunden werden. In NEAT wird dies erreicht, in dem jede strukturelle Änderung mit einer neuen globalen Innovationsnummer versehen wird. Dabei wird eine strukturelle Änderung nur durch das Hinzufügen einer neuen Verbindung, oder durch Veränderung des Quell- oder Zielknotens einer bestehenden Verbindung hervorgerufen. Das Hervorbringen von neuer struktureller Innovation geschieht zudem ausschliesslich durch Mutation, während bei der Kreuzung zweier Genome (Crossover) lediglich aus den vorhandenen Genen ein neues Genom rekombiniert wird. Dabei könnte die Gewichtung zueinander passender Gene bei der Kreuzung durchaus in geeigneter Weise miteinander verknüpft werden, da dies keiner strukturellen Änderung entspricht. Dies steht in starkem Kontrast zu anderen Verfahren, die durch Kreuzung strukturelle Veränderung herbeiführen, während Mutation in diesen Verfahren hauptsächlich dazu dient Parameterwerte anzupassen[26].

Mutation Die Mutation in NEAT kann entweder die Gewichtung von Verbindungen (oder Knotenparameter sofern vorhanden) verändern, oder eine *strukturelle Veränderung* hervorrufen. Eine strukturelle Veränderung kann hierbei auf zwei verschiedene Arten erfolgen:

- Bei der *AddConnection* Mutation wird eine neue Verbindung zwischen zwei bislang nicht verknüpften Knoten realisiert. Dadurch entsteht ein einzelnes neues Verbindungsgen. (Abbildung 3.13a)
- Bei der *AddNode* Mutation wird eine bestehende Verbindung in der Mitte getrennt und ein neuer Knoten eingefügt. Es entstehen zwei neue Verbin-

[26] Beispielsweise verwendet die zellulare Kodierung von Gruau Genetisches Programmieren. Die Kreuzung bewirkt hierbei den Austausch einzelner Teilbäume zwischen den Genomen, während die Mutation maßgeblich dazu dient, Parameterwerte anzupassen.

dungsgene, wobei das ursprüngliche Verbindungsgen deaktiviert wird. (Abbildung 3.13b)

Hierbei bekommt jedes neu erzeugte Verbindungsgen eine global streng monoton steigende Innovationsnummer zugeordnet. Somit lässt sich jedes Verbindungsgen eindeutig durch die gesamte Evolution hinweg identifizieren. Zudem lassen sich auf diese Weise Verbindungsgene entsprechend ihrem zeitlichen Enstehen nach sortieren.

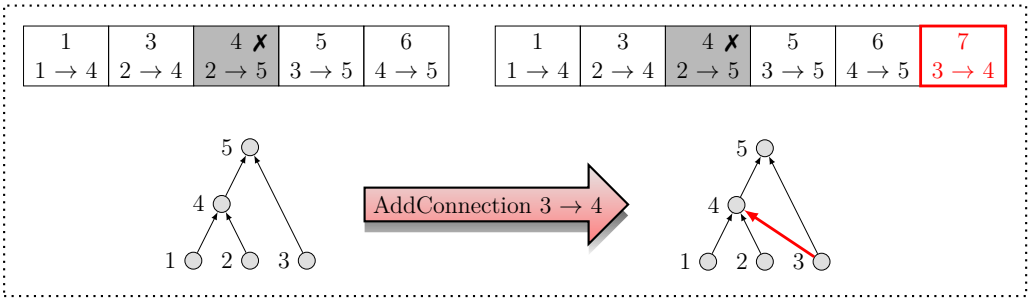
Kreuzung zweier Genome Die Kreuzung zweier Genome ist in Abbildung 3.14 dargestellt. Bei der wechselseitigen Zuordnung der Gene zweier Genome sind drei Fälle zu unterscheiden:

- Gene mit *gleichem Ursprung*
- *Disjunkte* Gene
- *Überschüssige* Gene (excess)

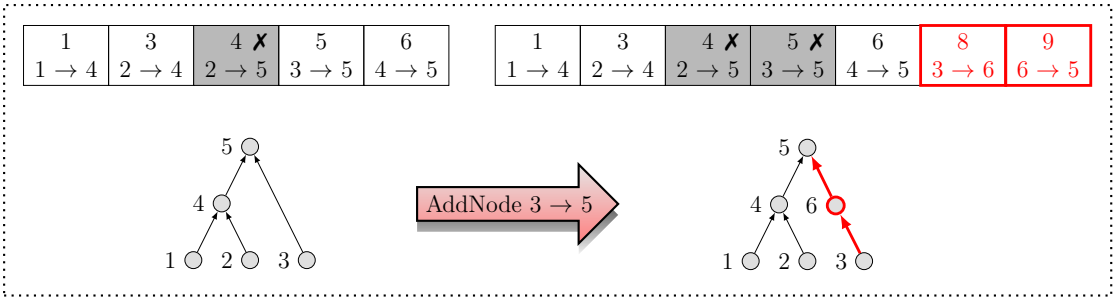
Gene mit dem gleichen Ursprung (d.h. mit gleicher Innovationsnummer) können direkt in den Nachkommen übernommen werden. Diese unterscheiden sich lediglich in ihrem Parameterwert. Die den beiden Genen zugeordneten Parameterwerte können hierbei in geeigneter Weise miteinander verknüpft werden. Beispielsweise könnte der Parameterwert, der dem Gen des fitteren Elterngenoms entstammt, dominierend sein und somit direkt und unverändert übernommen werden. Ebenfalls denkbar wäre eine numerische Verknüpfung beider Parameterwerte, beispielsweise durch *Simulated Binary Crossover* (SBX) [DA94]. NEAT verknüpft die Gewichtung passender Gene jedoch nicht, sondern übernimmt das Gewicht von einem zufällig ausgewählten Elternteil, unabhänig davon, ob dieses fitter ist oder nicht.

Disjunkte Gene sind Gene, die entweder in dem einen Elterngenom oder in dem anderen vorkommen, nicht jedoch in beiden. Dabei befindet sich die Innovationsnummer eines disjunkten Gens innerhalb des Bereichs aller Innovationsnummern des zweiten Genoms. Befindet sich die Innovationsnummer ausserhalb dieses Bereiches, so handelt es sich um ein überschüssiges Gen, ein Gen also, welches erst zu einem späteren Zeitpunkt hinzu kam. Disjunkte und überschüssige Gene werden in NEAT vom fitteren der beiden Elterngenome übernommen. Sind beide Elterngenome gleich fit, so entscheidet der Zufall. In Abbildung 3.14 haben beide Elterngenome die gleiche Fitness. Es ist somit reiner Zufall, dass alle disjunkten und überschüssigen Gene beider Eltern im Nachkommen enthalten sind.

Schutz neuer Innovationen durch Nischenbildung Ein weiterer wichtiger Aspekt von NEAT ist es, neu entstehende strukturelle Innovation in einer Population vor dem vorzeitigen Ausscheiden aus der Evolution zu beschützen, in dem es Nischen bildet, in denen nur Genome mit ähnlichen Innovationen gegeneinander konkurrieren. Somit wird einer neuen enstandenen strukturellen Innovation Zeit gegeben, sich zu entwickeln.



(a) Hinzufügen einer neuen Verbindung



(b) Erzeugen eines neuen Knotens durch Aufspalten einer bestehenden Verbindung

Abbildung 3.13.: Strukturelle Mutation in NEAT

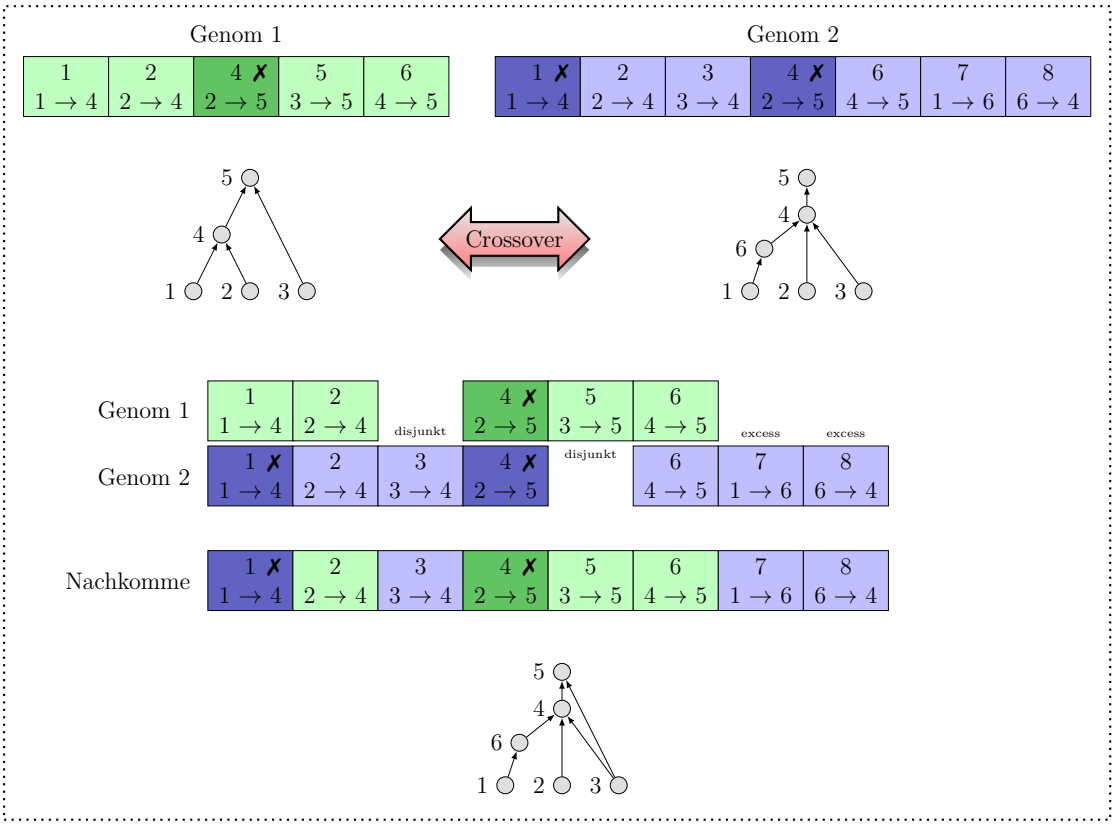


Abbildung 3.14.: Kreuzung zweier Genome in NEAT

Die Nischenbildung wird dabei maßgeblich durch die Ähnlichkeit der Genome untereinander bestimmt. Das gleiche Prinzip findet sich so oder in abgewandelter Form auch in der Natur, wo die Individuen innerhalb einer Nische bzw. einer Art, grössere Ähnlichkeiten in ihrer DNA aufweisen als zu artenfremden Individuen.

Die *Kompatibilität* δ_{ij} (bzw. genetische Ähnlichkeit) zweier Genome i und j ist gegeben durch Gleichung 3.1. Hierbei bezeichnet N die Gesamtzahl der Gene des längeren Genoms, E die Anzahl überschüssiger Gene, D die Anzahl disjunkter Gene, und $\overline{W}$ die mittlere Differenz der Gewichtungen aller übereinstimmender Gene. Die Koeffizienten c_1, c_2, c_3 ermöglichen eine differenzierte Gewichtung der einzelnen Komponenten und sind entsprechend von aussen vorzugeben.

$$\delta_{ij} = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W} \tag{3.1}$$

Ein Genom i wird einer Nische zugeordnet, wenn für ein beliebiges Genom j dieser Nische gilt: $\delta_{ij} < \delta_t$, mit δ_t als obere Schranke. Dabei wird jedem Genom die Nische zugeordnet, auf die dieses Kriterium zuerst zutrifft.

Die Reproduktion neuer Genome findet ausschliesslich innerhalb der Nischen statt. Sei N_j die aktuelle Anzahl der Individuen in Nische j, f_{ij} die Fitness des i-ten Individuums in Nische j, die mittlere Fitness einer Nische j gegeben durch $\hat{f}_j$ (3.2), dann berechnet sich die neue Anzahl an Individuen der Nische N'_j aus dem Verhältnis der mittleren Fitness der Nische zur Summe aller mittlerer Fitnesswerte (3.3).

$$\hat{f}_j = \sum_{i=1}^{N_j} \frac{f_{ij}}{N_j} \tag{3.2}$$

$$N'_j = \frac{\hat{f}_j}{\sum\limits_k \hat{f}_k} \tag{3.3}$$

Die Zahl der Nachkommen ändert sich somit je nachdem ob die durchschnittliche Fitness einer Nische oberhalb oder unterhalb des Durschnittes der gesamten Population liegt. Um die Nachkommen zu erzeugen, werden $r\%$ der besten Individuen einer Nische zufällig miteinander gepaart. Die so erzeugten Nachkommen ersetzen dabei alle Individuen dieser Nische komplett.

Schrittweises Erhöhen der Komplexität NEAT unterscheidet sich von anderen Ansätzen auch darin, dass die Startpopulation nur Genome von minimaler Topologie enthält (d.h. keine versteckten Knoten), während andere Verfahren häufig mit einer

Anfangspopulation bestehend aus zufälligen Topologien starten. Dies wird insbesondere bei Verfahren benötigt, die den Schutz kleiner, struktureller Veränderungen über einige Generationen hinweg nicht gewährleisten. Durch eine so gewählte Anfangspopulation gewährleisten sie zumindest ein gewisses Maß an Diversität, wobei keine dieser zufällig erzeugten Anfangstopologien jemals auf seine Fitness geprüft wurde. Dies hat zur Folge, dass diese Verfahren häufig einen viel grösseren Suchraum durchlaufen müssen als dies bei NEAT der Fall ist.

Fazit NEAT löst das Problem bei der Kreuzung von Genomen ungleicher Topologien auf effiziente Art und Weise indem es den historischen Ursprung struktureller Änderungen kodiert. Durch Nischenbildung wird zudem erreicht, dass kleine strukturelle Änderungen Zeit haben sich zu entfalten um sich somit eventuell positiv auf die Fitness auswirken zu können. Hierbei wird die Komplexität einer Topologie nur dann erhöht wenn es auch erforderlich ist. Allerdings verwendet NEAT eine direkte Kodierung und ist somit nicht geeignet, wiederkehrende Muster effizient zu kodieren. Benchmarks zeigen jedoch, dass NEAT eine wesentlich bessere Konvergenz aufweisst als indirekt-kodierende Verfahren wie beispielsweise Gruau's Zellulare Kodierung.

3.3.6. Compositional Pattern Producing Network

Compositional Pattern Producing Network (CPPN) [Sta07] sind azyklische Netzwerke, die durch die Komposition einzelner mathematischer Funktionen beliebig dimensionale Muster erzeugen. An die Eingänge des Netzwerkes werden hierbei die Ortskoordinaten gelegt. Am Ausgang wird der berechnete Bildwert abgelesen. Siehe Abbildung 3.15. Der Unterschied zu künstlichen neuronalen Netzen besteht im Wesentlichen in der Verwendung mehrerer verschiedenartiger Funktionen in den Knoten des Netzes[27].

Viele in der Natur vorkommende Arten von Mustern können mit diesem Ansatz erzeugt werden, wie etwa:

- *Symmetrie*, z.B. links-rechts Symmetrie bei Wirbeltieren
- *Unvollkommene Symmetrie*, z.B. Rechtshändigkeit
- *Wiederholung*, z.B. rezeptives Feld im Cortex
- *Wiederholung mit Variation*, z.B. Kortikale Säulen

Erreicht wird dies, indem asymmetrische Funktionen (z.B. Sigmoidfunktion), symmetrische Funktionen (z.B. Gaussche Funktion) sowie sich periodisch wiederholende Funktionen (z.B. Sinus oder Modulo) in geeigneter Weise miteinander kombiniert werden. Die Interpretation des von einem CPPN erzeugten Bildwertes hängt hierbei von der jeweiligen Anwendung ab.

[27] In künstlichen neuronalen Netzen kommt häufig nur eine Sigmoidfunktion bzw. tanh als Schwellwertfunktion zum Einsatz.

Im 2-dimensionalen Raum sind CPPN durch die Abbildung der beiden Raumkoordinaten x und y auf den Funktionswert $f(x, y)$ definiert, wobei $f(x, y)$ beliebig aus weiteren Funktionen zusammengesetzt sein kann. Einige Beispiele von durch CPPN erzeugter Muster mitsamt ihrer erzeugenden Funktionen sind in Abbildung 3.16 gegeben. Sehr schön lässt sich die Kombination verschiedener Muster anhand des mittleren Bildes (Zeile 2, Spalte 2) erkennen, welches durch die Überlagerung des letzes Bildes aus Zeile 1 mit dem ersten Bild aus Zeile 2 entstanden ist.

HyperNEAT Hypercube-basierte NeuroEvolution augmentierter Topologien [GS07] verwendet NEAT[28] um ein 4-dimensionales CPPN evolutionär zu entwickeln. Das Bild der so erzeugten Funktion $f(x1, y1, x2, y2)$ wird hierbei als Verbindungsparameter[29] zwischen den Neuronen an Position $(x1, y1)$ und $(x2, y2)$ gedeutet. Um aus dem so gewonnenen CPPN ein neuronales Netz zu entwickeln, muss zudem ein *Substrat* vorgegeben werden. Das Substrat bestimmt an welchen Positionen Neuronen platziert sind. Ist das neuronale Netz in ein reales System eingebettet, so kann sich das Substrat an der geometrischen Anordnung der physischen Sensoren des umgebenden Systems orientieren. Bei einem virtuellen System ohne physische Entsprechung kann eine beliebige, sinnvolle Anordnung gewählt werden. Eine mögliche Anordnung ist beispielsweise in Form eines regulären Gittermusters.

Seien zwei Neuronen i und j zusammen mit ihren Koordinaten (x_i, y_i) und (x_j, y_j) gegeben, so wird genau dann eine Synapse zwischen diesen beiden Neuronen gebildet wenn $f(x_i, y_i, x_j, y_j) = \alpha < \delta$, wobei α die Gewichtung der Synapse angibt und δ eine obere Schranke. Zusätzlich zu dem Wert des Bildes von f lässt sich die euklidische Distanz beider Neuronen dazu benutzen, die Lokalität von Verbindungen zu beschränken.

Durch die Verwendung mehrerer (logischer) Schichten können mit HyperNEAT beliebig komplexe Netze erzeugt werden. Dabei wird jede Schicht durch die Anzahl und Anordnung der Neuronen im Substrat beschrieben. Meisst unterscheiden sich die Schichten in einer Koordinate, z.B. in der z-Koordinate bei einem dreidimensionalen Substrat. Zyklische (oder rekurrente) Netze können auf verschiedene Arten realisiert werden. Eine Möglichkeit besteht in der speziellen Deutung des Bildwertes des CPPNs, wobei das Vorzeichen die Richtung der Verbindung beschreiben könnte. Durch Hinzunahme weiterer Ausgänge können bei Bedarf andere Eigenschaften des Netzes beschrieben werden.

Evolvable-Substrate HyperNEAT ES-HyperNEAT [RS12] ist eine Erweiterung von HyperNEAT. Hierbei wird, zusätzlich zu den Verbindungen zwischen den Neuronen eines Substrates, die geeignete Konfigurationen, d.h. Anzahl und Anordnung der Neuronen auf dem Substrat, evolutionär entwickelt.

[28] Siehe Abschnitt 3.3.5

[29] z.B. Verzögerungszeit der Synapse

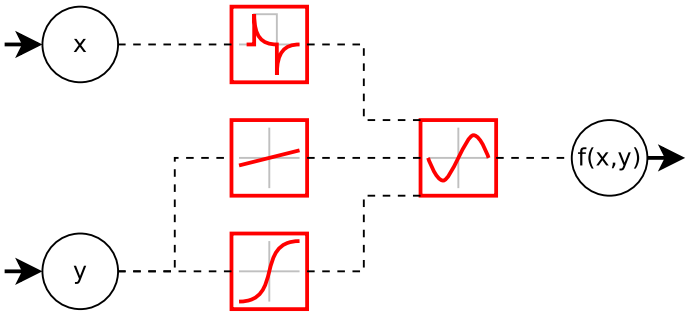
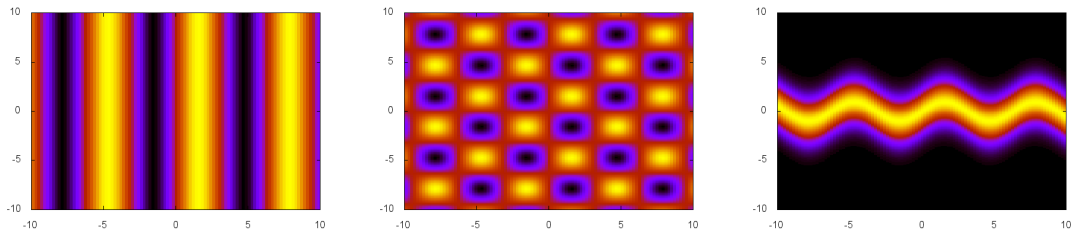
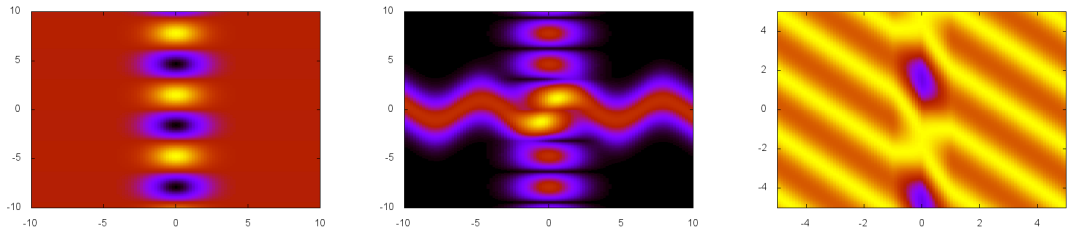


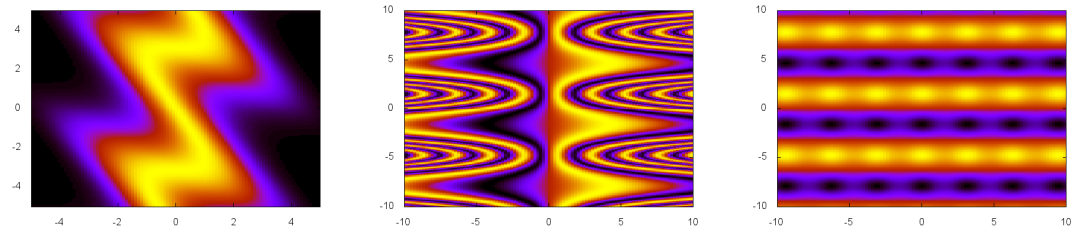
Abbildung 3.15.: Ein Compositional Pattern Producing Network (CPPN) zur Berechnung von $f(x, y)$. Die Knoten des Netzes bestehen hierbei aus mathematischen Funktionen, wie z.B. Sinus, Sigmoid oder Gaussche-Funktion.



(a) $\sin x$ **(b)** $\sin x \cdot \sin y$ **(c)** $G_y(\sin x)$



(d) $G_0(x) \cdot \sin y$ **(e)** $|G_y(\sin x)|+|G_0(x)\cdot \sin y|$ **(f)** $G_0(s(x) + \sin(x + y))$



(g) $G_0(x + \sin(x + y))$ **(h)** $\sin(x \cdot e^{\sin y})$ **(i)** $G_0(\sin x) \cdot \sin y$

Abbildung 3.16.: Beispiele einiger CPPNs. Mit Gausscher Funktion $G_\mu(x) = \frac{1}{\sqrt{2*\pi}} \cdot e^{\frac{-(x-\mu)^2}{4}}$ und symmetrischer Funktion $s(x) = |x| > 1.0\ ?\ 0.0 : 1.0 - |x|$.

Fazit

- Der Einsatz von CPPNs ermöglicht die Beschreibung von beliebigen funktionalen Zusammenhängen indem der Bildwert des Netzes in geeigneter Weise interpretiert wird.
- CPPNs erlauben zudem das Nachbilden von Mustern wie sie auch in der Natur auftreten. Insbesondere ist es möglich, Muster zu erzeugen, die folgende Eigenschaften aufweisen: *Symmetrie*, *Unvollkommene Symmetrie*, *Wiederholung* und *Wiederholung mit Variation*.
- Hypercube-basierte NeuroEvolution augmentierter Topologien (HyperNEAT) ermöglicht das effiziente Erzeugen sehr grosser Netze, indem es NEAT einsetzt um CPPN evolutionär zu entwickeln. Der Bildwert wird hierbei als Verbindungsgewicht zwischen je zwei Neuronen gedeutet. Die Modularität wird durch die Kombination von periodischen Funktionen mit asymmetrischen oder symmetrischen Funktionen erreicht. Allerdings muss die Anzahl und Positionierung der Neuronen (Substrat) von aussen vorgegeben werden.
- ES-HyperNEAT ermöglicht darüber hinaus das evolutionäre Entwickeln der Substrate.

3.3.7. Hierarchische Beschreibung nach Schmidt

Johannes Schmidt beschreibt in seiner Diplomarbeit [Sch09] einen hierarchischen Ansatz zur Erzeugung gepulster neuronaler Netze. Moduldefinitionen beschreiben hierbei die Struktur und Parameterisierung einzelner Teilnetze, wobei diese sich wechselseitig referenzieren und instanzieren können

Ein Genom in seinem Verfahren setzt sich aus einer Liste von Moduldefinitionen zusammen. Eine Moduldefinition besteht, analog zur biologischen DNA, aus einem nichtkodierenden Bereich (Intron), sowie einem kodierenden Bereich (Exon). Das Intron unterteilt sich nochmals in einzelne Bereiche und enthält Parameter mit folgenden Zuständigkeiten:

- Wahrscheinlichkeiten für Evolutionären Algorithmus
- Parameter für Neuronen
- Parameter für Synapsen
- Steuerung der Verbindungsweitergabe
- Steuerung der Modulerzeugung
- Steuerung der Markenweitergabe
- Steuerung der Bereichsvervielfältigung

Das Exon (kodierender Bereich) steuert die strukturelle Ausprägung eines Moduls. Erlaubte Codons im Exons sind hierbei:

n Erzeugt ein Neuron

a Erzeugt eine Marke

b Erzeugt eine Synapse

c Weitergabe einer Marke

x Erzeugt ein Tochtermodul

d Vervielfältigt die Regelkette des Exons

Die genaue Arbeitsweise der Erzeugung eines Netzes aus einem Genom soll im Folgenden anhand eines konkreten Beispiel erläutert werden.

Beispiel Gegeben sei ein Genom bestehend aus zwei Moduldefinitionen A und B. Moduldefinition A enthält im Exonbereich die Kodierung **nanacx**, B hingegen **nb**. Die erste Moduldefinition in jedem Genom ist speziell, da immer ein Modul anhand dieser Definition zu Beginn der Netzgenerierung erzeugt wird. Die Entwicklung des Netzes läuft wie folgt ab:

1. Die erste Instruktion des Exons von A ist **n** und erzeugt ein Neuron. Die Parameter dieses Neurons werden aus dem Intronbereich des Moduls gelesen. Das folgende **a** erzeugt eine Marke, die an das zuletzt erzeugte Neuron geknüpft ist. Durch Weitergabe von Marken lassen sich konkrete Verbindungen ausprägen.
2. Durch die Wiederholung von **na** wird erneut ein Neuron mitsamt verknüpfter Marke erzeugt.
3. Das folgende **c** sorgt für die Weitergabe einer Marke an ein anderes Modul. Welche Marke genau an welches Modul weitergegeben wird ist im Abschnittsbereich der Markenweitergabe des Introns definiert. Eine Markenweitergabe findet erst dann statt, wenn alle Module ausgeprägt sind. Dies ist sinnvoll, da in dem momentanen Zustand noch kein weiteres Modul existiert. Weiterhin kommen als mögliche Module zur Weitergabe einer Marke nur die Tochtermodule sowie das Elternmodul in Frage.
4. Im letzten Schritt der Moduldefinition von A wird durch **x** erreicht, dass ein neues Modul erzeugt wird. Welches Modul dabei genau erzeugt wird ist im Abschnitt der Modulerzeugung des Introns geregelt. Da unser Genom aus nur zwei Modulen besteht, und eine rekursive Modulerzeugung generell unterbunden wird, bewirkt das **x** in unserem Beispiel die Ausprägung eines Tochtermoduls von A anhand der Moduldefinition von B.
5. Das erste Codon der Moduldefinition von B erzeugt ein Neuron (**n**).
6. Das folgende **b** bewirkt die Bildung einer Synapse zu dem zuletzt erzeugten Neuron. Die Synapsenparamter werden hierfür dem entsprechenden Bereich des Introns von B entnommen. Die Wahl des postsynaptischen Neurons erfolgt aus der Menge der an dieses Modul weitergegebener Marken und wird

gesteuert durch Parameter des für die Verbindungsweitergabe zuständigen Intronbereichs.

Abbildung 3.17 zeigt die zwei für dieses einfache Beispiel möglichen Topologien des Netzes. Entweder is Neuron N1 mit N3 verbunden, oder N2 ist mit N3 verbunden, abhängig von den entsprechenden Werten im Intronbereich der Verbindungsweitergabe.

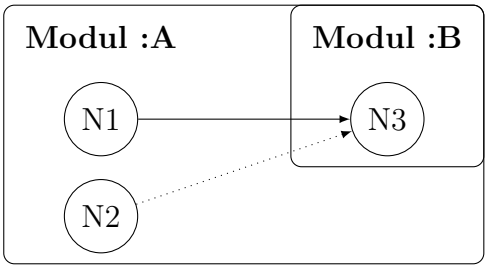


Abbildung 3.17.: Mögliche Ausprägungen des Beispielgenoms bestehend aus den Moduldefinitionen A: **nanacx** und B: **nb**. Die gestrichelte Verbindung beschreibt eine alternative Topologie.

Reproduktion Bei der Kreuzung zweier Genome wird uniformes Crossover eingesetzt. Die Moduldefinitionen bilden hierbei die Basiseinheiten des Austausches, d.h. für jede Moduldefinition wird erneut entschieden aus welchem Genom sie übernommen werden. Die Wahrscheinlichkeit für diesen Vorgang wird der jeweiligen Moduldefinition (dem Intronbereich zuständig für die Steuerung des EA) des fitteren Elternteiles entnommen.

Nach der Kreuzung zweier Genome wird das neu erzeugte Genome durch Mutation verändert. Mögliche Operationen sind hierbei:

- Löschen einer Moduldefinition
- Hinzufügen einer neuen Moduldefinition
- Duplizieren einer Moduldefinition
- Verändern einer Moduldefinition

Da die Kreuzung positionsabhängig ist, geschieht das Löschen durch deaktivieren einer Moduldefinition. Dies erhält somit die Reihenfolge der anderen Moduldefinitionen. Ebenso wird bei Hinzufügen einer neuen Moduldefinition auf deaktivierte Moduldefinitionen zurückgegriffen, oder es wird die Moduldefinition an das Ende des Genoms gehängt. Die Wahrscheinlichkeiten zur Steuerung des Mutationsvorganges werden der globalen Konfiguration entnommen.

Fazit

- Schmidt erreicht eine effiziente Kodierung wiederholt auftretender Strukturen durch die Kapselung in Moduldefinitionen.
- Crossover findet auf der Ebene der Moduldefinitionen statt. Moduldefinitionen an gleicher Position werden dabei gegeneinander ausgetauscht. Somit ist die Position einer Moduldefinition innerhalb des Genoms entscheidend[30]. Schmidt hofft hierbei[31], dass sich durch den Lauf der Evolution eine entsprechende Ordnung innerhalb des Genoms automatisch einstellt, sodass Genome an gleicher Position eine ähnliche Struktur und Funktion aufweisen werden. Ist dies nicht der Fall, so bewirkt die Kreuzung zweier Genome ein zufälliges Mischen der Moduldefinitionen.
- Die Struktur einzelner Module muss mühsam durch Mutation gefunden werden. Struktureller Austausch findet nur auf Basis des Austausches ganzer Module statt, setzt also das Vorhandensein bereits bekannter, guter, Module voraus.
- Eine genaue Steuerung der Verdrahtung zwischen Modulen lässt sich nicht erreichen, da hierfür Zufallswerte eingesetzt werden. Es ist somit auch sehr schwer anhand eines vorliegenden Genoms auf dessen Ausprägung als Netz zu schliessen.
- Der intensive Einsatz von Zufallsentscheidungen erschwert eine genauere theoretische Abschätzung des Verfahrens.

3.3.8. Biologisch-motiviertes Verfahren von Isto Aho et al.

In ihrer Veröffentlichung *Searching Neural Network Structures With L Systems and Genetic Algoritms* [Aho+97] beschreiben Isto Aho et al. ein biologisch motiviertes Verfahren für die Erzeugung neuronaler Netze, welches neben der Kombination aus evolutionärem Algorithmus und L-System, Elemente eines räumlichen Entwicklungsalgorithmus zur Ausprägung der Axone und Dendriten enthält. Ein weiteres besonderes Merkmal Ihres Verfahrens ist die Verwendung eines kontextsensitiven L-Systems, dass anstatt auf einer linearen Zeichenkette, auf einer 2-dimensionalen Zell-Matrix operiert. Dies ist äquivalent zu einem zellularen Automaten mit Moorscher Nachbarschaft, bei dem die Regeln des L-Systems in entsprechende Übergänge des zellularen Automaten umgewandelt werden.

Jede Zelle der Matrix repräsentiert ein potentielles Neuron, wobei der Typ des Neurons die zu verwendende Schwellwertfunktion des Neurons im ausgeprägten Netz definiert. Eine Zelle der Matrix kann durchaus auch nicht belegt sein.

[30] Die Position einer Moduldefinition in seinem Verfahren entspricht im Grunde der Innovationsnummer von NEAT (Abschnitt 3.3.5).

[31] ‚So könnte sich im Laufe der evolutionären Optimierung eine Einigkeit darüber ergeben, welche Teilbereiche der kDNA für gewisse Funktionen des SNN kodieren.‘ ([Sch09], Seite 73)

Anstelle von direkt kodierten Verbindungen zwischen den Neuronen treten Duftstoff-Markierungen (*scents*), die in einem separaten Entwicklungsschritt das Auswachsen der Axone regeln. Dabei besitzt eine Zelle jeweils einen Wert für den Quell-Duftstoff und den Ziel-Duftstoff. Axone wachsen in Richtung des nächstgelegenen Neurons mit gleicher oder ähnlicher Duftstoff-Markierung, jedoch einheitlich in eine Richtung (z.B. von links nach rechts), um zyklische Verbindungen zu vermeiden. Nachdem das Wachstum der Axone abgeschlossen ist, wachsen die Dendriten aus. Hierbei wird in entgegengesetzter Richtung zum Wachstum der Axone (z.B. von rechts nach links), von einem Neuron ausgehend nach einem Axon gesucht, mit dem eine synaptische Verbindung eingegangen werden soll. Dies geschieht erst mit geringem Abstand, dann im weiteren Umfeld des Neurons. Während pro Neuron nur genau ein Axon auswächst, so sind synaptische Verbindungen der Dendriten zu mehr als einem Axon erlaubt.

Neben dem Neuronentyp (Schwellwertfunktion) und den Duftstoff-Markierungen besitzt jede Zelle der Matrix noch ein weiteres Attribut, welches die Gewichtung des Axons angibt. Die Gewichtung des Axons hat dabei Einfluss auf die Lernrate des Backpropagation-Lernalgorithmus und dient der späteren Optimierung. Es gilt anzumerken, dass die Autoren kein gepulstes neuronales Netz entwickeln, sondern ein künstliches neuronales Netz[32].

Durch einen evolutionären Algorithmus werden Regeln eines L-Systems erzeugt. Eine Besonderheit, die Regeln des L-Systems betreffend ist, dass diese bei jeder Anwendung bzw. Ersetzung *altern*. Das maximale Alter einer Regel ist dabei ebenso wie die Regel selbst, im Genom definiert, und wird daher auch vom evolutionären Algorithmus optimiert. Die regelspezifische Alterung dient der Terminierung des Substitutionsalgorithmus und erlaubt eine Differenzierung einzelner Regeln[33].

Bei den Regeln gilt es zu beachten, dass diese nur den Zustand der (mittleren) Zelle verändern, nicht jedoch den Zustand der Zellen in der umliegenden Nachbarschaft (bzw. des äusseren Kontextes). Vier verschiedene Typen von Regeln kommen zum Einsatz (siehe die Beispiele in Abbildungen 3.18a bis 3.18d), die sich lediglich in der erforderlichen Übereinstimmung des Kontextes mit der Zellmatrix unterscheiden:

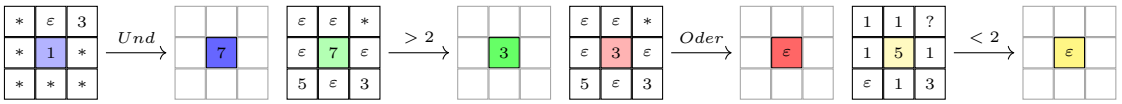
- *Und-Regeln* finden nur bei vollständiger Übereinstimmung Anwendung.
- *Oder-Regeln* finden Anwendung bei mindestens einer Übereinstimmung.
- *Größer-Als-Regeln* finden Anwendung bei mindestens n Übereinstimmungen, wobei n ein Parameter der Regel ist.
- *Kleiner-Als-Regeln* finden Anwendung bis zu n Übereinstimmungen.

Darüber hinaus kann der Kontext einer Regel Platzhalter enthalten, die entweder nur bei leeren Zellen eine Übereinstimmung erkennen (ε), bei beliebigen nicht leeren

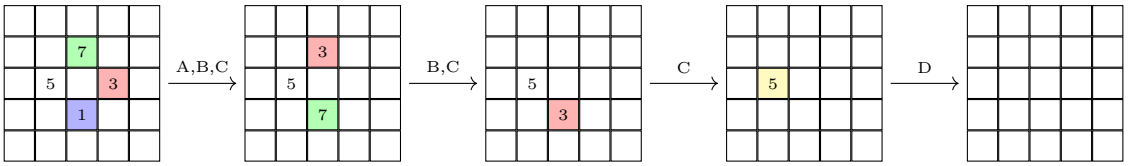
[32] Die Lernrate hat daher für uns keine weitere Bedeutung.

[33] Viele Verfahren die L-Systeme einsetzen verwenden hierfür meisst eine globale, maximale Iterationstiefe.

Zellen (?), oder aber bei jedem beliebigen Wert der Matrix, leere Zellen eingeschlossen (∗). Ein Beispiel für eine Anwendung der Regeln auf einer Zellmatrix ist in Abbildung 3.18e gegeben.



(a) Regel A **(b)** Regel B **(c)** Regel C **(d)** Regel D



(e) Entwicklung der Zellmatrix durch wiederholtes (paralleles) Anwenden der Regeln

Abbildung 3.18.: Beispiel für die vier verschiedenen Typen von Regeln, sowie der wiederholten Anwendung der Regeln auf einer Zellmatrix.

Die Autoren lassen den Leser leider über die genaue Funktionsweise des L-Systems im Unklaren. So wird nicht beschrieben, inwieweit alle Zell-Attribute (Neuronentyp, Quell- und Ziel-Duftstoff sowie Axon-Gewichtung) Einfluss auf das Anwenden (Matching) der Regeln haben, oder ob dabei nur der Neuronentyp sowie Quell- und Ziel-Duftstoff von entscheidender Bedeutung sind. Ein Matching auf der Axon-Gewichtung scheint indes kaum geeignet, da es sich hierbei um einen nicht-abzählbaren Fliesskomma-Wert handelt.

Zusammenfassend beschreibt Algorithmus 3 den Ablauf der Netzgenerierung und Bewertung. Dieser Schritt wird vom evolutionären Algorithmus für jedes Genom der Population ausgeführt.

Fazit

- Mit dem beschriebenen Verfahren können lediglich Feed-forward Netze erzeugt werden. Die Entwicklung von rekurrenten Netzen wurde von den Autoren nicht angedacht.
- Die Regeln des L-Systems haben eine feste Länge und Struktur (feste Nachbarschaftsbeziehung). Dies vereinfacht die genetischen Operatoren wie Mutation und Crossover.
- Die Autoren sehen die Hauptproblematik des Verfahrens in der langen Laufzeit des Algorithmus. In ihrem Experiment geben sie diese mit 1-2 Wochen auf 5 Sun Workstations und 2 Linux PCs (1997) an, um ein aus 7 Zellen (Neuronen) bestehendes Netz zur Lösung des XOR-Problems zu finden. Als Begründung geben die Autoren den hohen Aufwand an, der bei der Simulation des Auswachsen der Axone und Dendriten entsteht.

Algorithmus 3 Ein Schritt des evolutionären Algorithmus von Isto Aho et al.

1. Aus Genom wird L-System extrahiert.
2. Gewinnung einer Zell-Matrix durch wiederholtes Anwenden der Regeln des L-Systems bis keine Regel mehr anwendbar ist (Regeln veraltet).
3. Erzeugen der Neuronen anhand Zell-Matrix.
4. Erzeugen der Verbindungen zwischen den Neuronen anhand Duftmarkierungen der Zell-Matrix sowie Distanz der Neuronen innerhalb der Matrix zueinander.
5. Ermitteln der Eingabe-Neuronen (linker Rand der Zellmatrix).
6. Ermitteln der Ausgabe-Neuronen (rechter Rand der Zellmatrix).
7. Entfernen überflüssiger, nicht-verbundener Neuronen.
8. Bestimmung der Synapsen-Gewichte durch Anwenden des Backpropagation Lernalgorithmus unter Berücksichtigung der Axon-Gewichtung als Lernrate.
9. Simulation des Netzes und anschliessender Bewertung anhand Simulationsergebnis und Netzgröße.

- Die Autoren beschreiben leider nicht den genauen Algorithmus der für die Syanapsenbildung zwischen Dendriten und Axone verantwortlich ist.

3.4. Genaue Analyse der vorgestellten Verfahren

In diesem Abschnitt untersuchen wir die in Abschnitt 3.3 beschriebenen Verfahren anhand folgender Fragestellungen:

- Wie ist die Modularisierung in den einzelnen Verfahren realisiert?
- Wie wird eine Verdrahtung der Module erreicht?
- Können einzelne Parameter der Module variiert werden?
- Ist die Entwicklung der Neuronen an die Entwicklung der Synapsen gekoppelt?
- Wie werden die Ein- und Ausgänge des Netzes zum umgebenden System bestimmt?
- Wie gut lassen sich Beschreibungen von Hand erstellen?
- Wie effektiv arbeiten die genetischen Operatoren?
- Wie gut skaliert das Verfahen mit der Netzgrösse?
- Wie effizient (Laufzeit) ist die Netzgenerierung?

3.4.1. Modulbildung

Die Zellularen Kodierung nach Gruau erreicht die Modularität durch Verwendung mehrerer Instruktionsbäume, zwischen denen hin und her gewechselt werden kann[34].

Bei dem Verfahren, dass ein L-System zur Generierung von Instruktionen einer Graphgrammatik einsetzt, ist jede Regel des L-Systems als Modul anzusehen.

Die Erzeugung von Instruktionen einer Graphgrammatik durch ein Boolsches Genregulationsnetzwerk, lässt, auf den ersten Blick, keine klaren Modulgrenzen erkennen. Durch wenige, einfache Abhängigkeiten zwischen den Zuständen entsteht jedoch emergentes Verhalten, dass sich in Form von wiederkehrenden Mustern ausdrückt.

NEAT verzichtet ganz auf Modularisierung. HyperNEAT hingegen erreicht eine Modularisierung auf der Ebene der Gewichtung von Verbindungen (zwischen jeweils zwei beliebigen Knoten), indem symmetrische und periodische Funktionen miteinander verknüpft werden. Die Modularisierung bezieht Knoten jedoch nicht mit ein.

Das hierarchische Verfahren von Johannes Schmidt definiert explizit Module, die aus einem inneren Netzwerk aus Neuronen und Synapsen, als auch aus der Expression von Tochtermodulen bestehen.

Isto Aho et al. erreicht mit seinem Verfahren keine äusserlich klar erkennbare Modulbildung. Wir nehmen jedoch an, dass die Wechselwirkung einfacher Regeln des L-Systems emergentes Verhalten aufweisst, welches sich durch die Bildung wiederkehrenden Strukturen ausdrückt.

3.4.2. Ausprägung der Verbindungen zwischen Modulen

Ein Problem, dass sich durch die Bildung von Modulen ergibt, ist die Frage nach der Verdrahtung der Ein- und Ausgänge der Module mit ihrer Umgebung. Im folgenden wird angenommen, dass Module im allgemeinen mehrere Ein- bzw. Ausgänge aufweisen.

Gruau's Zellulare Kodierung ermöglicht die Adressierung einzelner Ein- und Ausgänge sowie Bereichen von Ein- und Ausgängen durch Angabe von deren Index, d.h. die Positionierung innhalb der modulspezifischen Liste der Ein- und Ausgänge. Dies macht die Verdrahtung extrem abhängig von der exakten Reihenfolge der Bildung der Verbindungen zu einem Modul. Allerdings entsteht durch die wiederholte Zellteilung mit anschliessender Selektion von Verbindungen eine gewisse Lokalität der Verbindungen, d.h. die Selektion von Verbindungen beschränkt sich immer nur auf eine Teilmenge aller Verbindungen des Netzes.

Bei der Graphgrammatik mit L-System können einzelne benachbarte Verbindungen oder Eingangsverbindungen des Elternknotens selektiert werden. Dies führt zu

[34] Es ensteht somit ein grob-granularer Instruktionsgraph mit Zyklen, wobei auch ein zyklenfreier Instruktionsgraph die Modularität erreichen würde.

dem gleichen Problem wie bei der Zellularen Kodierung. Die exakte Spezifikation einer Verbindung ist somit abhängig von der genauen Reihenfolge der Bildung der Verbindungen.

Die Erzeugung mittels Genregulationsnetzwerk beinhaltet einen Mechanismus, der benachbarte Zellen (Verbindungen im Zellgraphen mit identischem Quell- und Zielknoten) gegeneinander konkurrieren lässt. Die „Selektion“ einzelner Verbindungen auf andere Art und Weise ist nicht möglich.

HyperNEAT definiert für jedes beliebige Knotenpaar die Gewichtung der Verbindung. Da zudem jeder Knoten eine geometrische Position besitzt, lässt sich eine positionsabhängige Lokalität erreichen. Hiermit kann der Radius der Verbindungsbildung entsprechend eingeschränkt werden oder es liesse sich die Wahrscheinlichkeit der Verbindungsbildung in Abhängigkeit vom Abstand zweier Knoten setzen.

Bei dem Verfahren von Schmidt kodiert jedes Modul im Intronbereich einen speziellen Wert, durch den die Zufallsverteilung aller möglicher Verbindungen zum Elternmodul sowie zu den Kindmodulen gesteuert wird. Die Möglichkeit einer expliziten Verdrahtung von Ein- und Ausgänge über Modulgrenzen hinweg ist nicht gegeben.

Isto Aho et al. verwendet Markierungen bzw. Duftstoffe in den Neuronen um die Ausprägung der Verbindungen zu steuern. Hierbei entstehen Verbindungen nur zwischen ähnlich-riechenden Neuronen. Durch die Verwendung eines Zellgitters, wird zudem jedem Neuron eine diskrete Position zugewiesen, die dazu verwendet wird, ähnlich wie bei HyperNEAT, die Verbindungsbildung auf lokale Neuronen einzuschränken.

3.4.3. Variation von Modulen (Parameterisierung)

Das Verfahren von Gruau ermöglicht keine Parameterisierung der Module.

Eine direkte Variation der Modulparameter ermöglicht nur der Ansatz von Hornby und Pollack durch die Verwendung eines parametrischen L-Systems[35]. An die Regeln des L-Systems sind dabei Parameter geknüpft, die sowohl die strukturelle Ausprägung des Moduls steuern können, als auch die Gewichtung der Kanten. Allerdings erweist es sich als ziemlich schwer, eine geeignete parametrische Beschreibung von Modulen automatisiert (evolutionär) zu finden.

HyperNEAT erreicht durch die Komposition verschiedener mathematischer Funktionen[36] eine gewisse Form der Modularisierung, wobei Modularisierung in diesem Fall eher als das wiederholte Auftreten bestimmter Muster zu verstehen ist. Durch geeignete Komposition der Funktionen lassen sich beliebige Muster erzeugen und damit beliebige Parameter den einzelnen Verbindungen zuordnen. Im Allgemeinen lassen sich mit Compositional Pattern Producing Network (CPPN) beliebige Parameter an

[35] vgl. Beispiel 6 auf Seite 44
[36] Compositional Pattern Producing Network

Neuronen oder Synapsen knüpfen, wobei Neuronen und damit die Endpunkte der Synapsen durch ihre jeweilige geometrische Position bestimmt sind.

Um bei dem hierarchischen Verfahren von Schmidt die Variation eines Moduls zu erreichen, muss dieses zuerst kopiert werden. Anschliessend kann durch Mutation eine entsprechende Anpassung des Moduls erfolgen. Eventuelle funktionelle Zusammenhänge, wie z.B. zwischen einer äusseren Grösse wie der Frequenz, und den Parametern des neuronalen Netzes (z.B. der Verzögerung einer Synapse), können damit nicht effizient beschrieben werden und müssen somit für jedes zu variierende Modul neu (evolutionär) gefunden werden.

3.4.4. Kopplung von Neuronenbildung und Synapsenwachstum

Wie in der Natur zu beobachten ist, ist die Anzahl der synaptischen Verbindungen in biologischen Netzen wesentlich höher als die Zahl der Neuronen[37]. Dies legt die Schlussfolgerung nahe, dass auch bei künstlich erzeugten neuronalen Netzen ein signifikanter Unterschied im Verhältnis zwischen der Anzahl an Neuronen und Synapsen festzustellen ist. Eine direkte 1:1-Kopplung zwischen der Erzeugung von Neuronen und der Bildung von Synapsen erscheint aus diesem Grund nicht von Vorteil.

Bei Gruau's Zellularer Kodierung arbeiten die Operationen der Graphgrammatik auf den Knoten des Zellgraphen. Die Knoten haben dabei eine variable Anzahl an Ein- und Ausgangsverbindungen. Es existieren Operationen, die beispielsweise eine Verdopplung der Ein- und Ausgangsverbindungen bewirken (parallele Teilung) oder diese gleichmässig an einen neu erzeugten Knoten verteilen. Eine explizite Erzeugung einzelner Synapsen findet nicht statt. Gruau erreicht damit sehr leicht ein relativ hohes Verhältnis zwischen der Anzahl an Synapsen und Neuronen, ohne dass hierbei viele Operationen nötig sind.

Im Gegensatz zu Gruau's Verfahren muss bei der kantenorientierten Graphgrammatik von Hornby und Pollack jede einzelne Kante explizit erzeugt werden, d.h. die Anzahl der Verbindungen ist direkt an die Anzahl der ausgeführten Graphoperationen gekoppelt.

Anders ist dies bei der durch ein Genregulationsnetzwerk gesteuerten Graphgrammatik der Fall. Auch wenn hierbei eine kantenorientierte Graphgrammatik zum Einsatz kommt auf die prinzipiell das gleiche zutrifft wie bei dem Verfahren von Hornby und Pollack, so wird im Unterschied dazu der Zellgraph im Anschluss derart umgewandelt, sodass die Kanten des Zellgraphen die Knoten des späteren Netzwerkes repräsentieren. Damit wird ähnliches erreicht wie bei Gruau's Zellularer Kodierung.

Schmidt realisiert die Bildung von Synapsen durch die Weitergabe von Marken. Jede Marke wird dabei explizit erzeugt und ist fortan an ein bestimmtes Neuron gebunden. Folglich müssen n Marken erzeugt werden um n Synapsen zu bilden.

[37] Im menschlichen Gehirn kommen im Schnitt auf ein Neuron ca. 1000 synaptische Verbindungen.

Eine Möglichkeit der Vervielfältigung von Marken, um beispielsweise ein Neuron mit allen erzeugten Tochermodulen zu verbinden, besteht nicht.

Bei dem Verfahren von Isto Aho werden zunächst Neuronen gebildet, die in einer Zellmatrix platziert sind. Im Anschluss daran findet das Wachstum der Axone statt, gefolgt vom Wachstum der Dendriten. Dabei entstehen die Synapsen durch Kontakt der Dendriten mit den Axonen. Beeinflusst wird das Wachstum der Axone und Dendriten von Geruchsstoffen, mit denen die Neuronen markiert sind. Die Bildung der Synapsen ist daher nicht direkt an die der Neuronen gekoppelt, sondern indirekt durch deren Lage und Geruchsorientierung gegeben.

Bei HyperNEAT ist die Ausprägung von Synapsen an das Bild eines CPPN gekoppelt. Bereits ein sehr einfaches CPPN, wie etwa $f(x1, y1, x2, y2) = 1$ erreicht eine Vollverdrahtung aller Neuronen. Durch weitere Parameter kann die Lokalität der Synapsenbildung beeinflusst werden.

3.4.5. Bestimmung der Ein- und Ausgabeverbindungen des Netzes

Ziel der Bestimmung der Ein- und Ausgabeverbindungen ist es, den Austausch von Information zwischen dem umgebenden System und dem erzeugten Netz zu ermöglichen. Hierfür ist eine Zuordnung der Ein- und Ausgänge des umgebenden Systems zu den entsprechenden Ein- und Ausgängen des Netzes erforderlich.

Bei Gruau's Verfahren ist die Ursprungszelle (Zygote oder Axiom) mit allen Ein- und Ausgängen des umgebenden Systems verbunden. Durch fortgesetzte Zellteilung können diese Verbindungen teilweise oder vollständig an andere Zellen (auch mehrfach) weitergegeben werden. Ähnliches gilt für die Entwicklung eines Netzes durch das Genregulationsnetzwerk. Die Bestimmung der Ein- und Ausgabeverbindungen ist somit in beiden Fällen trivial.

Hornby und Pollack hingegen verwenden eine explizite Graphenoperation um ein Ausgangsneuron zu erzeugen. Eingangsneuronen oder Eingangsverbindungen können hingegen nicht explizit erzeugt werden und müssen daher auf andere Art und Weise gefunden werden. Dies kann beispielsweise durch eine Konvention erfolgen, oder durch Analyse der Knotengrade. Da in der Regel das umgebende System eine feste Anzahl an Ein- und Ausgängen vorgibt, benötigt dieses Verfahren zusätzlichen Aufwand um sich dem umgebenden System entsprechend anzupassen.

Schmidt löst das Problem durch Konvention, indem die erste Moduldefinition des Genoms die Eingänge des Netzes beschreibt, während die zweite Moduldefinition die Ausgänge beschreibt.

Isto Aho verwendet eine zweidimensionale Zellmatrix. Die Eingangsneuronen sucht er hierbei am linken äusseren Rand der Matrix, die Ausgangsneuronen entsprechend am rechten äusseren Rand. Die Anzahl der Ein- und Ausgänge ist somit durch die

Evolution der Zellmatrix bestimmt. Durch Beschränkung bzw. Ausweiten des Suchradius lässt sich die Anzahl jedoch entsprechend dem umgebenden System anpassen.

HyperNEAT erfordert die explizite Platzierung der Neuronen auf einem Substrat. Zudem ist die Zuordnung der Neuronen zu den Ein- und Ausgängen des umgebenden System vom Anwender fest vorgegeben, daher erübrigt sich eine weitere Bestimmung.

3.4.6. Effektivität des Evolutionären Algorithmus

Eine geeignete Wahl der Genetischen Operatoren beeinflusst massgeblich die Konvergenz des evolutionären Algorithmus.

NEAT ermöglicht die effektive Rekombination von Genomen ungleicher Topologie durch Festhalten des historischen Ursprungs der Gene. Entsprechendes trifft auf HyperNEAT zu, da es intern NEAT verwendet um ein CPPN zu entwickeln.

Gruau verwendet als Crossover-Operation das zufällige Austauschen von Teilbäumen zwischen den Genomen. Jedes Genom repräsentiert dabei eine möglicherweise völlig unterschiedliche Topologie. Eine topologische Zuordnung zwischen den Genomen findet nicht statt. Der Austausch von Teilbäumen erhält zwar einen Teil der Struktur, benötigt allerdings aufgrund der fehlenden topologischen Zuordnung viele Iterationen um gute Struktur zu finden.

Hornby und Pollack verwenden lineares 2-Punkt Crossover auf den Regeln des L-Systems. Aufgrund einer fehlenden topologischen Zuordnung trifft hier die gleiche Problematik zu wie bei Gruau's Verfahren.

Schmidt verwendet uniformes Crossover, wobei die Moduldefinitionen die Basiseinheiten des Austausches bilden. Schmidt geht davon aus, dass sich im Laufe der Evolution strukturell ähnliche Moduldefinitionen an gleicher Position im Genom platzieren. Entwickelt sich eine solche Ordnung nicht, so bewirkt seine Crossover-Operation eine zufällige Rekombination der einzelnen Moduldefinitionen, und verhindert somit eine effektive Evolution.

Das Verfahen von Isto Aho beruht auf der Emergenz von Struktur durch das Zusammenwirken einiger weniger, einfacher Regeln. Geht man von einer fest vorgegebenen Konfiguration der Zellmatrix aus, müssen lediglich wenige Regeln evolutionär gefunden werden. Somit reduziert sich entsprechend der Suchraum. Wird die Anfangskonfiguration der Zellmatrix ebenfalls evolutionär entwickelt, so geht dieser Vorteil verloren, und der Suchraum erhöht sich entsprechend. Gleiches trifft auf das Genregulationsnetzwerk zu. Über die Effektivität der genetischen Operatoren lässt sich in beiden Fällen, aufgrund des emergenten Verhaltens, keine Aussage treffen.

3.4.7. Skalierbarkeit der Netzgrösse

Es ist lediglich von HyperNEAT bekannt, dass damit Netze mit mehreren Millionen Synapsen erfolgreich erzeugt wurden. Dabei ist von Vorteil, dass ein CPPN grundsätzlich eine unendliche räumliche Auflösung besitzt, die Netzgrösse daher nur vom Sampling (Platzierung der Neuronen im Substrat) abhängig ist. Alle weiteren Verfahren wurden ausschliesslich für die Entwicklung sehr kleiner Netze eingesetzt.

Theoretisch skalieren jedoch die Verfahren die eine indirekte Kodierung verwenden mit der Netzgrösse. Das parametrische L-System von Horny und Pollack erlaubt beispielsweise die parametrische Beschreibung der Topologie und Gewichtung von Subnetzen. Die Schwierigkeit besteht nur darin, eine geeignete Beschreibung für diese Subnetze in Form von Regeln des L-Systems zu finden. Dies gestaltet sich als schwierig, weil zum einen der Suchraum aufgrund der komplexen Beschreibungsmöglichkeit sehr gross ist, zum anderen die genetischen Operatoren wenig effektiv arbeiten. Diese Problematik betrifft in gleicher Weise die Zellulare Kodierung von Gruau und auch Schmidt's Verfahren.

3.4.8. Laufzeit der Netzgenerierung

Isto Aho bemängelt an seinem Verfahren die sehr lange Laufzeit. Dies liegt an der verwendeten Methode für die Ausprägung der Axone und Dendriten.

HyperNEAT hat den Vorteil, dass ein CPPN grundsätzlich eine unendliche räumliche Auflösung besitzt, unabhängig von seiner Komplexität. Somit ist die Laufzeit lediglich abhängig vom Sampling, d.h. der Anzahl verwendeter Neuronen. Das Sampling von HyperNEAT lässt sich zudem auf triviale Art und Weise massiv parallelisieren.

Bei den anderen Verfahren ist die Laufzeit stark abhängig vom Genom. Genauere Angaben zur Laufzeit fehlen jedoch. Unsere eigene Implementierung der Methode von Hornby und Pollack (L-System), sowie des Verfahrens auf Grundlage eines Genregulationsnetzwerkes zeigen, dass zumindest bei den von uns getesteten Netzgrössen, die Laufzeit zur Generierung des Netzes im Verhältnis zur Berechnung der Fitness vergleichsweise gering ist.

3.5. Fazit

Ein schlüssiges Konzept zur effektiven genetischen Rekombination bietet lediglich NEAT an. Dies bildet die Grundlage von HyperNEAT und wird von diesem um die Fähigkeit erweitert, auch grosse Netze mit wiederkehrenden Mustern und Variation erzeugen zu können. Hiermit können effektiv sowohl Topologie als auch Gewichtung des Netzes erzeugt werden. Erreicht wird dies indem ein Compositional Pattern Producing Network zum Einsatz kommt, welches darüber hinaus bei Bedarf das

Einstellen beliebiger zusätzlicher Parametern erlaubt. Die räumliche Positionierung der Neuronen ermöglicht zudem die Lokalität von Verbindungen.

Zwar erlaubt das Verfahren von Hornby und Pollack eine effiziente Beschreibung aller Aspekte von Teilnetzen durch ein parametrisches L-System, allerdings können aufgrund der geringen Effektivität der verwendeten Crossover-Operation zur Rekombination zweier Genome, sowie dem hohen Suchraum, der aufgrund der Komplexität der verwendeten Kodierung entsteht, nur schwer solche Netze gefunden werden.

Dem Verfahren von Schmidt mangelt es ebenso an einer geeigneten genetischen Crossover-Operation. Die innerhalb der Module definierte zufallsgesteuerte intermodulare Verbindung von Neuronen erlaubt zudem weder die Ausprägung von regelmässigen Verbindungsstrukturen noch von Regelmässigkeiten in deren Gewichtung. Regelmäßigkeiten können allerdings durch Kopieren ganzer Moduldefinitionen erzeugt werden.

Die Zellulare Kodierung von Gruau leidet an derselben Problematik wie die Kodierung von Hornby und Pollack. Hinzu kommt, dass die Verbindungsparameter nicht effizient variiert werden können.

Das Verfahren von Isto Aho beruht auf dem emergenten Zusammenspiel einiger weniger Regeln. Gleiches trifft auf das Verfahren zu, welches ein Genregulationsnetzwerk in Form eines Boolschen Netzwerkes zur Steuerung einer Graphgrammatik einsetzt. Die praktische Eignung beider Verfahren lässt sich aufgrund des komplexen Verhaltens nur experimentell ermitteln. Es ist allerdings anzunehmen, dass komplexes Verhalten bereits durch wenige Regeln erzeugt werden kann, wie beispielsweise Stephen Wolfram eindrucksvoll in [Wol02] unter Beweis stellt.

4. Prototypen und Ideen

4.1. Verbindungsbildung durch Diffusionsbegrenztes Wachstum

Diffusionsbegrenztes Wachstum (engl. Diffusion Limited Aggregation abk. DLA) wurde erstmals 1981 von den Physikern Thomas Witten und Leonard Sander [WS81] beschrieben. Bei dem in Gasen und Flüssigkeiten auftretenden Prozess, bilden Moleküle, die durch Brownsche Bewegung (Diffusion) durch den Raum wandern, Ablagerungen, sobald diese auf ruhende Moleküle treffen. Die sich so ausbildenden Stoffablagerungen haben Ähnlichkeiten zu bekannten makroskopischen fraktalen Strukturen wie etwa die Verästelung bei Bäumen. Mit der Computer-gestützten Simulation lassen sich zudem viele natürliche Prozesse wie z.B. elektrische Entladungsvorgänge in Gasen [NPW84], das fraktale Wachstum von elektrochemischen Beschichtungen [HL90; BB84] oder das Wachstum von Schneeflocken [MMF87] simulieren. Dass sich auch biologische Strukturen mit Diffusionsbegrenztem Wachstum nachbilden lassen zeigen [Luc06] und [XM07]. So werden in [Luc06] dreidimensionale Modelle neuronaler Dendritenbäume verschiedenster Form generiert, die dem natürlichen Vorbild sehr nahe kommen. Die *Fragestellung* die sich für uns daraus ergeben hat ist, ob sich Diffusionsbegrenztes Wachstum prinzipiell auch für die Verbindungsbildung bei gepulsten neuronalen Netzen eignet, und wenn ja, wie gut und mit welchem Aufwand.

4.1.1. Simulation von Diffusionsbegrenztem Wachstum

Die Simulation von Diffusionsbegrenztem Wachstum ist trivial. Eine einfache sequentielle Implementierung ist Algorithmus 4 zu entnehmen. Hierbei wandert während eines Iterationsschrittes jeweils ein einzelner Partikel zufällig durch den Raum, bis dieser in seiner direkten Nachbarschaft auf einen bereits zur Ruhe gekommenen Partikel trifft, woraufhin beide Partikel ein Aggregat bilden, der bewegte Partikel somit auch zur Ruhe kommt. Abbildung 4.1 zeigt das Ergebnis einer solchen Simulation nach 5000 Iterationsschritten[1]. Ausgangspunkt hierbei war ein einzelner, in der Mitte des zweidimensionalen Raumes platzierter Partikel.

[1] $SimulateDLA(400, 300, 1, 5000, \{(200, 150)\})$

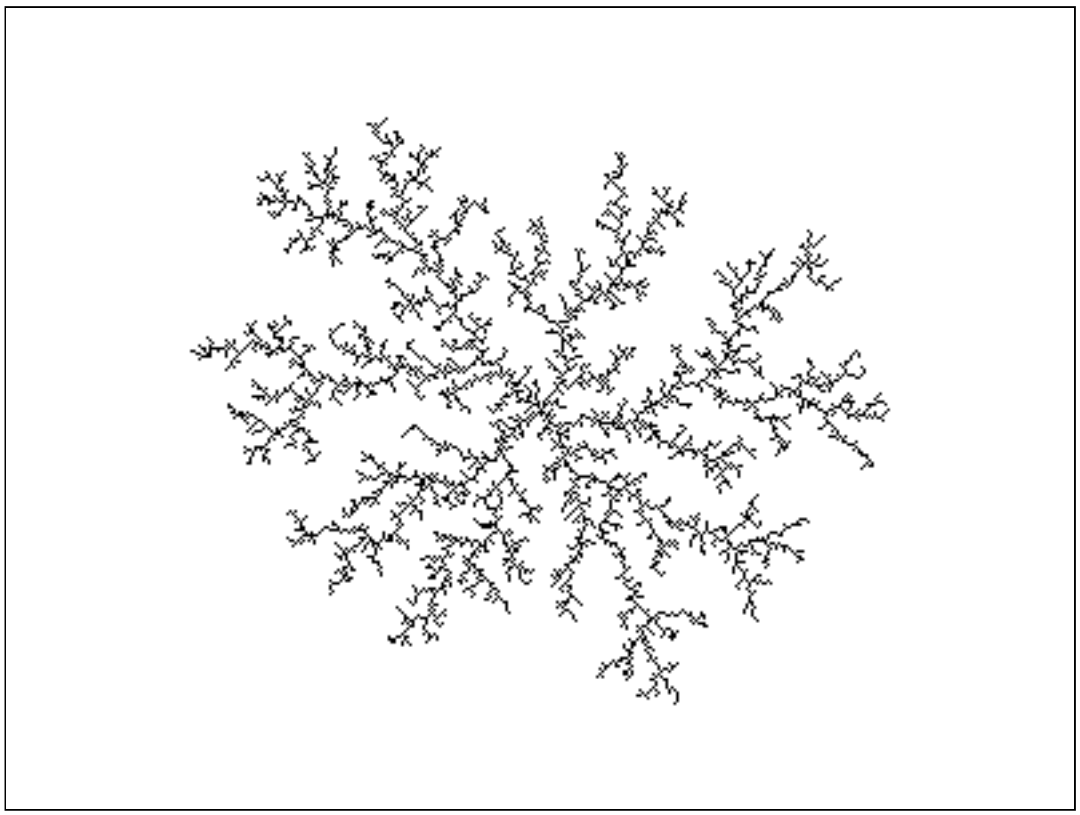


Abbildung 4.1.: Simulation von Diffusionsbegrenztem Wachstum. Iteration 5000.

Neben der sequentiellen Simulation lässt sich Diffusionsbegrenztes Wachstum auch massiv parallel durch den Einsatz randomisierter Zellularautomaten simulieren. Hierbei werden viele Partikel zur gleichen Zeit im Raum bewegt und auf Kontakt mit dem Aggregat getestet. Bei dieser Methode ist allerdings darauf zu achten, einen Iterationszyklus des zellularen Automaten derart in zwei Teilschritte zu zerlegen, sodass jeder Teilschritt nur Zellen ohne Überschneidungen in ihren Nachbarschaften bearbeitet. Ansonsten führt dies dazu, dass benachbarte Zellen einen Partikel mit sehr hoher Wahrscheinlichkeit verdoppeln.

4.1.2. Modellierung neuronaler Dendritenbäume

In [Luc06] verwenden die Autoren Diffusionsbegrenztes Wachstum um das Wachstum neuronaler Dendritenbäumen nachzubilden. Das entwickelte Modell ist in der Lage, anhand folgender Parameter, die Morphologie verschiedener biologischer Neuronentypen nachzubilden:

- Ausdehnung des Wachstumsraumes
- Zeitspanne des Prunings (Entfernung aussenliegender Partikel)
- Räumliche Konzentration *neurotropher* Partikel

Das vorgestellte Verfahren verwendet die parallele Simulation vieler gleichzeitig durch den Raum schwebender Partikel. Die Anzahl der Partikel ist hierbei durch die Ausdehnung des Raumes und die Stoffkonzentration gegeben. Um die Verästelung der sich auszubildenden Dendriten einzuschränken, werden terminale Partikel durch regelmässiges *Pruning* innerhalb einer bestimmten Zeitspanne zufällig von den Aussenkanten des Aggregats entfernt.

Während vorherige Veröffentlichungen die Gestalt von Neuronen meißt durch viele Neuronentyp-spezifische Parameter beschreiben, wie z.B. Teilungswahrscheinlichkeit der Verästelung, Orientierung oder Anzahl an Segmenten, so zeigen die Autoren mit dieser Veröffentlichung, dass komplexe Dentriten-ähnliche Strukturen auch ohne die Kenntniss des genauen intrinsischen Plans von deren Geometrie erzeugt werden können. Hierzu reichen wenige externe Umgebungsfaktoren wie z.B. die Partikelstoffkonzentration, Wettbewerb zwischen den Partikeln, sowie die Begrenzung des Raumes aus, um im Zusammenspiel mit den einfachen Regeln des diffusiven Wachstums räumliche Bilder verschiedener Neuronentypen des Cortex nachzubilden.

4.1.3. Ausprägung von Verbindungenstrukturen

Die Modellierung in [Luc06] beschränkt sich auf die von einzelnen Neuronen ausgehende Bildung von Dendriten. Der synaptische Kontakt zu Axonen anderer Neuronen ist nicht vorgesehen. Dies ist jedoch erforderlich um Verbindungsstrukturen neuronaler Netze erzeugen zu können.

Abbildung 4.2 zeigt das Ergebnis unseres Versuches, die Verbindungsstruktur eines Jeffress-Netzes mit zwei Eingängen und drei Ausgängen nachzubilden. Hierzu wurden zwei Neuronen am oberen Rand und drei am unteren Rand platziert. Sehr schön lässt sich das Wachstum der von den Neuronen ausgehenden Dendritenbäume erkennen, die allerdings im Laufe der Simulation mehr und mehr im Chaos untergehen. Abbildung 4.3 zeigt denselben Versuch mit aktiviertem Pruning. Dies führt zu einer starken Einschränkung des Verzweigungsgrades, allerdings wird dadurch auch die Wachstumsgeschwindigkeit erheblich reduziert.

Aufgrund der sehr langen Simulationszeiten haben wir an dieser Stelle von weiteren Versuchen abgesehen. Zudem ergibt sich im zweidimensionalen Raum ein weiteres Problem: Überkreuzverbindungen, z.B. zwischen den beiden äusseren Eingabe- und Ausgabeneuronen, sind hier nicht möglich. Eine Simulation im dreidimensionalen Raum würde dies zwar erlauben, geht allerdings zu Lasten noch längerer Laufzeiten und einem noch höheren Speicherbedarf.

4.1.4. Fazit

Zwar ist die Ausbildung der Verbindungsstrukturen gepulster neuronaler Netze durch die Simulation von Diffusionsbegrenztem Wachstum theoretisch denkbar. Aufgrund

des sehr hohen Rechen- und Speicheraufwandes jedoch mit den uns zur Verfügung stehenden Mitteln nicht praktikabel[2]. Zudem sind bei der beschriebenen Methode alle Neuronen derselben Stoffkonzentration ausgesetzt. Dies führt zum Verlust der Differenzierungsmöglichkeit einzelner Verbindungen. Will man dennoch eine solche Differenzierung realisieren, so müsste sowohl die Partikelstoffkonzentration als auch die Einstellungen des Prunings in einzelnen Raumsegmenten variiert werden. Weiterhin lässt sich die Wiederholbarkeit der Simulation nur durch das Festhalten des vom Pseudozufallszahlengenerator verwendeten Startwertes erreichen. Die Verwendung einer echten Zufallsquelle (bzw. durch Reseeding des Zufallszahlengenerators) ermöglicht hingegen keine Wiederholbarkeit.

4.2. Verbindungsbildung durch Space Colonization

Space Colonization ist ein Verfahren, dass in der Computergraphik eingesetzt wird um realitätsnahe Bilder von Bäumen anzufertigen [RLP07]. Im Unterschied zur Modellierung von Bäumen mithilfe von Lindenmayer-Systemen, spielt der Wettbewerb um Raum bei Space Colonization eine dominierende Rolle. Abbildung 4.4 zeigt die grundlegende Funktionsweise des Verfahrens. Ein Baum ist hierbei durch die Menge seiner Knoten repräsentiert. Das Wachstum des Baumes wird durch fest im Raum platzierte Attraktoren beeinflusst. Während ein Attraktor jeweils nur einen Knoten beeinflussen kann, und zwar den innerhalb seines Einflussradius d_i am nächsten liegenden Knoten, so kann andererseits ein Knoten des Baumes von mehreren Attraktoren gleichzeitig beeinflusst werden. Wird ein Knoten beeinflusst, so erfährt dieser eine Kraft in Richtung der normierten Summe seiner beeinflussenden Attraktoren (4.4d). Als Folge entsteht ein neuer Knoten im Abstand d_m (4.4e). Nicht dargestellt ist, dass Attraktoren gelöscht werden, sobald ein Knoten in dessen Nähe kommt ($< d_k$). Algorithmus 5 beschreibt den genauen Ablauf einer einzelnen Iteration des Verfahrens.

Unterscheidung zu Diffusionsbegrenztem Wachstum Während beim Diffusionsbegrenztem Wachstum (siehe Abschnitt 4.1) einzelne Partikel zufällig durch den Raum wandern bis diese mit ruhenden Partikeln in Kontakt kommen und ein Aggregat bilden, so sind bei Space Colonization die Attraktoren fest im Raum platziert und verändern während ihrer gesamten Lebensdauer ihre Position nicht. Auch die Knoten (des Baumes) verändern ihre Position nicht, aus ihnen gehen lediglich neue Triebknoten hervor. Ist eine initiale Konfiguration bestehend aus Attraktoren und Knoten gegeben, so handelt es sich bei Space Colonization demnach um ein rein deterministisches Verfahren, während DLA fortlaufend randomisiert wird. Ein weiterer Unterschied besteht in der Repräsentation des Raumes. DLA verwendet hierfür

[2] In [Luc06] sind 51 Sekunden für die Berechnung eines 30×30×30 Raumes angegeben, 2906 Sekunden für einen Raum mit 60×60×60 Partikeln. Unsere eigene Berechnung spielen sich im selben Bereich ab.

Algorithmus 4 Simulation von Diffusionsbegrenztem Wachstum

```
1: procedure SimulateDLA(width, height, walkDistance, iterations, seeds)
2:     for x ← 1, width do
3:         for y ← 1, height do
4:             Space(x, y) ← 0                    ▷ Initialisiere zweidimensionalen Raum
5:         end for
6:     end for
7:     for all (x, y) ∈ seeds do
8:         Space(x, y) ← 1                        ▷ Setze Startkonfiguration
9:     end for
10:    for i ← 1, iterations do                   ▷ Random-walk eines Partikels
11:        p_x ← random(1, width)                 ▷ Zufällige Startposition
12:        p_y ← random(1, height)
13:        while Neighborhood(Space, p_x, p_y) = 0 do   ▷ solange ohne Kontakt
14:            d_x ← random(−walkDistance, walkDistance)
15:            d_y ← random(−walkDistance, walkDistance)
16:            p_x ← clip(p_x + d_x, width)       ▷ bewege Partikel
17:            p_y ← clip(p_y + d_y, height)
18:        end while
19:        Space(p_x, p_y) ← 1                    ▷ Parikel bildet Aggregat
20:    end for
21: end procedure
22: procedure Neighborhood(Space, x, y)           ▷ Moore-Nachbarschaft
23:    N_1 ← Space(x − 1, y − 1) + Space(x + 0, y − 1) + Space(x + 1, y − 1)
24:    N_2 ← Space(x − 1, y + 0) + Space(x + 0, y + 0) + Space(x + 1, y + 0)
25:    N_3 ← Space(x − 1, y + 1) + Space(x + 0, y + 1) + Space(x + 1, y + 1)
26:    returnN_1 + N_2 + N_3
27: end procedure
28: procedure Clip(pos, high)
29:    return max(min(pos, high), 1)
30: end procedure
```

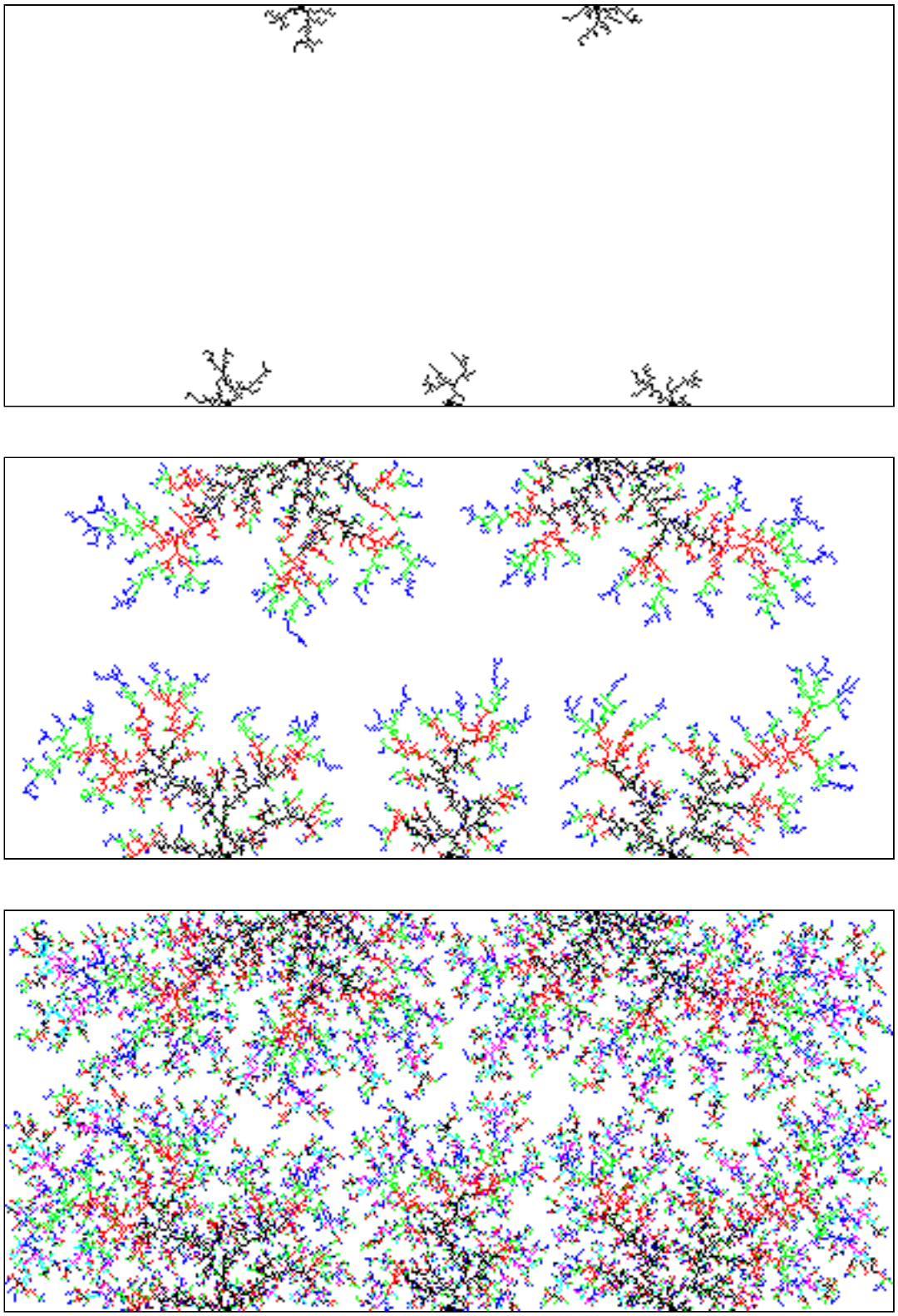


Abbildung 4.2.: Simulation von Diffusionsbegrenztem Wachstum mit fünf Keimzellen. Iterationen: 500, 8000, 20000 (von oben nach unten).

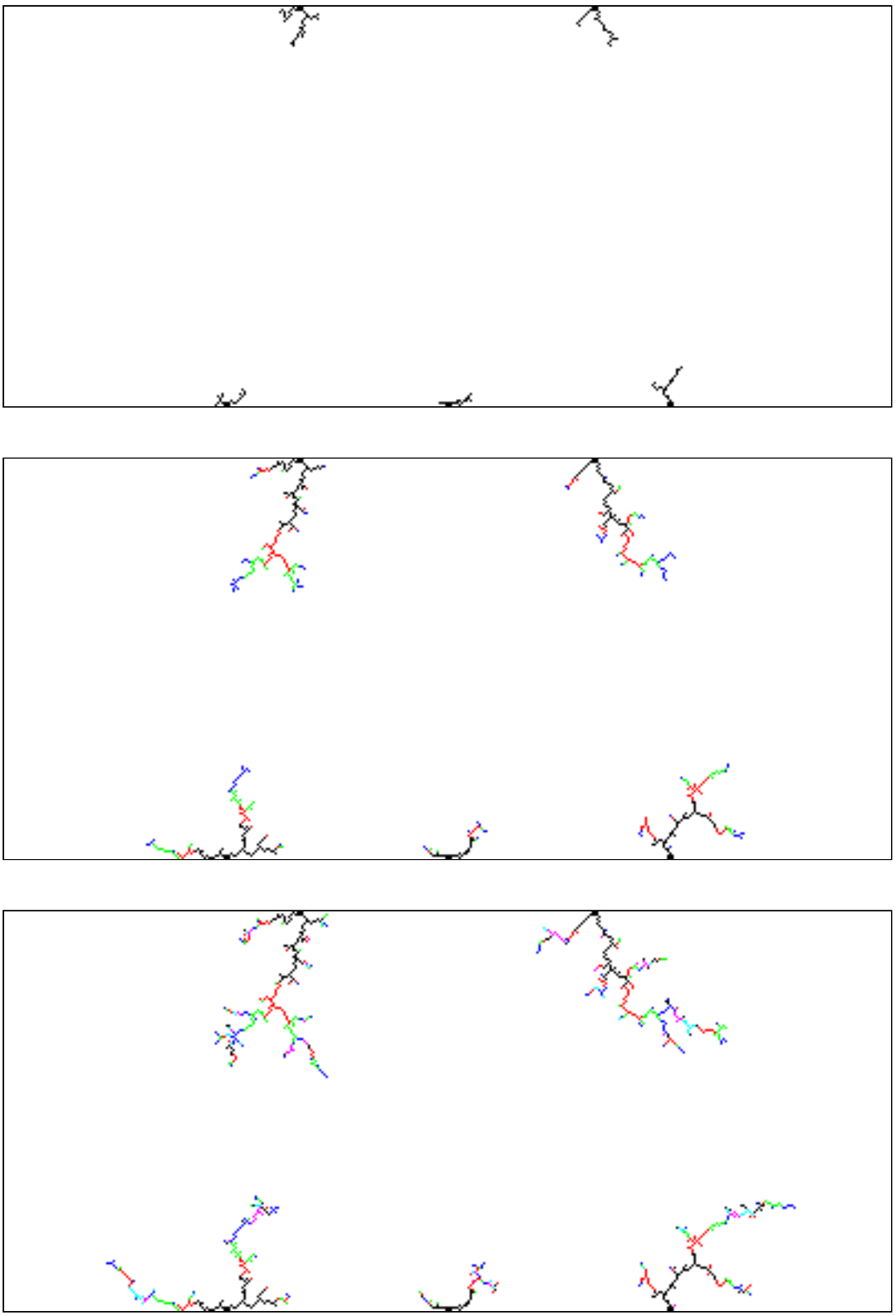


Abbildung 4.3.: Simulation von Diffusionsbegrenztem Wachstum mit fünf Keimzellen und *Pruning* (Intervall: 10, Zeitfenster: 100, Wahrscheinlichkeit: 40%). Iterationen: 500, 8000, 20000 (von oben nach unten).

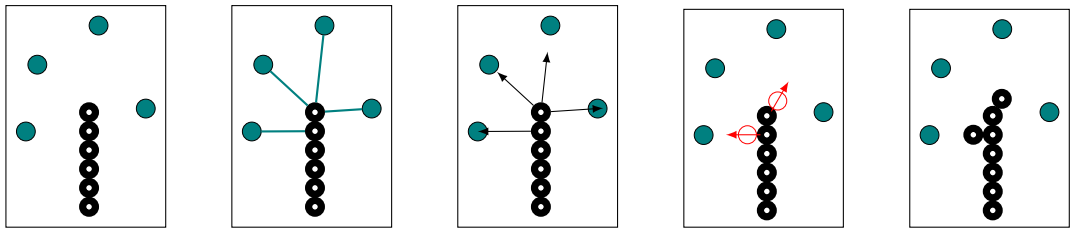


Abbildung 4.4.: Space Colonization Algorithmus. Schwarz: Knoten des Baumes. Grün: Attraktoren. **a)** Anfangskonfiguration. **b)** Bestimme für jeden Attraktor den jeweils nächsten Knoten. **c)** Berechne normierte Distanz der Anziehungskräfte. **d)** Berechne resultierende Kraft und multipliziere mit Verschiebedistanz d_m. **e)** Füge neu erzeugte Knoten hinzu.

Algorithmus 5 Space Colonization Algorithmus

procedure SpaceColonization($nodes$, $attractors$, d_m, d_i, d_k)
  for all $\vec{v} \in nodes$ **do**
    $S(\vec{v}) \leftarrow \varnothing$
  end for
  for all $\vec{a} \in attractors$ **do**
    Finde Knoten $\vec{v} \in nodes$ der am nächsten ist zu $\vec{a}$ ▷ 4.4b
    if $||\vec{v}$ - $\vec{a}|| < d_i$ **then** ▷ Im Einflussbereich?
      $S(\vec{v}) \leftarrow S(\vec{v}) \cup \{\vec{a}\}$
    end if
  end for
  $newNodes \leftarrow \varnothing$
  for all $\vec{v} \in nodes$ **do** ▷ Berechne neu entstehende Knoten
    if $S(\vec{v}) \neq \varnothing$ **then**
      $\vec{v_n} \leftarrow \vec{v} + d_m \cdot \sum_{\vec{s} \in S(\vec{v})} \frac{\vec{s}-\vec{v}}{||\vec{s}-\vec{v}||}$ ▷ Abbildungen 4.4c und 4.4d
      $newNodes \leftarrow newNodes \cup \{\vec{v_n}\}$
      for all $\vec{a} \in attractors$ **do**
        if $||\vec{v_n}$ - $\vec{a}|| < d_k$ **then** ▷ Neuer Knoten innerhalb Löschradius?
          $attractors \leftarrow attractors \setminus \{\vec{a}\}$ ▷ Entferne Attraktor
        end if
      end for
    end if
  end for
  $nodes \leftarrow nodes \cup newNodes$ ▷ Füge neue Knoten hinzu (4.4e)
end procedure

meisst einen diskreten Raum, während der Raum bei Space Colonization kontinuierlich ist und indirekt durch die Positionung der Attraktoren und Knoten aufgespannt wird. Dies erlaubt eine effizientere Speicherung, allerdings auf Kosten der Laufzeit.

Komplexität der Laufzeit Eine naive Implementierung hat eine Komplexität (pro Iterationszyklus) von $\mathcal{O}(N \cdot A)$, mit N Anzahl der Knoten und A Anzahl der Attraktoren, da für jeden Attraktor der jeweils nächstgelegene Knotenpunkt gefunden werden muss. Die Anzahl der Knotenpunkte ist hierbei im Laufe der Simulation monoton steigend, die der Attraktoren monoton fallend. Durch raumteilende Verfahren[3] lässt sich der Aufwand jedoch entsprechend reduzieren.

4.2.1. Ausprägung von Verbindungenstrukturen

Wir haben versucht, Verbindungsstrukturen von gepulsten neuronalen Netzen durch Space Colonization nachzubilden. Um zuerst ein Gefühl für die Auswirkung einzelner Parameter zu bekommen, haben wir diese, wie in Abbildung 4.5 gezeigt, variiert. Einzustellende Parameter sind hierbei:

- Anzahl Attraktorpunkte und deren Positionierung
- Maximaler Verzweigungsgrad $maxBranches$
- Löschradius d_k
- Einflussradius d_i
- Bewegungslänge d_m

Wählt man den Einflussradius d_i zu klein, so bleibt das Knotenwachstum in Bereichen geringer Attraktorendichte stehen. Ist d_i dagegen unendlich, so garantiert dies, dass zu jedem Attraktorpunkt eine Verbindung auswächst. In den Beispielen haben wir die Positionierung der Attraktoren zufällig gewählt. Will man hingegen Netze evolutionär erzeugen, so bietet es sich an, die Positionung der Attraktoren im Genom zu kodieren und entsprechend zu optimieren. Weiterhin muss die Platzierung der Neuronen vorgegeben werden.

Abbildung 4.6 zeigt das Resultat eines Experimentes, in dem wir versucht haben, die Verbindungsbildung eines Jeffress-Netzes mit 2 Eingängen und 3 Ausgängen nachzubilden. Hierzu haben wir die Eingangsneuronen (oben, schwarz) als Knoten platziert, während die Ausgangsneuronen als spezielle Attraktoren realisiert sind (unten, rot). Die Eingangsneuronen sollen sich so den Weg zu den Ausgangsneuronen suchen. Kommt es zu einem Kontakt mit einem speziellen Attraktor (infolge blau eingefärbt), so entsteht eine Verbindung zwischen dem Ursprungsneuron und dem durch den Attraktor repräsentierten Neuron. Die Gewichtung dieser Verbindung ist durch die Pfadlänge gegeben.

[3] Quadtree (2d) bzw. Octtree (3d)

4.2.2. Fazit

Wir haben gezeigt, dass es prinzipiell mit Space Colonization möglich ist, Verbindungsstrukturen von gepulsten neuronalen Netzen zu erzeugen. Allerdings sind Überkreuzverbindungen ähnlich wie bei Diffusionsbegrenztem Wachstum im zweidimensionalen Raum schwer zu realisieren, da die Knoten gegenseitig um den Raum konkurrieren. Durch Simulation im dreidimensionalen Raum lässt sich dieses Problem jedoch beheben. Die Laufzeit des Versuches in Abbildung 4.6 beträgt ungefähr 1 Sekunde (100 Attraktoren, 50 Iterationen). Für die evolutionäre Verbindungsbildung grösserer Netze halten wir deshalb das Verfahren in der dargestellten Form für nicht konkurrenzfähig.

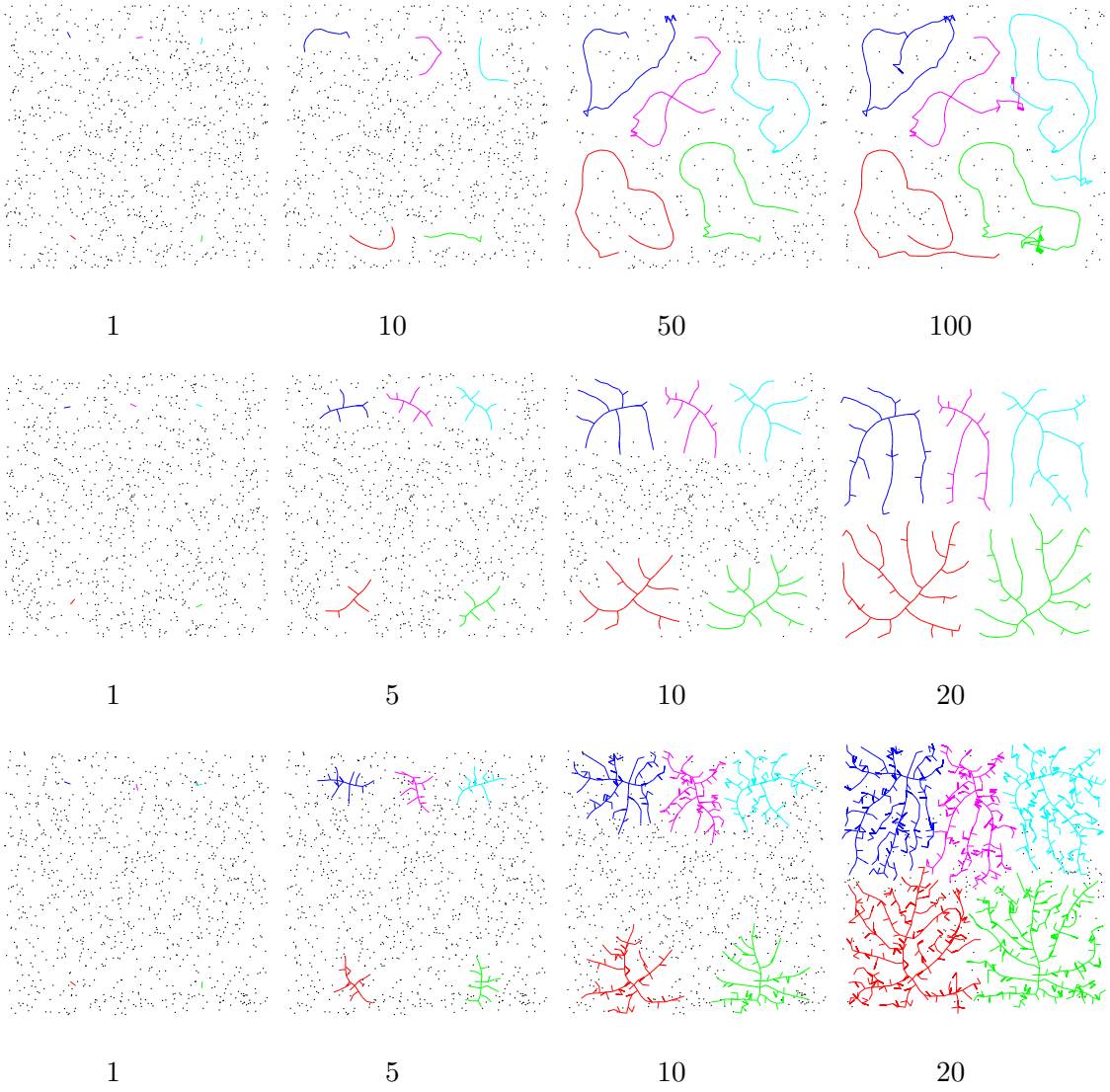


Abbildung 4.5.: Simulation von Space Colonization mit unterschiedlicher Parameterisierung. Jeweils 1000 Attraktorpunkte und $d_m = 0.05$, $d_i = 0.25$. Iteration im Untertitel. **oben)** $maxBranches = 1$, $d_k = 0.1$. **mitte)** $maxBranches = 2$, $d_k = 0.1$. **unten)** $maxBranches = 10$, $d_k = 0.02$.

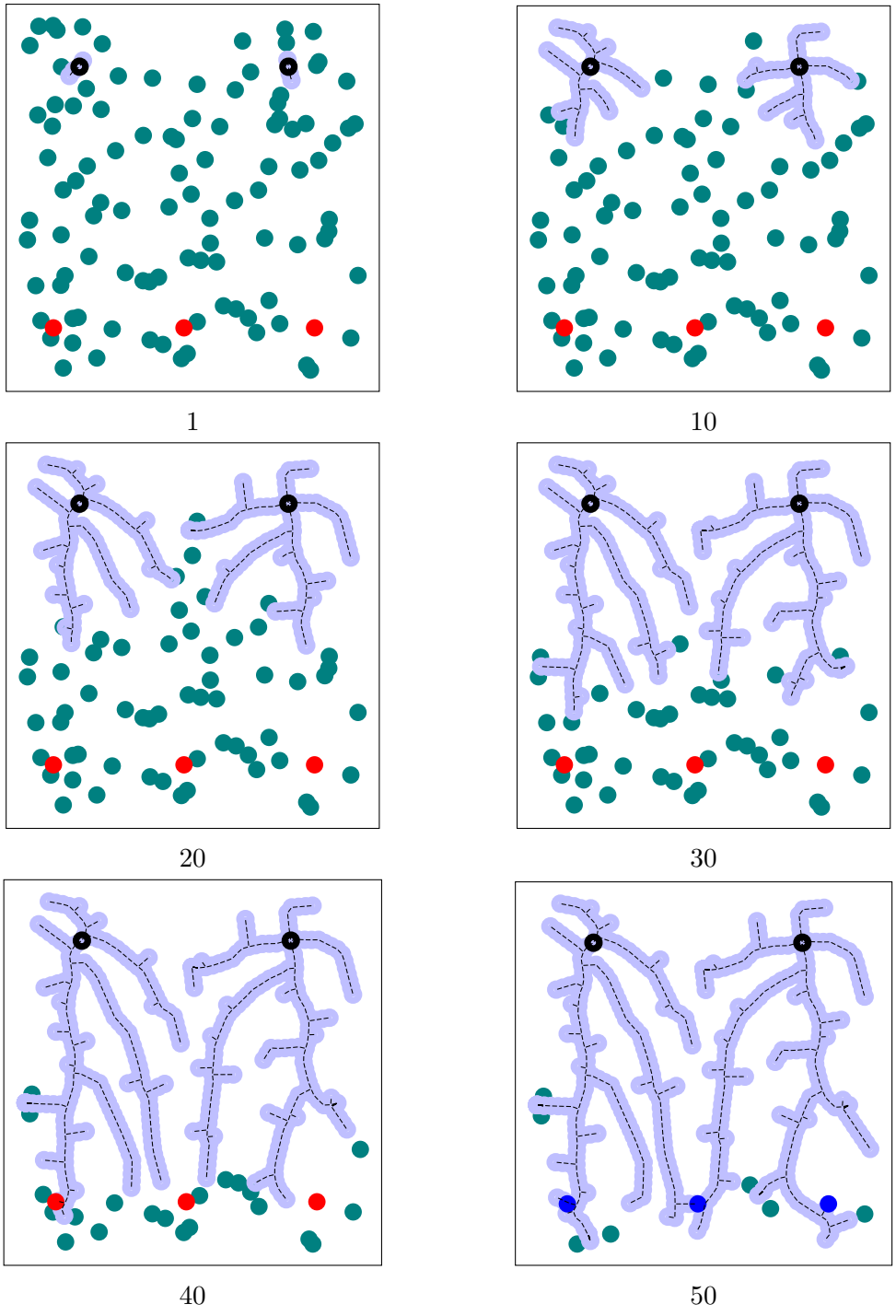


Abbildung 4.6.: Ausbildung von Verbindungen mit Space Colonization. Iteration im Untertitel. Ursprungsneuronen (schwarz). Zielneuronen (rot). Verbundende Zielneuronen (blau). Attraktoren (grün). Zielneuronen sind Attraktoren. Einstellung: $numPoints = 100$, $d_m = 0.04$, $d_k = 0.05$, $d_i = 0.5$.

5. Entwicklung eines Verfahrens zur Evolutionären Optimierung von Gepulsten Neuronalen Netzen

In dem vorliegenden Kapitel bauen wir auf den in Kapitel 3 gesammelten Erfahrungen über die dort behandelten Erzeugungsalgorithmen auf, und beschreiben das von uns ausgewählte und entsprechend angepasste Verfahren, welches primär für die evolutionäre Optimierung von gepulsten neuronalen Netzen eingesetzt werden soll.

Um bestimmte Eigenschaften des Verfahrens besser veranschaulichen und überprüfen zu können, wie etwa dessen Fähigkeit zur Erzeugung modularer Strukturen, haben wir das Verfahren zudem derart flexibel gehalten, sodass neben dem Spezialfall der gepulsten neuronalen Netzen auch beliebige weitere Graphen erzeugt werden können. Die Verwendung der Graphenähnlichkeit zu einem gegebenen Zielgraph als Fitnessfunktion versetzt uns hierbei in die Lage, das Verfahren nach klar definierten Strukturen suchen zu lassen. Ein weiterer Grund für dieses Vorgehen ist es, mögliche Fehler in der Konfiguration und in der genauen Durchführung der Simulation der gepulsten neuronalen Netze ausschliessen zu können. Ohne dieses Vorgehen wären wir nicht in der Lage, mögliche Probleme eindeutig dem Erzeugungsalgorithmus oder der Konfiguration der Simulation zuordnen zu können.

In Abschnitt 5.1 gehen wir auf die von uns gestellten Anforderungen an das Verfahren näher ein. Im Anschluss analysieren wir in Abschnitt 5.2 einige in Frage kommende Verfahren und begründen warum wir uns für welche Auswahl entschieden haben. In Abschnitt 5.3 beschreiben wir schliesslich das von uns entwickelte Verfahren. Eine Auswertung über die Performanz des Verfahrens findet sich in Kapitel 6.

5.1. Anforderungen

Wie dem Titel dieser Arbeit zu entnehmen ist, besteht unser Ziel in der *Verbesserung eines hierarchischen evolutionären Algorithmus* für den Einsatz im Bereich des Maschinellen Lernens. In unserem konkreten Fall bedeutet dies, dass der Algorithmus für die Optimierung der Topologie und Gewichtung von gepulsten neuronalen Netzen eingesetzt werden soll. Die evolutionäre Optimierung ist insbesondere bei den gepulsten neuronalen Netzen von Bedeutung, da anders als etwa bei den künstli-

chen neuronalen Netzen keine allgemeingültigen Algorithmen[1] für das Einlernen der Gewichtung zur Verfügung stehen. Zwar eignet sich Hebbsches Lernen oder Spike-Timing Dependent Plasticity (STDP) zur Optimierung der Syanpseneffizienz, die Verzögerungszeit der Synapsen lässt sich jedoch nicht einstellen. Daher ist die Vollverdrahtung im Falle der gepulsten neuronalen Netzen nicht praktikabel und hätte zudem negative Auswirkungen auf die Performanz.

Auf die Anforderungen, die wir an das zu entwickelnde Verfahren stellen, gehen wir in den folgenden Abschnitten ein.

5.1.1. Beherrschbarkeit von komplexen Netzen

Eine der Hauptproblematiken bei der evolutionären Optimierung mit der wir uns hier auseinandersetzen, ist die Beherrschbarkeit von komplexen Netzen. Während bei einem direkt kodierenden Verfahren beispielsweise der Suchraum bei kleinen Netzen noch überschaubar und beherrschbar ist, so steigt dieser nicht polynomiell mit der Netzgrösse an und führt zu einer *Explosion des Suchraumes*. Komplexe Netze, so wie wir sie in diesem Kontext verstehen, kennzeichnen sich hierbei durch eine hohe Anzahl an Kanten und Knoten. Diese sind allerdings nicht willkürlich und zufällig angeordnet, sondern unterliegen vielmehr bestimmten Regelmässigkeiten.

5.1.2. Effiziente Kodierung wiederkehrender Strukturen

Um komplexe Netze beherrschbar zu machen wurde beispielsweise in der vorherigen Arbeit von Schmidt [Sch09][2] ein Verfahren zur hierarchischen Beschreibung von Netzen entwickelt, mit der Zielsetzung, wiederkehrende Strukturen effizient kodieren zu können. Grössere Netze werden hierbei aus der Verschaltung kleinerer Module erzeugt. Somit reduziert sich der Kodierungsaufwand und damit einhergehend der Suchraum den der evolutionäre Algorithmus durchlaufen muss. Infolgedessen sollte es mit einem solchen Algorithmus möglich sein, komplexere Netze zu beherrschen. Weitere Verfahren, die dieses Ziel auf teilweise recht unterschiedliche Art und Weise verfolgen, haben wir in Kapitel 3 kennengelernt.

Anforderung an das zu entwickelnde Verfahren ist daher, bedingt durch die Anforderung der Beherrschbarkeit von komplexen Netzen, auch die effiziente Kodierung wiederkehrender Strukturen.

5.1.3. Effektive Genetische Operatoren

Desweiteren haben wir im vorangegangen Kapitel 3 die Schlussfolgerung getroffen, dass eine zielgerichtete, strukturerhaltende Verknüpfung der Genome massgeblich

[1] wie z.B. Backpropagation
[2] siehe Abschnitt 3.3.7

zum Erfolg eines evolutionären Verfahrens beiträgt. Folglich sollte das zu entwickelnde Verfahren effektive genetische Operatoren implementieren.

5.1.4. Anforderungen an die erzeugten Netze

Die Netze die wir erzeugen wollen, gliedern sich in drei logische Schichten. Die Eingangsschicht beinhaltet hierbei die Neuronen (bzw. Knoten), die Eingangssignale des umliegenden Systems empfangen. Analog dazu beinhaltet die Ausgangsschicht die Neuronen, die eine Ausgabe an das umliegende System zurückliefern. Die mittlere Schicht hingegen beinhaltet alle Neuronen die für die tatsächliche Berechnung erforderlich sind.

Dabei ist es nicht zwingend erforderlich, dass ein vom Verfahren erzeugtes Netz diese drei Schichten auch physisch realisiert. Es reicht vollkommen aus, wenn das Verfahren bestimmt, welche der Neuronen (bzw. Knoten) ein Eingabesignal erhalten, und von welchen Neuronen eine Ausgabe gelesen werden soll. Die Anzahl der Eingänge und Ausgänge wird hierbei von aussen vorgegeben und ist abhängig von der zu lösenden Aufgabe.

Anforderungen an die Gepulsten Neuronalen Netze In den so erzeugten Netzen werden die Neuronen durch die Knoten repräsentiert, die Synapsen von den Kanten des Netzes. Da wir das Modell von Izhikevich verwenden wollen[3], genügt es, wenn den Neuronen jeweils ein einfacher reelwertiger Parameter im Bereich zwischen -1.0 und 1.0 zugewiesen wird. Dabei bezeichnet ein negativer Wert ein hemmendes, *inhibitorisches* Neuron, ein positiver Wert ein verstärkendes, *exzitatives* Neuron. Neuronen, die ein reguläres oder flatterndes Pulsverhalten aufweisen (*regular spiking* oder *chattering*), sind Untertypen des exzitativen Neuronentyps und können durch diesen beschrieben werden.

Den Synapsen im Modell von Izhikevich sind jeweils zwei Parameter zugeordnet. Zum einen die synaptische Effizienz bzw. Leitfähigkeit, sowie die Verzögerungszeit. Die Verzögerungszeit lieg im Bereich zwischen $1ms$ und $50ms$. Die Angabe der synaptischen Effizienz machen wir optional, da diese sich auch durch das Einlernen mittels Spike-Timing Dependent Plasticity (STDP) bestimmen lässt.

Neben der Topologie des Netzes müssen demnach noch folgende Parameter vom Verfahren bestimmt werden:

- Zuordnung der Ein- und Ausgänge zu bestimmten Neuronen.
- Neuronentyp: Reelwertiger Parameter $r \in [-1.0, 1.0]$.
- Verzögerungszeit der Synapsen: Reelwertiger Parameter[4] $t \in [1, 50]ms$.

[3] siehe Abschnitt 2.1.7

[4] Die Präzision ist hierbei abhänig von dem bei der Simulation verwendeten Zeitschritt

- Optional: Effizienz der Synapsen: Reelwertiger Paramter $w \in [-10, 10]$.

Die erzeugten Netze sollen auch explizit Zyklen enthalten können, und wenn möglich mehrere Verbindungen zwischen zwei Knoten zulassen. Letzterer Punkt ist wünschenswert, aber nicht unbedingt erforderlich.

Anforderungen an die Netze für die Bewertung anhand der Zielgraphsuche
Wie im Einführungstext bereits erwähnt, wollen wir die Netze auch an einen gegebenen Zielgraph annähern. Dies dient der besseren Visualisierung und der genaueren Untersuchung verschiedener Aspekte des Verfahrens. Die Anforderungen an die so erzeugten Netze sind prinzipiell dieselben wie für die gepulsten neuronalen Netze, nur dass hier die Parameter der Knoten und Kanten einen anderen Wertebereich besitzen. Zudem verzichten wir hier auf den optionalen zweiten Parameterwert der Verbindungen.

- Parameter der Knoten: Reelwertiger Parameter $r \in [0.0, 1.0]$.
- Parameter der Kanten: Reelwertiger Parameter $w \in [0.0, 1.0]$.

5.2. Analyse und Begründung der Entscheidung

Durch die intensive Auseinandersetzung mit den vorhandenen Erzeugungsalgorithmen in Kapitel 3, sowie den Erfahrungen, die wir während der Implementierung unserer eigenen Prototypen gesammelt haben, sind wir zu dem Schluss gekommen, dass für unsere Anforderungen, die Wahl eines CPPN[5] als Kodierungsmethode am geeignetsten erscheint.

5.2.1. Verwendung eines CPPN

Die Vorteile, die wir in der Verwendung eines CPPN zur indirekten Kodierung von Graphen erkennen, sind im Folgenden zusammengefasst:

- Beliebige Aspekte eines Graphen lassen sich durch ein CPPN beschreiben, wie beispielsweise die Knotengewichtung, Kantengewichtung oder die Verbindungstopologie.
- Einzelne Aspekte, wie etwa die Gewichtung von Verbindungen, können zudem funktionalen Abhängigkeiten unterliegen, z.B. in Form von Gradientenverläufen. Eine funktionale Abhängigkeit zwischen einzelnen Aspekten ist ebenso möglich.

[5] siehe Abschnitt 3.3.6

- Muster, wie sie auch in der Natur auftreten, können durch ein CPPN nachgebildet werden. Insbesondere ist es möglich, Muster zu erzeugen, die folgende Eigenschaften aufweisen: *Symmetrie*, *Unvollkommene Symmetrie*, *Wiederholung* und *Wiederholung mit Variation*.
- Die Bildung von Modulen sowie die wiederholte Applikation der Module ist durch die Verwendung von periodischen Funktionen möglich. Eine Variation der Modulparameter wird zudem durch Kombination mit asymmetrischen Funktionen erreicht.
- Prinzipiell lassen sich mit einem CPPN Graphen beschreiben, die Strukturen enthalten, die unabhängig von der Komplexität (Anzahl der Knoten) des erzeugten Graphen sind. Dies bedeutet, dass mit ein und demselben CPPN Graphen unterschiedlicher Komplexität erzeugt werden können. Erreicht wird diese durch ein verfeinertes Sampling des Substrates.
- Dieses Hochskalieren der Komplexität bzw. Verfeinerung der Auflösung macht erst die effiziente Erzeugung beliebig grosser Netze möglich. Dies ist insbesondere im Bereich der Bilderkennung ein wichtiges Merkmal.
- Im Unterschied zu Verfahren, die Graphen schrittweise entwickeln, kann bei einem CPPN der Graph direkt in einem Schritt erzeugt werden. Die Auswertung des CPPN kann dabei massiv parallel erfolgen.

Neben den oben genannten Vorteilen wird die Technik bereits erfolgreich in einer Reihe wissenschaftlicher Arbeiten eingesetzt. Auf diese möchte ich im Folgenden kurz eingehen.

HyperNEAT HyperNEAT [GS07] basiert auf einem CPPN, welches durch NEAT [SM02] evolutionär entwickelt wird. Auf HyperNEAT wurde bereits in Abschnitt 3.3.6 eingegangen. Zu NEAT findet sich eine genaue Erklärung in 3.3.5.

HyperGP HyperGP [BKŠ09] hingegen unterscheidet sich von HyperNEAT in der Verwendung von Genetischem Programmieren anstelle von NEAT zur Optimierung des CPPN. Die Autoren verzichten hierbei auf Nischenbildung und die Kenntnisse über strukturelle Ähnlichkeiten der Genome bei der Paarung. Diese Fähigkeiten werden dem Verfahren später in Form von HyperGPEFS [DS12] hinzugefügt.

HyperNEAT-CCT HyperNEAT-CCT (HyperNEAT plus Connection Cost Technique) [HCM14] basiert genau wie HyperNEAT auf einem CPPN, verwendet jedoch anstelle von NEAT den multimodalen evolutionären Algorithmus NSGA-II [Deb+02] zur Optimierung des CPPN. Etwas irreführend ist hierbei die Namensgebung. HyperNEAT hat sich als Synonym für die Verwendung eines CPPN zur Entwicklung von neuronalen Netzen etabliert und setzt dabei *nicht* die Verwendung des NEAT Algorithmus voraus[6].

[6]HyperNSGA wäre in diesem Fall eine bessere Wahl für den Namen des Algorithmus gewesen.

Während bei NEAT die strukturelle Ähnlichkeit der Genome durch historische Markierungen zur Nischenbildung, und damit zum Schutz neu entstehender Innovationen eingesetzt wird, so verwendet HyperNEAT-CCT ein Verfahren, dass die Vielfalt fördert und damit indirekt den Schutz neuer Innovationen gewährleistet.

Um dies zu erreichen, wird für jedes CPPN alle möglichen Ausgabewerte berechnet, die für jede mögliche Eingabe entstehen können, und mit einem mittleren Wert der gesamten Population verglichen. Die Abweichung von diesem Mittelwert fliesst als weiteres Kriterium in die Bewertung mit ein und ermöglicht eine Diversität auf Basis des Verhaltens der Netze (Behavioral Diversity). Möglich wird dies, da NSGA-II die Lösungen innerhalb derselben Pareto-Front anhand der sogenannten Crowding-Distanz auswählt, d.h. Lösungen mit einem grösseren Abstand zueinander bevorzugt in die neue Population übernommen werden.

Als drittes Kriterium fliessen zudem die Kosten der Verbindungen (Connection Cost) des Netzes in die Bewertung der Genome mit ein. Diese wird als Summe der quadrierten Längen aller Verbindungen berechnet und dient der Bevorzugung kürzerer Verbindungen. Zudem hat dies eine höhrere Modularität zur Folge.

Um die Verbindungskosten weniger stark in die Bewertung mit einfliessen zu lassen als die beiden anderen Kriterien, verwenden die Autoren PNSGA [CML13], eine probabilistische Variante von NSGA-II. Hierbei wird jedem Kriterium eine Wahrscheinlichkeit zugeordnet, mit der diese Einfluss auf die von NSGA-II verwendete Dominanz-Relation zur Bestimmung der Pareto-Fronten hat. Sind alle Kriterien mit der Wahrscheinlichkeit von 1.0 belegt, so handelt es sich um klassisches NSGA-II. Ist ein Kriterium beispielsweise mit der Wahrscheinlichkeit von 0.25 definiert, so fliesst dieses im Schnitt nur in einem von vier Fällen in die Bewertung mit ein.

Bei den genetischen Operatoren setzt HyperNEAT-CCT nur asexuelle Mutation ein (strukturelle Mutation sowie Gewichtsmutation) und verzichtet komplett auf paarweise Kreuzung (Crossover). Die Autoren verwenden aus Gründen der Einfachheit nicht das Konzept der Markierung des historischen Ursprungs von Innovationen wie es in NEAT zum Einsatz kommt, und können daher auch keine sinnvolle paarweise Kreuzung durchführen.

HyperNEAT-ES Während bei allen vorherigen Verfahren die Platzierung der Neuronen auf einem Substrat vorgegeben sein muss, so wird die Konfiguration (Anzahl Neuronen, Anzahl Schichten, Dichte) bei HyperNEAT-ES [RS12] ebenso evolutionär entwickelt. Dabei wird NEAT zur Optimierung des CPPN eingesetzt.

Anwendungsgebiete HyperNEAT-basierter Verfahren In [DKS09] wird HyperNEAT verwendet um virtuellen Robotern das Fahren in einer simulierten Umgebung beizugringen, während in [Lee+13] mehrbeinigen physischen Robotern eine Gangart antrainiert wird. [GS08] beschreibt den Einsatz von HyperNEAT zum Erlernen der

Regeln von Schach und geht insbesondere auf die Ausnutzung der problemspezifischen Geometrie ein und wie sich dies positiv auf die Performanz des Lernalgorithmus auswirkt.

In dem Paper von HyperNEAT-ES [RS12] wird T-Maze [BF03; Sol+08] als Problemstellung verwendet. In T-Maze muss ein Agent ein Labyrinth in Form eines oder mehrerer T's durchfahren und dabei möglichst viele Belohnungspunkte sammeln, die im Labyrinth verstreut sind. Nach Einsammeln eines Belohnungspunktes muss der Agent dabei zu seinem Startpunkt zurückkehren.

HyperNEAT-CCT verwendet hingegen, neben 5-XOR und H-XOR, das Retina Problem [KA05; CML13] als Problemstellung. Bei dem Retina Problem bekommt ein neuronales Netz die Eingaben von zwei Mustern (links und rechts) und muss hierbei entscheiden, ob es sich bei dem linken und rechten Muster um das gleiche Muster handelt (Ausgabe $>= 0$) oder nicht (Ausgabe < 0).

5.2.2. Wahl des Evolutionären Algorithmus

Wie wir im letzten Abschnitt gesehen haben, kommen in den dort beschriebenen Verfahren zur Optimierung von CPPNs drei verschiedene Typen von evolutionären Algorithmen zum Einsatz. Dies sind:

GP: *Genetisches Programmieren.* Eingesetzt von HyperGP. GP unterscheidet sich im Wesentlichen von einem genetischen Algorithmus (GA) in der zugrundeliegenden Repräsentation des Genoms, meisst in Form eines Baumes oder Liste. Zudem wird GP gerne für die indirekte Kodierung eingesetzt. Während bei einem direktkodierenden GA meisst eine 1:1-Beziehung zwischen *Genotyp* $\rightarrow$ *Phenotyp* besteht, so findet sich bei der indirekten Kodierung mit GP meisst eine N:1-Beziehung, aufgrund der Redundanzen in der Kodierung. Darüber hinaus ist die Abbildung von *Phenotyp* $\rightarrow$ *Fitness* bei einem GP häufig ebenso N:1.

NSGA-II: *GA mit nichtdominiertem Sortieren.* Eingesetzt von HyperNEAT-CCT. Erlaubt die Optimierung anhand mehrerer Zielvorgaben. NSGA-II verwendet als Selektionsoperator Elitäres Abschneiden (Elitist Truncation) mit dem Pareto-Rang und der Crowding-Distanz als Ordnung. Für die Wahl der Individuen zur Rekombination wird binäres Tournament auf derselben Ordnung angewendet. Genetische Operatoren definiert NSGA-II nicht.

NEAT: *NeuroEvolution augmentierter Toplogoien.* Eingesetzt von HyperNEAT. Eines der Hauptmerkmal von NEAT ist die topologieerhaltende Kreuzung von Genomen durch Gen-Alignment. In diesem Punkt unterscheidet sich NEAT im Besonderen von GP, wobei letzteres meisst nur ein zufälliges Durchmischen der Genome an grob-granularen Schnittpunkten erlaubt, nicht jedoch eine toplogische Zuordnung. Die Nischenbildung von NEAT kann als ein Teil des Selektionsmechanismus angesehen werden und somit als orthogonaler Aspekt

zur Kreuzung betrachtet werden. Allerdings wird die Nischenbildung in NEAT erst durch das Maß der genetischen Kompatibilität, welches in Zusammenhang mit der Kreuzung und Mutation steht, effizient ermöglicht.

Genetisches Programmieren (GP), NEAT und NSGA-II beinhalten zum Teil orthogonale Konzepte, die miteinander kombiniert werden können. So lassen sich Teilaspekte von NEAT, wie z.B. die Kreuzung, mit dem Selektionsmechanismus von NSGA-II kombinieren, indem dieser für die Selektion innerhalb der von NEAT gebildeten Nischen eingesetzt wird.

Selbstverständlich existieren eine Reihe weiterer evolutionärer Algorithmen, insbesondere für die multimodale Optimierung. Dessen zusätzliches Studium und Implementierung würde den Rahmen dieser Arbeit jedoch bei weitem sprengen. Ein genauer Vergleich verschiedener multimodaler Algorithmen ist [Neb+08] zu entnehmen. Als Ergebnis aus dieser Studie ist festzuhalten, dass NSGA-II zwar nicht immer die beste Performanz liefert, jedoch in den meissten Fällen sehr robust abschneidet.

Neben dem Hauptbestandteil von evolutionären Algorithmen, der *Selektion*[7], sollte der Algorithmus den wir für die Optimierung der CPPN einsetzen wollen, noch folgende Eigenschaften aufweisen:

1. Ein Mechanismus zur *Effektiven Kreuzung* zweier Genome. Die Effektivität steht hierbei in Bezug zur Konvergenzgeschwindigkeit des Algorithmus.
2. Ein Mechanismus zum *Schutz neuer Innovationen*, etwa durch Nischenbildung oder durch Hinzunahme eines weiteren Kriteriums in die Fitnessbewertung.
3. Ein Mechanismus zum Erhalt der *Diversität*. Ein Mangel an genetischer Vielfalt führt im schlimmsten Fall zur vollständigen Wirkungslosigkeit der Kreuzungsoperation. Änderungen können somit nur noch durch Mutation hervorgebracht werden. Diese operiert jedoch nur punktuell und benötigt viel Zeit um eine entsprechende Verbesserung zu bewirken.

Während das Fehlen von Diversität sich lediglich in einer Verminderung der Performanz bemerkbar macht, so kann das Fehlen des Schutzes neuer Innovationen zu einem vollständigen Ausbleiben einer guten Lösung führen. Dann nämlich, wenn ein Elitärer Algorithmus zum Einsatz kommt und sich jede Veränderung erst durch eine schlechtere Fitness bemerkbar macht und damit sofort verworfen wird. Eine multimodale Fitnessfunktion verringert dieses Risiko, indem es die Wahrscheinlichkeit erhöht, dass bei einer Änderung am Genom eine Verbesserung in eines der mehreren Kriterien erfolgt.

Aus unserer Sicht erfüllt keiner der untersuchten Verfahren alle Aspekte vollständig. Auf die einzelnen oben genannten Aspekte gehen wir in den folgenden Abschnitten genauer ein.

[7] Selektion ist nur nötig, da uns begrenzte Ressourcen zur Verfügung stehen. Aus demselben Grund findet Selektion auch in der Natur statt.

5.2.3. Effektive Kreuzung

Von allen untersuchten Algorithmen verwendet lediglich NEAT einen Ansatz der eine effektive Kreuzung zweier Genome verspricht. Knoten und Kanten der zu optimierenden Graphen enthalten hierfür global eindeutige Innovationsnummern. Diese ermöglichen eine effiziente Zuordnung von Knoten und Kanten aller Graphen der Population, ohne dass hierfür eine rechenintensive topologische Analyse notwendig wird. Somit kann bei der Kreuzung der beiden Genome ein Austausch von Information zwischen zueinander passender Knoten und Kanten erfolgen.

Finden allerdings parallele Entwicklungen statt, z.B. in unterschiedlichen Nischen, so führt dies zu einer genetischen *Divergenz* und damit zur Unvereinbarkeit bestimmter Merkmale bei der Kreuzung, selbst wenn beide Genome eine identische Topologie aufweisen. Dies mag zwar biologisch plausibel sein, hat jedoch in unserem Kontext folgende Nachteile:

- Die Kreuzung findet ausschliesslich anhand des gemeinsamen historischen Ursprungs statt, nicht aufgrund der Zuordnung der wirklichen Topologien beider Graphen. Genome ohne gemeinsame Vorfahren divergieren zu 100% und sind somit vollständig inkompatibel.
- Daher ist es nicht möglich, Genome aus verschiedenen Läufen des evolutionären Algorithmus in einer gemeinsamen Population zu vereinen, etwa um bereits erlerntes Wissen zu kombinieren[8].
- Die Nischenbildung fördert eine parallel Entwicklung und damit die Divergenz der genetischen Kompatbilität.
- Die globale Kenntnis und die erforderliche Synchronisation aller existiernder Innovationen erschwert zudem eine effiziente massiv-parallele Implementierung.

Im Eingangstext des Kapitels hatten wir bereits erwähnt, dass wir unser Verfahren auch anhand der Zielgraphsuche austesten wollen. Hierzu haben wir unter anderem den in [Nik12] als *Neighbor Matching* beschriebenen Algorithmus zur Bestimmung der Ähnlichkeit zweier Graphen implementiert. Der Algorithmus bestimmt iterativ eine Zuordnung zwischen den Knoten zweier Graphen und minimiert dabei die Summe in den Abweichungen der Knotenähnlichkeiten.

In unserem Verfahren wollen wir das *Neighbor Matching* bei der Kreuzung zweier Genome zur Bestimmung des Gene-Alignments einsetzen. Damit wollen wir erreichen, dass der Austausch von genetischer Information nur zwischen ähnlichen Knoten und Kanten erfolgt. NEAT bezeichnet diese Eigenschaft unter dem Namen *Gene-Alignment of Disparate Topologies*. Da es sich bei den von uns erzeugten CPPNs um recht kleine Netze handelt, ist diese topologische Analyse darüber hinaus auch

[8] Inwieweit dies von Erfolg gekrönt ist, sei an dieser Stelle ausser Acht gelassen. Zumindest erscheint uns diese Möglichkeit von Interesse.

effizient durchführbar, zumal wir das Ähnlichkeitsmaß, anders als NEAT, nicht zur Nischenbildung einsetzen[9].

Alle oben genannten Nachteile, die den Ansatz von NEAT in Bezug auf die Kreuzung betreffen, können durch den Einsatz von *Neighbor Matching* mit unserem Verfahren behoben werden. Zudem kapseln Genome nun vollständig die gesamte Information, die für alle genetischen Operatoren notwendig ist. Davon ist insbesondere die strukturelle Mutation betroffen, die in NEAT ein globales Wissen voraussetzt. Auch können wir mit unserem Ansatz Genome, die aus beliebigen evolutionären Läufen stammen, miteinander kombinieren. Den Einsatz einer, im Vergleich zu NEAT, rechenintensiveren topologische Analyse, machen wir dabei durch die Möglichkeit der trivialen Parallelisierbarkeit wett.

Einen weiteren Aspekt den wir gegenüber dem ursprünglichen NEAT verbessern wollen ist die recht starke Veränderung, die von der Kreuzungsoperation ausgeht. Bei NEAT wird die Gewichtsinformation passender Kanten entweder von dem einen Elternteil oder aber von dem anderen in den Nachkommen übernommen. Eine Anpassung der Gewichtswerte findet dabei nicht statt. Einige Implementierung wie z.B. ANJI[10] erlauben, neben dem Austausch der Gewichtsinformation, auch eine probabilistische Angleichung der Werte. Die recht starke Auswirkung der Kreuzungsoperation von NEAT hat dazu geführt, dass manche Ansätze komplett darauf verzichten, so haben auch wir bessere Ergebnisse erzielt, wenn auf Kreuzung verzichtet wurde. Diesen negativen Effekt betrifft insbesondere Verfahren die HyperNEAT bzw. ein CPPN einsetzen, da hier das Verändern einer Gewichtung eine sehr viel stärkere Auswirkung haben kann als dies bei der direkten Kodierung von Graphen der Fall ist.

5.2.4. Schutz neuer Innovationen

Ein zentraler Aspekt bei NEAT ist die Nischenbildung. Erst durch die Nischenbildung ist es möglich, dass kleine Veränderungen am Genom, die sich nicht sofort in der Fitness auswirken, in der Population eine gewisse Zeit überleben können, um sich entsprechend entfalten zu können. Dies erlaubt es, dass NEAT die Mutation-Operation derart ausführt, dass möglichst geringe Änderungen am Genom entstehen. Dies hat zudem den Vorteil, dass jede kleine Änderung auch bewertet wird. Ohne Nischenbildung würden die Mutationen jedoch sofort den etablierten Genomen unterliegen und ausgelöscht werden.

HyperNEAT-CCT hingegen erreicht den Schutz neuer Innovationen durch Hinzunahme der *Behavioral Diversity* als weiteres Kriterium zur multimodalen Fitnessbewertung. Dies hat den Vorteil, dass bei CPPNs von der Topologie nicht auf das Verhalten geschlossen werden kann und somit die Behavioral Diversity ausdrucksstärker ist als das von NEAT eingesetzte Maß der genetischen Kompatibilität.

[9] Um eine optimale Nischenbildung zu gewährleisten müsste man $O(n^2)$ Vergleiche durchführen.
[10] Another NEAT Java Implementation: `http://anji.sourceforge.net/`

5.2.4.1. Nischenbildung in NEAT

Grundlage der Nischenbildung in NEAT ist die Genetische Kompatibilität. Die Genetische Kompatibilität ist ein Mass, dass bestimmt in wievielen Genen sich zwei Genome überschneiden, wie stark die Gewichtsdifferenz bei den überschneidenden Genen ist, und wie stark sich beide Genome in der Anzahl disjunkter und exzessiver Gene unterscheiden. Einzelne Faktoren können hierbei unterschiedlich gewichtet werden (siehe Gleichung 3.1).

Für die Zuordnung eines Genoms zu einer Nische verwendet NEAT einen Grenzwert. Liegt die Genetische Kompatibilität des Genomes zu einem beliebig ausgewählten Genom der Nische unter diesem Grenzwert, so wird es dieser Nische zugeordnet. Findet sich keine passende Nische, so wird eine neue Nische erzeugt.

Bei unserer Implementierung der Nischenbildung von NEAT hat sich gezeigt, dass sich insbesondere die Wahl des Grenzwertes als schwierig herausstellt. Ist der Grenzwert zu gross gewählt, so fallen alle Genome in eine einzige Nische. Ist er zu klein gewählt, so bildet jedes Genom seine eigene Nische. Beide Extreme führen dazu, dass aus dieser Situation kein Entkommen mehr ist, da sich die genetische Diversität innerhalb der Nischen nur noch geringfügig ändert.

Bestimmung des Grenzwertes Um den Grenzwert entsprechend einzustellen haben wir verschiedene Ansätze ausprobiert. Eine Möglichkeit besteht in der Bestimmung der mittleren Distanz zwischen den Nischen. Dazu werden zufällig je zwei disjunkte Nischen selektiert aus denen jeweils ein zufälliges Genome als Probe entnommen wird. Diese beiden Genome, aus unterschiedlichen Nischen, werden miteinander verglichen und die Genetische Kompatibilität (Distanz) ermittelt. Durch mehrmaliges Ausführen dieser Prozedur wird ein Mittelwert oder Median gebildet. Dieser Mittelwert wird als Grenzwert zur Nischenbildung verwendet.

Nischenbildung ohne Grenzwert Da sich die Bestimmung eines geeigneten Grenzwertes als schwierig herausgestellt hat, haben wir versucht ohne einen Grenzwert auszukommen. Um dies zu erreichen haben wir die gesamte Population in n Nischen unterteilt, mit der Eigenschaft, dass Genome innerhalb der eigenen Nische eine grössere genetische Kompatibilität zueinander aufweisen als zu Genomen ausserhalb der eigenen Nische.

Bei der Reproduktion, die innerhalb der einzelnen Nischen erfolgt, haben wir hierzu die elitären Genome in ihrer jeweiligen Nische belassen, die erzeugten Nachkommen jedoch der Nische zugeordnet, mit der die grösste Übereinstimmung ermittelt wird. Für jeden Nachkommen haben wir hierfür jeweils k Proben aus jeder Nische genommen und so die mittlere Distanz des Nachkommens zu dieser Nische ermittelt. Die Nische mit der geringsten mittleren Distanz wurde als Zielnische gewählt.

Eine weitere Methode besteht darin, eine festen Grenzwert zu verwenden, jedoch die Anzahl der Nischen zu begrenzen. Ist die Bildung von n Nischen erreicht, so

wird ein Genom entweder zufällig einer Nische zugeordnet, oder es wird der Nische zugeordnet, mit der es die grösste Übereinstimmung wie oben beschrieben aufweist.

Nischenbildung anhand k-nächster Nachbaren Eine Methode, die von der Referenzimplementierung SharpNEAT [11] verwendet wird, ist die Einteilung in Nischen anhand der Bestimmung der k-nächsten Nachbaren. Hierfür wird jedem Genom ein Koordinatenpunkt in einem hoch-dimensionalen Raum zugeordnet, wobei die Kanten-Innovationen die Dimensionen bilden. Die Gewichtung der jeweiligen Kante bestimmt den Koordinatenwert der entsprechenden Dimension. Anschliessend werden die Genome anhand einer Clusteranalyse in Nischen eingeteilt.

Kritik an der Ablationsstudie von NEAT in Bezug auf die Nischenbildung Die Autoren von NEAT belegen in einer Ablationsstudie [SM01], dass drei Aspekte in Kombination erforderlich sind, um die gute Performanz von NEAT erklären zu können. Diese sind:

1. Schutz neuer Innovation durch Nischenbildung,
2. Entwicklung ausgehend von einer minimale Anfangstopologie, sowie
3. schrittweise Steigerung der Komplexität.

Hierzu vergleichen die Autoren NEAT gegenüber einem Algorithmus, dem ein einzelner oben genannter Aspekt fehlt. Kritisch zu sehen ist die Ablation der Nischenbildung, d.h. der Vergleich von NEAT gegenüber NEAT ohne Nischenbildung, da hierbei nicht nur die Nischenbildung entfernt wurde, sondern auch mit einer zufälligen Startkonfiguration begonnen wurde. Begründet wird dies, um den Erhalt der Diversität zu gewährleisten. Dies widerspricht jedoch dem zweiten Aspekt, der Entwicklung ausgehend von einer minimalen Topologie. Somit kann keine eindeutige Aussage über die Signifikanz der Nischenbildung von NEAT gewonnen werden.

5.2.4.2. Behavioral Diversity in HyperNEAT-CCT

Um die Behavioral Diversity zu berechnen, wertet man zuerst für jede mögliche Eingabe das CPPN aus. Im Falle von HyperNEAT besteht die Eingabe aus den Koordinatenpaaren der Neuronen für die eine Gewichtung ermittelt werden soll. Diese sind a priori durch das Substrat fest vorgegeben. Aus den so entstehenden Ausgabewerten wird ein Bitvektor erzeugt, der an der jeweiligen Stelle genau dann eine 1 enthält sofern die Ausgabe des CPPN $>= 0$ ist, ansonsten eine 0. Dies wird für jedes Genom der gesamten Population durchgeführt. Die Behavioral Diversity wird nun als mittlere Hamming-Distanz des Bitvektors eines Genoms zu allen anderen Bitvektoren der Population berechnet.

[11] `http://sharpneat.sourceforge.net/`

Beispiel Es sei ein eindimensionales Substrat mit 2 Neuronen gegeben. Die Neuronen sind an den Positionen (-1.0) und (1.0) platziert. Es existieren 2^2 mögliche Eingaben in das CPPN, je nachdem für welche Verbindung die Gewichtung berechnet werden soll: $(-1.0, -1.0)$, $(-1.0, 1.0)$, $(1.0, -1.0)$ und $(1.0, 1.0)$. Die Eingabe $(-1.0, 1.0)$ würde beispielsweise die Gewichtung zwischen dem Neuron 0 an Koordinate (-1.0) und Neuron 1 an Koordinate (1.0) bestimmen.

Seien vier Genome A, B, C und D gegeben, die sich in ihren CPPNs unterscheiden. Durch Auswertung ihres CPPNs erhalten wir die zugehörigen Bitvektoren $b_A = 0000$, $b_B = 0110$, $b_C = 1101$ und $b_D = 1111$. Sei zudem $d_H(a, b)$ als Hamming-Distanz der Bitvektoren a und b definiert.

Im ersten Schritte berechnen wir zwischen allen Paaren die Hamming-Distanz ihrer Bitvektoren.

$$
\begin{array}{lll}
d_H(b_A, b_B) = 2 & d_H(b_A, b_C) = 3 & d_H(b_A, b_D) = 4 \\
d_H(b_B, b_A) = 2 & d_H(b_B, b_C) = 3 & d_H(b_B, b_D) = 2 \\
d_H(b_C, b_A) = 3 & d_H(b_C, b_B) = 3 & d_H(b_C, b_D) = 1 \\
d_H(b_D, b_A) = 4 & d_H(b_D, b_B) = 2 & d_H(b_D, b_C) = 1
\end{array}
$$

Die mittlere Hamming-Distanz $\hat{d}_H(i)$ eines Genomes i berechnet sich nun durch 5.1, wobei n die Anzahl der Genome angibt.

$$
\hat{d}_H(i) = \frac{\sum_{j \neq i} d_H(b_i, b_j)}{n - 1} \tag{5.1}
$$

Berechnet man die mittlere Hamming-Distanz für alle vier Genome, so erhält man:

$$
\begin{aligned}
\hat{d}_H(A) &= (2 + 3 + 4)/3 = 3.0 \\
\hat{d}_H(B) &= (2 + 3 + 2)/3 = 2.33 \\
\hat{d}_H(C) &= (3 + 3 + 1)/3 = 2.33 \\
\hat{d}_H(D) &= (4 + 2 + 1)/3 = 2.33
\end{aligned}
$$

Genom A hat mit 3.0 die höchste Behavioral Diversity und würde daher selbst bei einer schlechteren Fitness, z.B. in der Erkennungsrate, mit in die nächste Generation übernommen werden. Dies liegt daran, da es Teil der ersten Pareto-Front ist. Die Genome B, C und D hingegen haben nicht nur eine schlechtere Fitness in der Behavioral Diversity als Genom A, sondern konkurrieren zudem noch gegeneinander

indem sie ein schlechtes Maß in der Crowding-Distanz bekommen. Sollten B, C und D alle in der gleichen Pareto-Front zusammen mit weiteren Genomen liegen, so würde, abhängig von der Crowding-Distanz der anderen Kriterien, eventuell nur eines der Genome von B, C oder D bevorzugt in die nächste Generation übernommen werden.

5.2.4.3. Fazit

Zum Schutz neuer Innovationen wollen wir in unserem Verfahren Behavioral Diversity einsetzen. Diese Entscheidung begründen wir zum einen anhand der oben beschriebenen Schwierigkeiten mit der Nischenbildung von NEAT die bei unserer Implementierung auftraten. Zum anderen sehen wir den klaren Vorteil von Behavioral Diversity darin, dass es direkt das Verhalten eines CPPN bewertet, nicht nur dessen Topologie. Zudem erlaubt der Einsatz von NSGA-II die Hinzunahme weiterer ähnlicher Kriterien in die Fitnessbewertung eines Genomes, wie z.B. die Verbindungskosten, während dies bei NEAT nicht vorgesehen ist.

5.2.5. Erhalt der Diversität

Während bei HyperNEAT-CCT die Behavioral Diversity in Kombination mit der Crowding Distanz von NSGA-II für den Erhalt der Diversität sorgen soll, so dient in NEAT die Nischenbildung diesem Zweck. Allerdings findet bei NEAT innerhalb der Nischen kein Erhalt der Diversität statt. Dies hat sich bei unserer Implementierung von HyperNEAT aufgrund der Problemen bei der Nischenbildung als eine Ursache für die schlechte Performanz herausgestellt.

Grundsätzlich sehen wir in der Behavioral Diversity, aufgrund der Beurteilung des Verhaltens anstelle der Topologie, eine bessere Maßnahme sowohl für den Erhalt der Diversität als auch zum Schutz neuer Innovationen. Genome, die einen sehr grossen Abstand zum Mittelwert des Verhaltens der gesamten Population aufweisen, werden so bevorzugt und können unabhängig von ihrer sonstigen Fitness, wie z.B. der Erkennungsrate oder Ähnlichkeit zum Zielgraphen, überleben.

Allerdings zeigte sich in unserem Prototyp, dass die Behavioral Diversity alleine, den Erhalt der Diversität nicht garantieren kann. Der Grund hierfür liegt in der Degenerierung der elitären Selektion von NSGA-II bei Vorhandensein von vielen identischen Fitnesswerten[12]. Diese Situation kann entstehen, wenn durch Mutation oder Kreuzung keine Änderung am Fitnesswert bewirkt wird. Findet dies insbesondere bei Individuen mit guter Fitness statt, so verdrängen die ebenso fitten Nachkommen mit der Zeit alle weiteren Individuen der Population.

[12] Ähnliche Erfahrungen wurden auch von anderen Autoren gemacht: 'elitist truncation of NSGA-II decrease population diversity rapidly in few generations, and leads to poor performance.' ([Wat+10])

Bei der numerischen Optimierung tritt dieses Problem bei NSGA-II nur selten auf. Dies liegt daran, dass die Abbildung von *Genom* → *Fitness* meisst 1:1 erfolgt. Anders ist dies bei der Optimierung von Problemen die eine indirekte Kodierung des Genoms erfordern. Hier findet oft eine N:1-Abbildung zwischen *Genom* → *Fitness* statt. Dies führt zu o.g. Problem.

5.2.5.1. Erweiterung von NSGA-II um Redundanz-Regulierung

Eine genaue Beschreibung des oben genannten Problems, sowie ein Ansatz wie man NSGA-II entsprechend verändert kann um dieses zu vermeiden, findet sich in [Wat+10]. Kernpunkte dieses Ansatzes, den wir im folgenden als *NSGP* bezeichnen, sind hierbei:

- *Semi-elitäres Abschneiden:* Bei NSGP werden aus jeder Pareto-Front nur diejenigen Individuen entnommen, die sich in ihren Fitnesswerten unterscheiden. Im Unterschied dazu werden bei NSGA-II alle Individuen einer Pareto-Front komplett übernommen, mit Ausnahme der zuletzt betrachteten Front, deren Individuen eventuell nicht mehr ganz in die Population übernommen werden können, da ansonsten die gewünschte Populationsgrösse überschritten wird.
- *Verbot redundanter Individuen:* Wird ein Nachkomme erzeugt, dessen Genom in dieser Form schon im Laufe der Evolution aufgetreten ist, so wird der Nachkomme solange mutiert bis dessen Genom eine Neuerung hervorbringt.
- *Phenotyp-uniformes Sampling:* Dies betrifft die Warhscheinlichkeit mit der ein Individuum zur Bildung von Nachkommen herangezogen wird. Bei NSGA-II ist diese Wahrscheinlichkeit für jedes Individuum gleich, unabhängig davon, ob das Individuum sich mit anderen Individuen denselben Fitnesswert teilt oder nicht. Bei NSGP hingegen ist diese Wahrscheinlichkeit abhängig von der Anzahl N_{diff} unterschiedlicher Punkte des Objektivraumes. Individuuen die im Objektivraum auf denselben Punkt fallen (d.h. mit gleicher Fitness), teilen sich demnach die Wahrscheinlichkeit: $p_i = 1/C_i \cdot N_{diff}$, p_i ist hierbei die Wahrscheinlichkeit eines Individuum in Punkt i, wobei C_i Individuen auf diesen Punkt fallen.
- *Durchschnittliche Crowding-Distanz:* Die Crowding-Distanz wird zusammen mit dem Pareto-Rang bei der Tournament-Wahl benötigt, um *fittere* Individuen bei der Reproduktion zu bevorzugen. Die Berechnung der Crowding-Distanz bei NSGA-II hat jedoch bei Individuen mit gleichen Fitnesswerten zur Folge, dass diese eine unterschiedliche Bewertung in deren Crowding-Distanz bekommen. Dies liegt daran, da immer der Abstand zu umliegenden Werten eines bestimmten Kriteriums berechnet wird. Fallen mehrere Individuen in einem bestimmten Kriterium auf denselben Punkt, so bekommt nur das erste und das letzte dieser Individuuen eine positive Distanz zu ihrem Nachbarn, die Mittleren hingegen bekommen eine Distanz von 0. NSGP verwendet daher

für Individuuen, die auf denselben Punkt fallen, die durschnittliche Crowding-Distanz des Punktes.

- *Grosse Tournamentwahl:* NSGA-II verwendet binäres Tournament, um Eliten bei der Reproduktion zu bevorzugen. Da NSGP auf Redundanzen verzichtet, die Population somit weniger Kopien der Eliten enthält als bei NSGA-II, so wird eine grössere Tournamentwahl benötigt[13].

5.3. Beschreibung des eigenen Verfahrens

Nach der im letzten Abschnitt durchgeführten genauen Analyse einzelner Optionen sind wir zu folgender Entscheidung gekommen:

- Als Kodierungsmethode setzen wir ein CPPN ein, welches wie in HyperNEAT zur Bestimmung der Verbindungsgewichte konfiguriert wird. Darüber hinaus verwendet wir das CPPN um die Knotengewichtung zu ermitteln.
- Die Optimierung der CPPNs erfolgt evolutionär, wobei wir *NSGP* einsetzen. Dieses erweitert den multimodalen Algorithmus NSGA-II um die in [Wat+10] beschriebenen Eigenschaften zur Redundanz-Regulierung. Die Selektion der Individuen erfolgt anhand mehrerer Kriterien:
  1. Problemspezifische Fitness (Erkennungsrate bzw. Zielgraphähnlichkeit),
  2. Behavioral Diversity (Schutz der Innovationen *und* Erhalt der Diversität),
  3. Verbindungskosten (Bevorzugung modularer Strukturen).
- Effektive Kreuzung erreichen wir durch eine topolgische Analyse der zu kreuzenden CPPNs. Hierfür setzen wir *Neighbor Matching* [Nik12] ein. Darüber hinaus nähern wir die Gewichtswerte passender Kanten gegeneinander an, und verhindern damit eine zu starke Auswirkung der Kreuzung.
- Die strukturelle Mutation und Mutation der Gewichte übernehmen wir in weiten Teilen von NEAT. Bei der Bildung neuer Verbindungen bevorzugen wir jedoch Knoten, mit niedrigem Verbindungsgrad. Hinzu kommt eine Mutation, die auf der Aktivierungsfunktion der Knoten operiert, sowie Operationen, die symmetrische Kanten erzeugen.

In den folgenden Abschnitten beschreiben wir die einzelnen Aspekte unseres Verfahrens im Detail.

5.3.1. Ablauf des Evolutionären Algorithmus

Als Basis kommt der elitäre und multimodale *Nondominated Sorting Genetic Algorithm II* [Deb+02], abgekürzt NSGA-II, zum Einsatz. Eine Beschreibung von NSGA-II findet sich in Abschnitt 2.2.4.1. Um eine hohe Diversität zu gewährleisten, haben

[13]Das Paper beschreibt eine Tournamentgrösse von 10 bei einer Populationsgrösse von 50.

wir diesen wie in Abschnitt 5.2.5.1 und [Wat+10] beschrieben um Fähigkeiten der Redundanz-Regulierung erweitert. Fortan nennen wir den erweiterten Algorithmus *NSGP*.

Der Ablauf eines Evolutionsschrittes ist in Algorithmus 6 beschrieben. Ausgangspunkt ist eine nach Punkt 8 ff. des Algorithmus bewertete Anfangspopulation. Die Crowding-Distanz ist der normierte Abstand eines Lösungspunktes zu seinen benachbarten Punkten derselben Pareto-Front, berechnet über alle Kriterien. Als Beispiel zeigt Abbildung 2.6 drei Pareto-Fronten bei der Optimierung des ZDT1 Problems. Hier kann die Crowding-Distanz als Euklidische Distanz benachbarter Punkte angesehen werden, mit entsprechend dem Wertebereich der Dimension normierten Koordinaten. Die Crowding-Distanz wird zur Auswahl von Lösungen innerhalb derselben Pareto-Front verwendet. Eine genauere Beschreibung von NSGA-II, insbesondere die der Crowding-Distanz und der Bestimmung der Pareto-Fronten in $O(MN^2)$ ist dem Paper zu entnehmen.

5.3.2. Konfiguration des CPPN

Für die Knoten unseres CPPN verwendet wir die in Abbildung 5.1 angegebenen Aktivierungsfunktionen. Die Eingangssignale eines Knotens werden hierbei mit der entsprechenden Gewichtung der eingehenden Kante multipliziert und aufsummiert. Hiervon wird der Bildwert anhand der Aktivierungsfunktion des Knotens berechnet und als Ausgangssignal an die nachfolgenden Knoten weitergegeben. Die Kantengewichte sind auf den Bereich $[-3, 3]$ beschränkt. Zudem erlauben wir maximal eine Verbindung zwischen je zwei Knoten. Bei einem CPPN handelt es sich um ein azyklisches Netz, somit lässt sich jeder Ausgang als eine Funktion der Eingänge beschreiben.

Symbol	Beschreibung	Funktion $f(x)$	Bildbereich
$\mathcal{L}$	Lineare Funktion	x	$(-\infty, +\infty)$
$\mathcal{L}_{cl}$	Lineare Funktion, beschränkt	$max(-1.0, min(1.0, x))$	$[-1, 1]$
abs	Betragsfunktion	$\|x\|$	$[0, \infty)$
$\mathcal{G}_{0,1}$	Gaussche Funktion	$e^{-(2.5x)^2}$	$[0, 1)$
$\mathcal{G}$	Bipolare Gaussche Funktion	$2e^{-(2.5x)^2} - 1$	$[-1, 1)$
$\mathcal{S}$	Bipolare Sigmoid Funktion	$2/(1 + e^{-4.9x}) - 1$	$[-1, 1)$
sin	Sinus	$sin(x)$	$[-1, 1]$
cos	Cosinus	$cos(x)$	$[-1, 1]$
1.0	Konstant 1	1	$[1, 1]$

Abbildung 5.1.: Aktivierungsfunktionen des CPPN

Ein- und Ausgänge des CPPN Wir verwenden ein CPPN mit sechs Eingängen und vier Ausgängen (siehe Abbildung 5.2). Da die Neuronen im dreidimensionalen

Algorithmus 6 Ablauf des evolutionären Algorithmus (NSGP/NSGA-II) zur Optimierung von gepulsten neuronalen Netzen

1. Eine vollständig bewertete Population der Grösse $\mu+\lambda$ liegt vor. Diese besteht aus der Vereinigung von Eltern- und Nachkommenpopulation (Punkt 10).

2. Durch nicht-dominierendes Sortieren anhand der drei Kriterien Erkennungsrate, Behavioral Diversity und Connection Cost werden die Pareto-Fronten F_i bestimmt.

3. Jede Pareto-Front F_i wird nochmals in Gruppen G_{ij} unterteilt, dabei enthält Gruppe G_{ij} alle Individuen der Front F_i, die auf den selben Punkt p_j im Objektivraum fallen (d.h. mit gleicher Fitness).

4. Für die Bildung der Nachkommen wird die Elternpopulation wie folgt ausgewählt: Begonnen wird mit der ersten Front F_0. Aus jeder Gruppe G_{0j} wird zufällig ein Individuum entfernt und in die neue Population übernommen. Aus den folgenden Fronten F_i werden durch Anwendung des gleichen Verfahrens Individuen ausgewählt, bis insgesamt μ Individuen gefunden wurden.

5. Die mittlere Crowding-Distanz wird berechnet (siehe Abschnitt 5.2.5.1).

6. Aus der Elternpopulation werden Individuen paarweise zur Nachkommensbildung mit Tournament der Grösse k und Phenotyp-uniformes Sampling (siehe Abschnitt 5.2.5.1) ausgewählt. Als Ordnung dient der Pareto-Rang und die mittlere Crowding-Distanz.

7. Durch Mutation oder Kreuzung werden λ Nachkommen erzeugt. Hierbei wird darauf geachtet, dass keine Genome entstehen, die bereits aufgetreten sind.

8. Die Nachkommen werden wie folgt bewertet:

   - Durch Auswerten des im Genom enthaltenen CPPN wird dessen *Verhaltens-Bitvektor* bestimmt.
   - Aus dem CPPN wird ein neuronales Netz (NN) erzeugt. Bei der Erzeugung wird die *Connection Cost* des Netzes berechnet. Anschliessend wird das NN mit Testdaten gefüttert und simuliert. Nach dem Einlernen wird die *Erkennungsrate* anhand weiterer Testdaten ermittelt.

9. Die *Behavioral Diversity* wird für alle Individuen der Population bestimmt. Dieses Maß ist, genauso wie die Crowding-Distanz, abhängig von den anderen Individuen der Population und muss daher für jede Generation neu berechnet werden.

10. Die bewerteten Nachkommen werden mit der Elternpopulation vereinigt. Diese enthält nun $\mu + \lambda$ Individuen.

11. Sofern die Zielvorgabe, z.B. eine bestimmte Erkennungsrate, nicht erreicht wurde, beginne wieder mit Schritt 1.

Raum platziert sind, besteht die Eingabe aus den Koordinatenvektoren $(x1, y1, z1)$ und $(x2, y2, z2)$. Die vier Ausgänge sind:

- Verzögerungszeit der Synapse t.
- Effizienz der Synapse w (optional).
- Link Expression Output ex.
- Neuronenparameter r.

Der Link Expression Output gibt an, ob eine Verbindung zwischen den zwei, durch die Eingänge bestimmten, Neuronen ausgeprägt werden soll (> 0) oder nicht (≤ 0). Dies dient der Entkopplung von Gewichtung und Verbindungsbildung (Topologie) und ermöglicht dünnbesetzte Netze [VS11].

Um den Neuronenparameter r zu bestimmen, wird der zweite Koordinatenvektor $(x2, y2, z2)$ auf Null gesetzt.

Diese sechs Eingänge und vier Ausgänge sind an fest vorgegebene Knoten gekoppelt. Diese sind in der Abbildung 5.2 grau unterlegt, und bilden die Eingabe- bzw. Ausgabeschicht. Bei den Eingängen wird jeweils eine lineare Funktion verwendet, während bei den Ausgängen, bis auf den Link Expression Output, eine bipolare Gaussche Funktion zum Einsatz kommt. Letzteres dient der Beschränkung des Wertebereiches auf $[-1, 1]$. Ein weiterer fest vorgegebener Knoten ist der sog. Bias-Knoten. Dieser liefert ein konstantes Ausgabesignal von 1.0. Die in Abbildung 5.2 zwischen Ein- und Ausgabeschicht dargestellten Knoten (verdeckte Schicht) werden von dem evolutionären Algorithmus erzeugt und verändert. Ebenso die Verbindungen (gestrichelt) die zwischen den Knoten der verdeckten Schicht und den anderen Schichten realisiert werden.

5.3.3. Konfiguration des Substrates

Für die Positionierung der Neuronen auf dem Substrat verwenden wir Koordinaten im 3-dimensionalen Raum. Die z-Koordinate beschreibt hierbei immer die Schicht in der sich ein Neuron befindet. Neben der Eingangsschicht ($z = 0$) und der Ausgangsschicht ($z = n - 1$) verwenden wir abhängig von der zu lösenden Problemstellung h weitere verdeckte Schichten ($n = 2 + h$). Die x und y-Koordinaten dienen der Platzierung innerhalb einer Schicht. In unseren Beispielen verwenden wir allerdings nur die x und z-Koordinate.

In Abbildung 5.3 ist ein Beispiel für die Platzierung von Neuronen in drei Schichten zu sehen. Abbildung 5.4 zeigt ein weiteres Beispiel, in dem 15 Neuronen auf drei verdeckte Schichten aufgeteilt werden.

Da die Aufteilung der Neuronen nur anhand der z-Koordiante erfolgt, halten wir zudem noch getrennt von der Position für jedes Neuron fest, ob es sich um ein Eingabe-, Ausgabe- oder ein verdecktes Neuron handelt. Dies erleichtert später eine Zuordnung

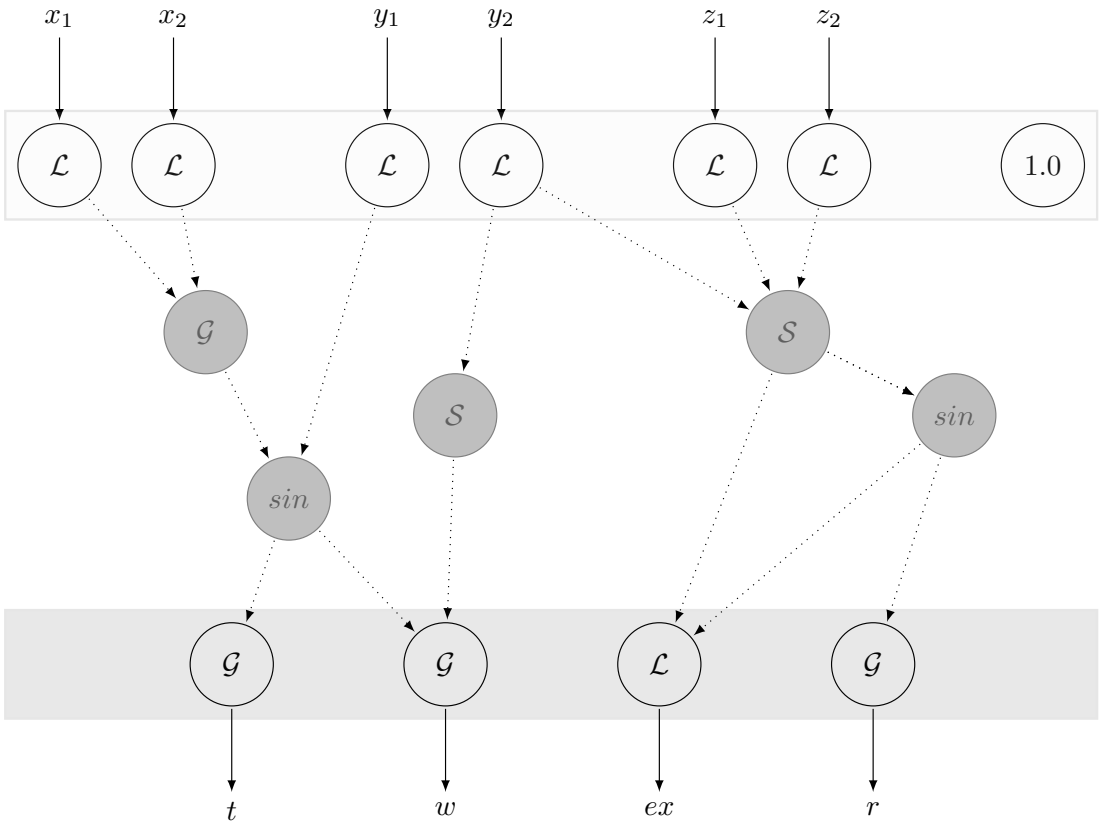


Abbildung 5.2.: Ein- und Ausgaben in das CPPN. Koordinate des Quellneurons: $(x1, y1, z1)$. Koordinate des Zielneurons (bei Verbindungsbildung): $(x2, y2, z2)$. Verzögerungszeit der Synapse: t. Effizienz der Synapse: w. Link Expression Output: ex. Neuronenparameter: r.

der Ein- und Ausgänge des neuronalen Netzes zu den entsprechenden Neuronen. Bei der Zielgraphsuche ist diese Information bereits im Zielgraph enthalten und dient der Berechnung der Anzahl Neuronen bzw. Knoten pro Schicht.

Die genaue Konfiguration des Substrats, d.h. die Anzahl der Schichten und die Verteilung der Neuronen innherhalb der Schichten hängt dabei stark von der zu lösenden Problemstellung ab. Im Falle der Zielgraphsuche teilen wir die Knoten analog zu Abbildung 5.3 in drei Schichten auf.

Weitere mögliche Anordnungen sind beispielsweise:

- Auf der Kreisfläche, z.B. dem Vorbild der Sonnenblume folgend [Vog79] (siehe Abbildung 5.5).
- Auf beliebigen Oberflächen durch Poisson Disk-Sampling [Bri07].
- Durch Verwendung einer Quasi-Zufallsfolge, z.B. SOBOL [Sob] oder Nierreiter [Nie88].

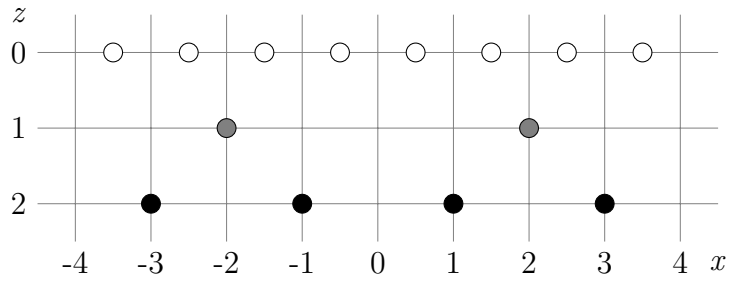


Abbildung 5.3.: Platzierung von 8 Eingabeneuronen, 2 inneren Neuronen und 4 Ausgabeneuronen auf einem Substrat mit 3 Schichten.

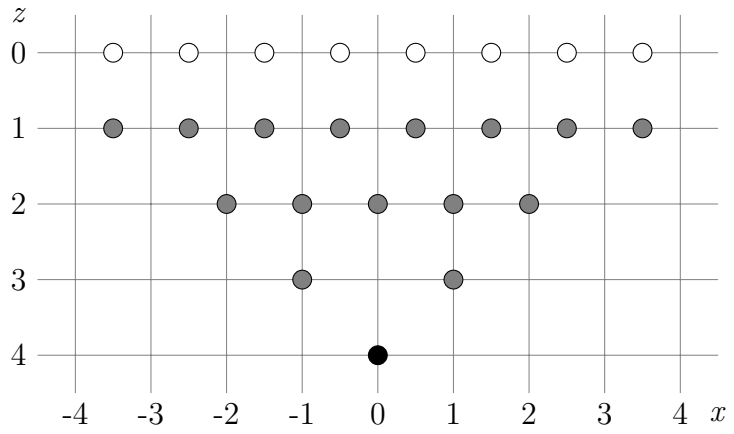


Abbildung 5.4.: Platzierung von 8 Eingabeneuronen, 15 inneren Neuronen und 1 Ausgabeneuron auf einem Substrat mit 5 Schichten.

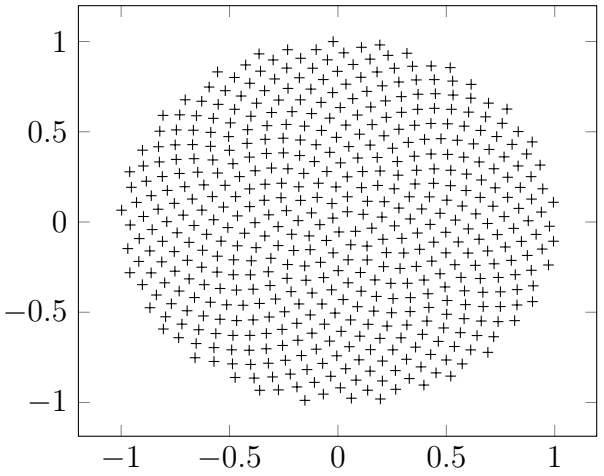


Abbildung 5.5.: Platzierung von 500 Neuronen innerhalb eines Kreises unter Verwendung des Sonnenblumen-Layouts [Vog79].

5.3.4. Genetische Operatoren

Folgende genetische Operatoren kommen zum Einsatz:

- Strukturelle Mutation
- Mutation der Gewichtung
- Kreuzung der Gewichtung passender Kanten mit Werteangleich

5.3.4.1. Strukturelle Mutation

Bei der strukturellen Mutation kann zwischen verschiedenen Operationen ausgewählt werden. Diese sind:

AddNode: Eine existierende Kante wird durch einen neu erzeugten Knoten mit zufälliger Aktivierungsfunktion in der Mitte getrennt. Die neu entstehende zweite Kante bekommt die gleiche Gewichtung zugewiesen, wie die existieren Kante.

DropNode: Entfernt einen Knoten. Durch k-maliges Losen wird der Knoten mit dem geringsten Knotengrad ausgewählt. Alle ein- und ausgehenden Kanten des Knotens werden zudem gelöscht.

ModifyNode: Verändert die Aktivierungsfunktion eines Knotens.

Connect: Verbindet zwei bislang nicht direkt miteinander verbundene Knoten, ohne dass dabei ein Zyklus im Netz entsteht. Die Wahl der Knoten erfolgt zufällig, es werden jedoch Knoten mit geringerem Knotengraden bevorzugt. Dies wird durch k-maliges Losen realisiert. Das Gewicht der Kante wird zufällig aus dem erlaubten Bereich gewählt.

Disconnect: Entfernt eine existierende Kante.

SymmetricFork: Eine Kante wird zufällig ausgewählt, und eine weitere Kante, die dem gleichen Startknoten entspringt, mit negierter Gewichtung dem Netz hinzugefügt, falls dadurch kein Zyklus entsteht.

SymmetricJoin: Analog zu SymmetricFork, nur dass hierbei der Zielknoten statt des Startknotens mit der zufällig ausgewählten Kante übereinstimmt.

Jede dieser Operationen wird mit einer konfigurierbaren Wahrscheinlichkeit ausgewählt. Ist eine Operation nicht durchführbar, beispielsweise wenn *Disconnect* auf einem Netz ohne Kanten angewendet wird, so wird eine andere Operation zufällig ausgewählt.

5.3.4.2. Mutation der Gewichtung

Bei der Mutation der Gewichtung eines Genoms wird jede Kante mit der Wahrscheinlichkeit p_{mutEl} verändert, mindestens jedoch eine Kante. Das Gewicht einer zu verändernden Kante wird hierbei um eine Zufallszahl der Normalverteilung $\mathcal{N}(0.0, \sigma)$ nach oben oder unten modifiziert.

5.3.4.3. Kreuzung der Gewichtungen

Für die Ermittung der übereinstimmenden Kanten zweier CPPNs A und B[14] setzen wir *Neighbor Matching* [Nik12] ein. Dieses Verfahren wenden wir auf den zwei CPPNs der zu kreuzenden Genome an und erhalten eine Ähnlichkeitsmatrix $\sigma_{i,j} \in [0, 1]$ zwischen den Knoten $i \in A$ und $j \in B$. Durch Anwendung der Kuhn-Munkres Methode [Kuh55][15] finden wir zudem eine optimale Zuordnung $\phi : A \to B$ der Knoten aus A zu den Knoten aus B, mit der Eigenschaft, dass die Summe der Knotenähnlichkeiten dieser Zuordnung maximal ist.

Da bei einem CPPN zwischen einem beliebigen Knotenpaar maximal eine Kante existiert, so können wir aus der Zuordnung der Knoten direkt auf die Zuordnung der Kanten schliessen. Sei (l, k) eine Kante aus A, so ist die aus B zugeordnete Kante durch $(\phi(l), \phi(k))$ bestimmt, falls diese im Graph existiert. Die Ähnlichkeit dieser beiden zugeordneten Kanten definieren wir zudem als das Produkt der Knotenähnlichkeiten $\sigma_{l,\phi(l)}\sigma_{k,\phi(k)} = \xi_{(l,k)}$.

Weiterhin verwenden wir bei dem *Neighbor Matching* eine Färbung der Knoten, sodass zwei Knoten nur dann eine Ähnlichkeit > 0 aufweisen, wenn sie in ihrer Aktivierungsfunktion übereinstimmen.

Sei (l, k) eine Kante aus A, und $(\phi(l), \phi(k))$ die aus B zugeordnete existierende Kante. Bei der Kreuzung verändern wir mit einer Wahrscheinlichkeit von $p_{xEl} \cdot \xi^2_{(l,k)}$ die Gewichtung der Kante (l, k), ansonsten bleibt die Gewichtung erhalten. Die Veränderung der Gewichtung erfolgt durch *Simulated Binary Crossover* (SBX) [DJ11], und zwar durch eine elternzentrische Kreuzung (parent centric crossover). Die Wahrscheinlichkeit p_{xEl}[16] ist hierbei konfigurierbar und definiert die maximale Häufigkeit mit der eine kantenweise Gewichtsveränderung durchgeführt wird. Zudem verwenden wir eine *Cut-off* Ähnlichkeit ξ_{cutoff}. Ist $\xi_{(l,k)} < \xi_{cutoff}$ so findet keine Veränderung der Gewichtung statt.

Findet durch die Kreuzung keine Gewichtsveränderung statt, so wird zudem eine Mutation der Gewichtung, wie in Abschnitt 5.3.4.2 beschrieben, durchgeführt.

[14] *Gene Alignment*
[15] Ungarischer Algorithmus
[16] *xEl* steht für *xover* (crossover) *Element*

5.3.5. Bewertungsfunktion

Die Bewertungsfunktion besteht aus drei verschiedenen Kriterien:

- *Problemspezifische Fitness:* Zum Beispiel die Erkennungsrate bei gepulsten neuronalen Netzen, oder die Ähnlichkeit zu einem gegebenen Graph bei der Zielgraphsuche.
- *Behavioral Diversity:* Dieses Maß wird auf den CPPNs der Population berechnet und gibt an, wie sehr sich diese in ihrem Verhalten unterscheiden. Eine hohe Diversität in dem Verhalten der CPPNs der gesamten Population fördert eine bessere Exploration des Suchraumes. Zudem dient dieses Kriterium dem Schutz neuer Innovationen.
- *Connection Cost:* Dieses Maß wird bei der Erzeugung eines Netzes durch das CPPN berechnet und dient der Bevorzugung modularer Strukturen. Hierzu wird das Quadrat der Längen aller Verbindungen aufsummiert.

In HyperNEAT-CCT fliesst die Connection Cost nur zu 25% mit in die Bewertung ein. Hierfür wird PNSGA-II verwendet, eine probabilistische Variante von NSGA-II. Da sich die von NSGA-II verwendete Dominanz-Relation (siehe Definition definition 1) aufgrund der benötigten Transitivität bei der Berechnung der Pareto-Fronten nicht auf triviale Art und Weise in eine probabilistische Dominanz-Relation umwandeln lässt, haben wir ganz auf die Implementierung von PNSGA-II verzichtet. Denkbar wäre hingegen, die Bewertung der Connection Cost nur bei jeder vierten Iteration mit hinzuzunehmen. Die Hinzunahme weiterer Kriterien, abhängig von der Problemstellung ist denkbar. Teilweise implementiert haben wir die folgenden zusätzlichen Kriterien:

- *Saturation:* Dieses Kriterium bewertet Übersättigungen der Aktivierungsfunktionen der Knoten des CPPNs. Hierbei wird für jede mögliche Eingabe, die Anzahl der aufgetretenen Übersattigungen in Abhängigkeit zu der Anzahl der Knoten gezählt.
- *Complexity:* Dieses Kriterium zählt die Anzahl der Knoten des CPPNs und soll weniger komplexe CPPNs bevorzugen.
- *Age Diversity:* Dieses Kriterium bestimmt die Abweichung des Alters eines Individuums vom Altersdurschnitt der gesamten Population. Ältere oder jüngere Individuen in Bezug zum Durchschnitt werden somit bevorzugt.

Da NSGA-II nicht geeignet ist für die Verwendung von vielen Kriterien, können einzelne Kriterien wahlweise hinzugenommen werden oder durch andere ersetzt werden.

5.3.5.1. Problemspezifische Fitness

Bei den gepulsten neuronalen Netzen verwenden wir die Erkennungsrate als problemspezifische Fitness. Diese definiert sich aus dem Verhältnis der korrekt erkannten Eingaben K zu der Gesamtzahl aller getesteter Eingaben N:

$Erkennungsrate = \frac{K}{N}$

Bei der Zielgraphsuche haben wir zwei verschiedene Verfahren zur Bewertung der Ähnlichkeit von Graphen eingesetzt:

- *Neighbor Matching* [Nik12]
- Triadischer Zensus gerichteter Graphen [BM01]

Neighbor Matching Neighbor Matching vergleicht zwei Graphen A und B und ermittelt die Ähnlichkeit zwischen Knoten der beiden Graphen in Form einer $m \times n$ Ähnlichkeitsmatrix. Die Elemente der Matrix liegen zwischen 0 (keine Ähnlichkeit) und 1 (isomorph). Um aus der Ähnlichkeitsmatrix eine einzelne Maßzahl zu berechnen, lässt sich durch Anwendung der Kuhn-Munkres Methode [Kuh55] eine optimale Zuordnung $\phi : A \to B$ der Knoten aus A zu den Knoten aus B finden, mit der Eigenschaft, dass die Summe der Knotenähnlichkeiten dieser Zuordnung maximal ist. Wird die Summe mit $min(m, n)$ oder $max(m, n)$ normiert, so ergibt sich wieder ein Wert in $[0, 1]$. Eine Normierung mit $max(m, n)$ bestraft das Fehlen von Knoten, während dies bei $min(m, n)$ nicht der Fall ist. Wir haben uns für die Normierung mit dem Maximum entschieden.

Neighbor Matching berücksichtigt allerdings nicht die Kantengewichtung der beiden Graphen. Um dennoch Unterschiede in der Kantengewichtung mit in das Ähnlichkeitsmaß einfliessen zu lassen, haben wir Neighbor Matching wie in Definition definition 5 beschrieben erweitert. Dies erlaubt uns eine Bewertung eines evolutionär entwickelten Graphen zu einem gegebenen Zielgraph mit oder ohne Gewichtung durchzuführen.

Definition 5 (Kantenähnlichkeit zweier Graphen) *Aus Gründen der Einfachheit gehen wir davon aus, dass beide Graphen A und B denselben Knotengrad haben, d.h. $m = n$ gilt. Ausgangspunkt ist die Ähnlichkeitsmatrix $\sigma_{i,j} \in [0, 1]$ zwischen den Knoten $i \in A$ und $j \in B$. Desweiteren sei $\phi : A \mapsto B$ eine optimale bijektive Zuordnung der Knoten aus A zu den Knoten aus B, mit der Eigenschaft, dass die Summe der Knotenähnlichkeiten dieser Zuordnung $\sum_i \sigma_{i,\phi(i)}$ maximal ist.*

Zwischen jeweils einem zugeordneten Knotenpaar $i \in A$ und $\phi(i) \in B$ berechnen wir nun die Kantenähnlichkeit $\chi_{i,j}$ nach Definition definition 6.

Die Kantenähnlichkeit zweier Graphen A und B definieren wir nun als:

$$\chi_{A,B} = \sum_i \frac{\chi_{i,\phi(i)}}{max(m, n)}$$

Definition 6 (Kantenähnlichkeit zweier Knoten) *Gegeben seien zwei Knoten i und j mit ausgehenden Kanten $(e_{i,1}, \ldots, e_{i,m})$ und $(e_{j,1}, \ldots, e_{j,n})$. Die Gewichtung der Kanten sei durch $w(e) \mapsto [0, 1]$ bestimmt.*

Wir initialisieren eine $max(m,n) \times max(m,n)$ Matrix Δ, die alle Kombinationen der Differenzen der Kanten von i und j enthält, wie folgt:

$$\Delta_{k,l} = \begin{cases} |w(e_{i,k}) - w(e_{j,l})| & \textit{falls } k \leq m \wedge l \leq n \\ 1 & \textit{ansonsten} \end{cases}$$

Mittels der Kuhn-Munkres Methode suchen wir eine optimale bijektive Zuordnung ϕ_Δ, sodass $\sum_k \Delta_{k,\phi_\Delta(k)}$ minimal ist.

Die Kantenähnlichkeit $\chi_{i,j} \in [0,1]$ der Knoten i und j definieren wir dann als:

$$\chi_{i,j} = 1 - \sum_{1 \leq k \leq max(m,n)} \frac{\Delta_{k,\phi_\Delta(k)}}{max(m,n)}$$

Triadischer Zensus Während Neighbor Matching ein relatives Mass für die Ähnlichkeit zweier Graphen zueinander darstellt, so handelt es sich bei dem Triadischen Zensus um eine absolute Metrik. Hierbei wird die Häufigkeit der 16 möglichen in Abbildung 5.6 dargestellten Motive, die bei gerichteter Verbindung von drei Knoten entstehen können, gezählt. Als Ergebnis erhält man für jeden Graph einen Vektor $\vec{c} = (c_1, c_2, \ldots, c_{16})$. Das Maß der Ähnlichkeit zwischen zwei Graphen A und B lässt sich somit leicht als Euklidsche Distanz zwischen $\vec{c_A}$ und $\vec{c_B}$ definieren. Für die Bestimmung des Triadischen Zensus verwenden wir den in [BM01] dargestellten subquadratischen Algorithmus.

5.3.5.2. Behavioral Diversity

Um die Behavioral Diversity zu bestimmen, wird zuerst der Verhaltens-Bitvektor eines CPPN ermittelt. Hierfür legen wir jede mögliche Eingabe an das CPPN an und ermitteln die zugehörige Ausgabe. Jede dieser Ausgaben o wird in ein Bit b des Bitvektors umgewandelt mit $b = 1 \iff o \geq 0$ ansonsten gilt $b = 0$. Da unsere CPPNs vier Ausgänge verwenden, erhalten wir für jede Eingabe vier Ausgabenwerte. Diese schreiben wir sequentiell in den Bitvektor.

Ist für jedes Genom der Population der Verhaltens-Bitvektor bestimmt, so kann daraus die Behavioral Diversity berechnet werden. Diese ist als mittlere Hamming-Distanz des Bitvektors eines Genoms zu allen anderen Bitvektoren der gesamten Population definiert (siehe Gleichung Abschnitt 5.2.4.2).

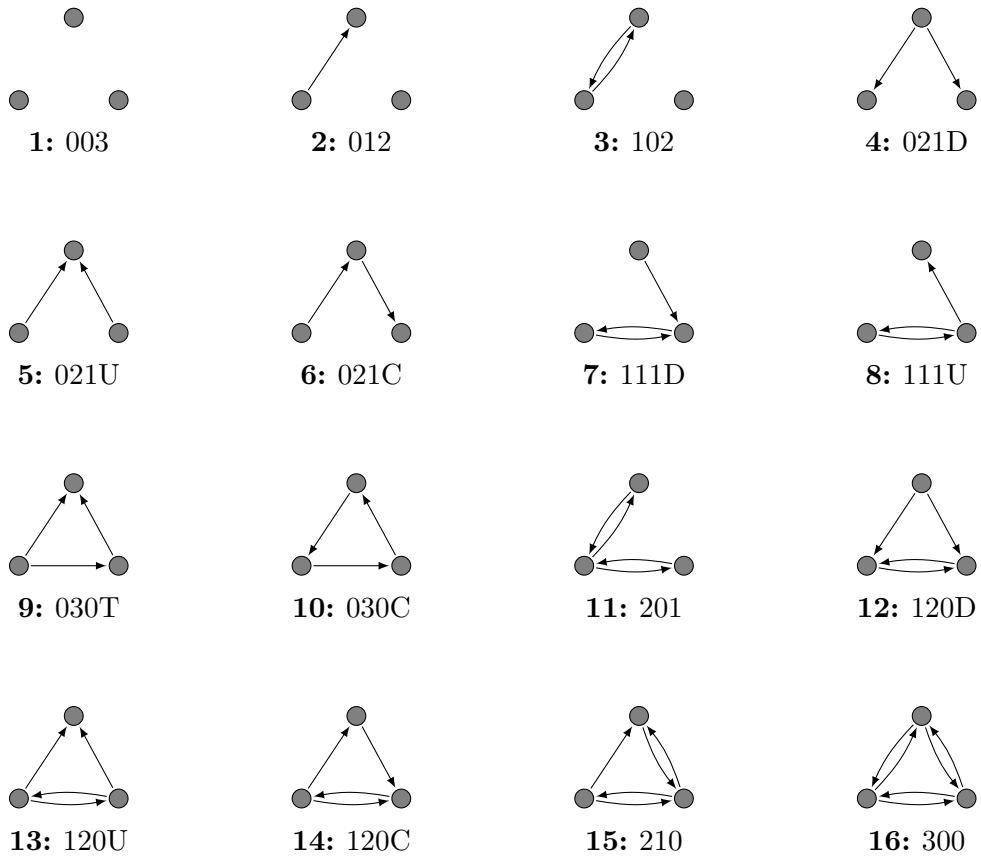


Abbildung 5.6.: Motive des gerichteten Triadischen Zensus

5.3.6. Konfiguration der Gepulsten Neuronalen Netze

Folgende Einstellungen bzw. Entscheidungen müssen getroffen werden, bevor mit der Evolution begonnen werden kann:

- *Konfiguration des Substrates.*
- *Wahl der Kodierung* für die Eingaben und Ausgaben des Netzes (Zeitkodierung, Ratenkodierung, ...).
- Der *Testablauf*, d.h. welche Daten für den Einlernvorgang und für die spätere Klassifikation verwendet werden und wie lange und wie oft diese an das Netz angelegt werden.

Die Konfiguration des Substrates orientiert sich dabei an der Komplexität der Ein- und Ausgaben. Bei der Bilderkennung würden wir zum Beispiel für jedes Pixel ein Eingabeneuron verwenden, und jeweils ein Ausgabeneuron für jede mögliche Klassifikation. Die Wahl der Anzahl verdeckter Schichten und Neuronen hingegen gestaltet sich schwieriger und lässt sich letztendlich nur durch Ausprobieren finden.

Für die Kodierung von Bildinformation eignet sich beispielsweise die Abbildung eines schwarzen Pixels auf einen konstanten Eingabestrom. Besteht die Ausgabe nur aus einem Neuron, so lässt sich mit dem Netz anhand des Feuerzeitpunktes erkennen, ob die entsprechende Eingabe erkannt wurde oder nicht.

Nach der Wahl der oben genannten Einstellungen, kann mit der Evolution begonnen werden. Aus der Auswertung der CPPNs erhalten wir nun alle benötigten Informationen, die zur Erzeugung eines gepulsten neuronalen Netzes erforderlich sind:

- Der Neuronentyp ist für jedes Neuron durch den Ausgang $r \in [-1, 1]$ des CPPNs gegeben. Wir bilden negative Werte auf inhibitorische Neuronen ab, positive auf erregende Neuronen. Dabei bestimmt $|r|$ das genaue Verhalten des Neurons, wie in Abschnitt 2.1.7 definiert.
- Für jede auszubildende Synapse ist deren Verzögerungszeit t und die optionale Effizienz w durch die gleichnamigen Ausgänge des CPPNs gegeben. Es muss lediglich t in den Bereich $[1, 50]\,ms$, und w in den Bereich $[-10.0, 10.0]$ transformiert werden.
- Das Substrat enthält zudem die Information, welches Neuron ein Eingabe- bzw. ein Ausgabeneuronen ist.

6. Auswertung

Wir haben das von uns in Kapitel 5 entwickelte Verfahren anhand der Annäherung an einen vorgegebenen Zielgraph getestet. D.h. die von unserem Verfahren erzeugten Graphen werden nicht erst in gepulste neuronale Netze überführt, um dann simuliert und bewertet zu werden, sondern die erzeugten Graphen werden direkt anhand der Ähnlichkeit zu einem fest vorgegebenen Zielgraph bewertet. Dies hat gegenüber der Simulation den Vorteil, dass sich bestimmte Eigenschaften unseres Verfahrens besser veranschaulichen und untersuchen lassen können.

Hierbei gehen wir von der Annahme aus, dass die Annäherung an einen vorgegebenen Zielgraph ein mindestens ebenso schweres Problem für unser Verfahren darstellt, wie die Entwicklung eines gepulsten neuronalen Netz mit anschliessender Simulation und Bewertung anhand der Erkennungsrate. Wenn zudem unser Verfahen in der Lage ist, ein bereits existierendes und für gut befundenes gepulstes neuronales Netz, welches uns in Form eines gewichteten Graphen vorliegt, evolutionär anzunähern, so folgern wir daraus, dass unser Verfahren dieses Netz auch unter Verwendung der Erkennungsrate als Fitnessfunktion hätte finden können. Ferner gehen wir sogar davon aus, dass die Annäherung an einen Zielgraph schwerer ist als die Entwicklung eines gepulsten neuronalen Netzes, unter anderem auch deshalb, weil wir annehmen, dass die gepulsten neuronalen Netze auch aus einer mittelmässigen Topologie durch den Einlernvorgang noch eine gute Erkennungsrate erzielen können, während es bei der Annäherung an einen Zielgraph, abgesehen von Ungenauigkeiten der Graphenähnlichkeit, keinen Spielraum gibt. Der experimentelle Nachweis dieser These bzw. die Validierung der hier erzielten Ergebnisse anhand der Simulation von gepulsten neuronalen Netzen steht noch aus.

6.1. Einstellungen

Für alle Versuche verwenden wir, falls nicht anders beschrieben, die in Tabelle 6.1 beschriebenen Einstellungen. Die Platzierung der Neuronen haben wir in drei logische Schichten unterteilt. Die Neuronen der Eingangsschicht sind bei $z = 1.0$, die der verdeckten Schicht bei $z = 0.0$ und die Neuronen der Ausgangsschicht bei $z = -1.0$ platziert. Bei manchen Versuchen entfällt die verdeckte Schicht. Die x-Koordinate der Neuronen ist von $-n/2$ bis $n/2$, bei n Neuronen pro Schicht, linear verteilt. Die y-Koordinate ist für alle Neuronen 0.0. Verbindungen haben wir, falls nicht anders beschrieben, nur zwischen der logischen Eingangsschicht und der Ausgangsschicht, sowie der verdeckten Schicht und der Ausgangsschicht zugelassen.

Parameter	Wert
Populationsgrösse	50
Anzahl Nachkommen	25
Tournament	2
Neustart nach	250 Iterationen
Link Expression bei	$0.1 \leq ex \leq 0.5$
Linkgewichtung	$[-1.0, 1.0]$
Linkveränderung	normalverteilt mit $\sigma = 0.1$
Mutationsrate	50%
Elementmutationsrate	5%
MutateWeight	100
AddNode	2
DropNode	1
ModifyNode	0
Connect	2
Disconnect	2
SymmetricJoin	2
SymmetricFork	2

Tabelle 6.1.: Einstellungen des Evolutionären Algorithmus

Als anwendungsspezifische Fitnessfunktion kommt Neighbor-Matching zum Einsatz. In einigen Versuchen haben wir zudem unsere Erweiterung des Neighbor-Matchings, dass die Kantengewichtung mitberücksichtigt, verwendet.

6.2. Visualisierung

Abbildung 6.1 zeigt ein Tool das wir zum Austesten verschiedener Einstellungen des evolutionären Algorithmus entwickelt haben. Es lassen sich alle Parameter zur Laufzeit verändern und das Ergebnis *live* beobachten. Die gezeigte Ansicht gibt einen Überblick über einen Teil der Population. In der oberen Hälfte sind die entwickelten Graphen der besten Individuen dargestellt, während die untere Hälfte die zugehörigen CPPNs zeigt. Dies ist insbesondere hilfreich um die Diversität der Population zu visualisieren. Eine weitere Ansicht stellt nur das beste Individum dar. Hiermit lässt sich das CPPN genauer inspizieren.

6.3. Untersuchte Graphen

Die Analyse in [Mil+02] zeigt, dass in komplexen Netzen bestimmte Verbindungsmotive häufiger auftreten als dies in zufälligen Netzen der Fall ist. Zu den untersuchten Netzen gehören unter anderem Genregulationsnetze sowie biologische neuronale

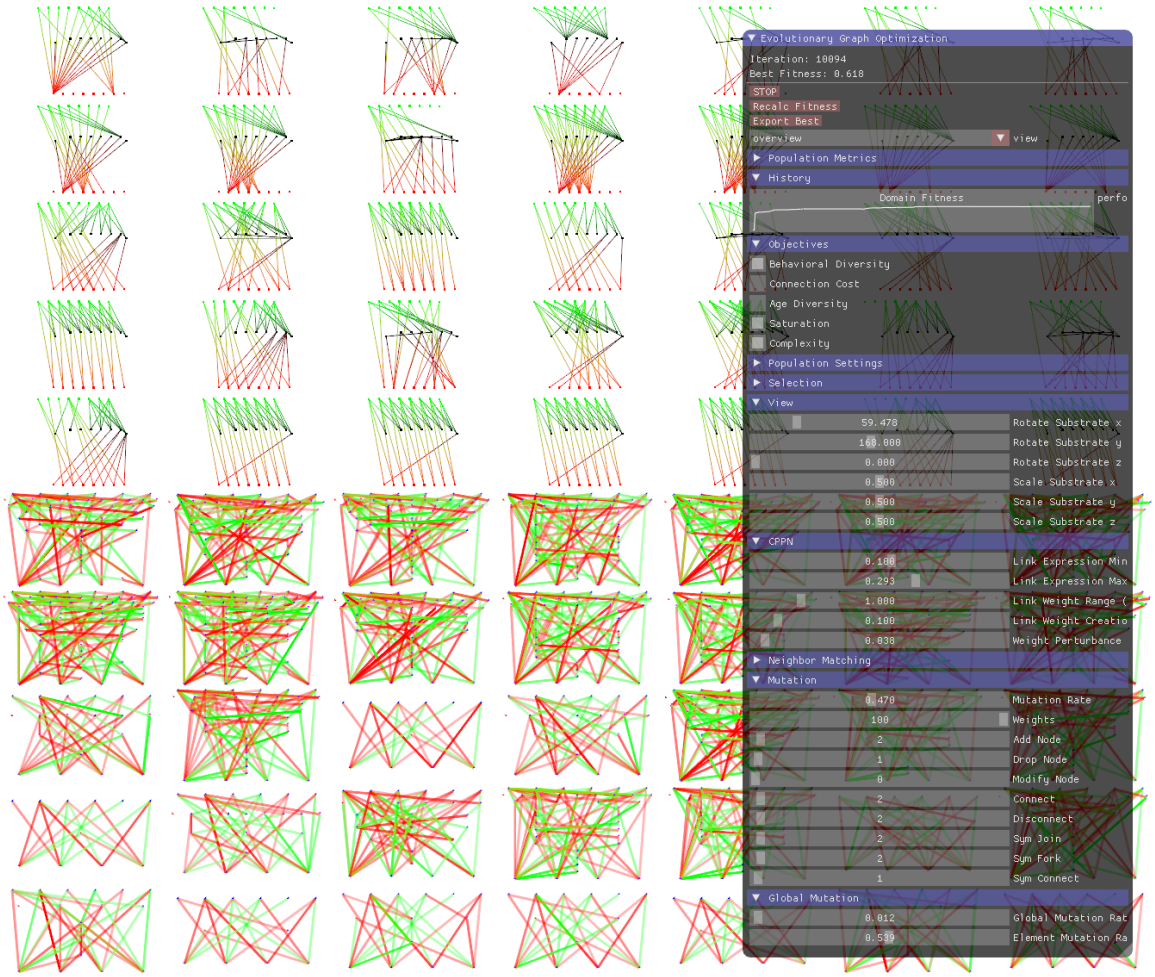
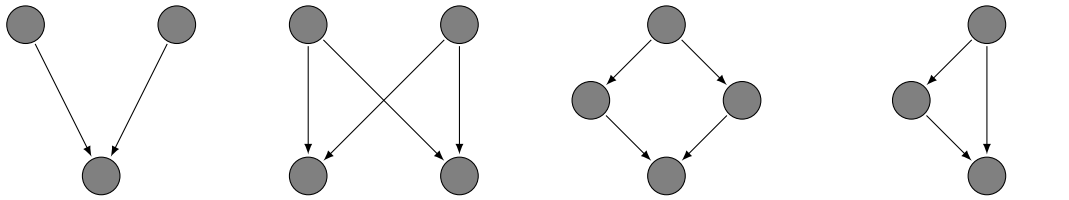


Abbildung 6.1.: Tool zur Visualisierung/Austesten verschiedener Einstellungen.

Netze. Ein ähnliches Ergebnis beschreibt [Luc09] für das Genregulationsnetz von E. coli[1] und [Joh04] bei dem neuronalen Netze von C. Elegans[2]. Zu den häufigsten Motiven zählen die in Abbildung 6.2 dargestellten Motive Bipartiter Graph, Bi-Fan, Bi-Parallel und Feed-Forward Loop (FFL).

In den folgenden Abschnitten testen wir unser Verfahren anhand der Fähigkeit zur Erzeugung dieser Motive bzw. an Graphen die eine Vervielfältigung dieser Motive wie in Abbildung 6.3 dargestellt enthalten. Darüber hinaus untersuchen wir unser Verfahren anhand der Vollverdrahtung, Vollverdrahtung mit Variation in der Gewichtung, sowie an einem komplexeren gepulsten neuronalen Netz, welches die Positionsschätzung des Sandskorpions *Smeringurus mesaensis* nachbildet.

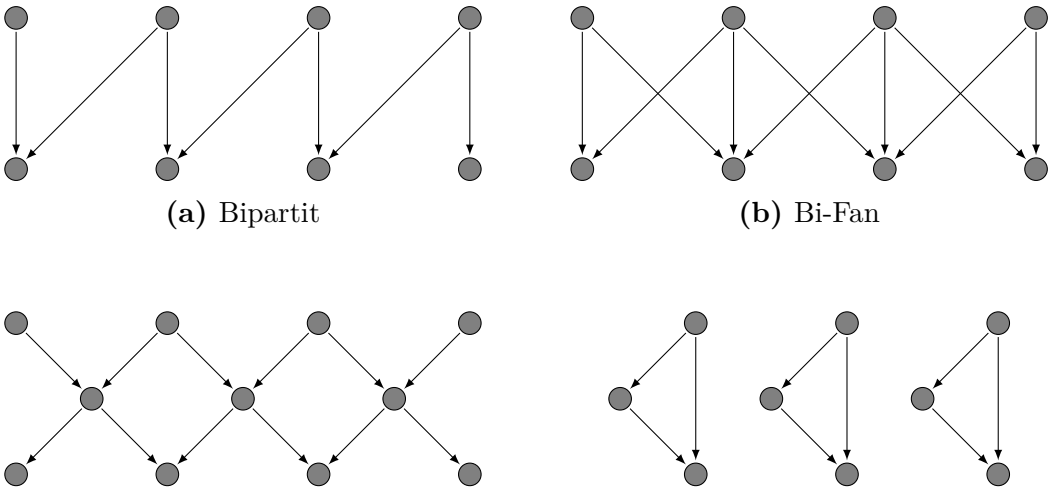
[1] Escherichia coli, ein stäbchenförmiges Bakterium
[2] Caenorhabditis elegans, ein Fadenwurm



(a) Bipartit **(b)** Bi-Fan **(c)** Bi-Parallel **(d)** FFL[a]

[a]Feed-Forward Loop

Abbildung 6.2.: Häufige Motive in biologischen Netzen



(a) Bipartit **(b)** Bi-Fan

(c) Bi-Parallel **(d)** FFL[a]

[a]Feed-Forward Loop

Abbildung 6.3.: Vervielfältigung der Motive aus Abbildung 6.2

6.3.1. Vollverdrahtung

Eine Vollverdrahtung der Neuronen lässt sich mit einem CPPN sehr leicht realisieren. Dazu ist lediglich die Aktivierung des *Link-Expression Outputs* (LEO) nötig. Ist dieser bei ≥ 0.0 aktiv, so realisiert bereits ein unverbundenes CPPN eine Vollverdrahtung aller Neuronen. Aus diesem Grund haben wir für diesen und alle weiteren Versuche die Aktiviung des LEO auf den Bereich zwischen 0.1 und 0.5 begrenzt.

Bei diesem Versuch wurde bereits nach zwei Iterationen ein Ergebnis gefunden, unabhängig von der gewählten Anzahl an zu verbindender Neuronen. Eines der gefundenen Netze verknüpft den Bias-Eingang, der einen konstanten Wert von 1.0 liefert, entsprechend gewichtet mit dem LEO. Ein zweites Netz verknüpft den z_2 Eingang negativ gewichtet mit dem LEO, während ein drittes Netz den z_1 Eingang positiv gewichtet mit dem LEO verknüpft.

Haben wir zudem wechselseitige Verbindungen zwischen beiden Schichten zugelassen, so erhielten wir als Lösung nur ein mögliches Netz. Dieses schaltet z_1 positiv gewichtet auf den LEO. Da z_1 nur bei einem Neuron aus der Eingangsschicht positiv ist, wird somit nur für die Neuronen dieser Schicht Verbindungen zu den Neuronen anderer Schichten ausgebildet.

6.3.2. Vollverdrahtung mit Variation der Gewichtung

Wir haben versucht das in Abbildung 6.4 dargestellte 2-3 Jeffress Netz mit zwei Eingangsknoten und drei Ausgangsknoten und der Gewichtung der Kanten von 0.0 bis 1.0 mit unserem Verfahren anzunähern. Das 2-3 Jeffress Netz stellt in diesem Fall eine Vollverdrahtung beider Schichten dar.

Bereits nach 112 Iterationen konnten wir (wiederholbar) das in Abbildung 6.5 dargestellte CPPN finden, dass den Zielgraph zu 99.5% annähert. Dabei haben wir als Fitnessfunktion unsere Erweiterung des Neighbor-Matchings auf die Kantengewichte verwendet. Die Gewichtung der Kanten wird hierbei durch den t-Ausgang des CPPNs repräsentiert. Dieser ist bei dem gefundenen Netz in Abhängigkeit zu x_2 geschaltet, d.h. die Position der Ausgangsneuronen X, Y oder Z auf dem Substrat. Das erzeugte Netz hat somit fälschlicherweise die Gewichtung 1.0 für die Kante $A \rightarrow X$, während 0.0 erwartet wird. Ebenso vertauscht ist die Gewichtung von $A \rightarrow Z$. Wir führen dies jedoch auf die verwendete Fitnessfunktion zurück, die symmetrisch vertauschte Gewichtungen nicht erkennt und somit die gleiche Ähnlichkeit zuordnet.

6.3.3. Bipartiter Graph

Wir haben vier verschiedene Netzgrössen getestet, mit jeweils 5, 10, 100 und 200 Neuronen pro Schicht. Innerhalb von 250 Iterationen konnten wir wiederholbar eine

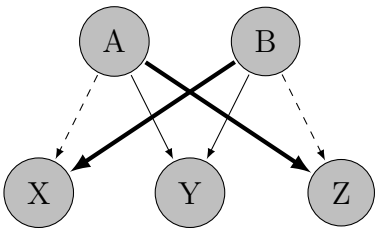


Abbildung 6.4.: 2-3 Jeffress Netz mit Variation in der Gewichtung. $w(A \to X) = w(B \to Z) = 0.0$, $w(A \to Y) = w(B \to Y) = 0.5$ und $w(A \to Z) = w(B \to X) = 1.0$

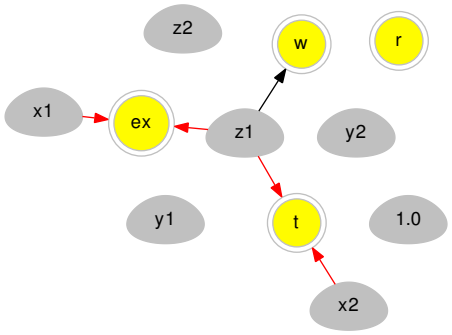
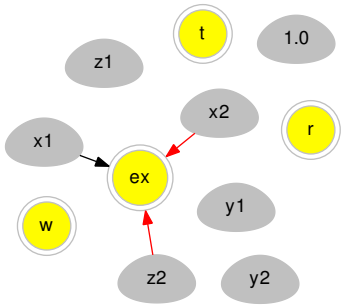


Abbildung 6.5.: CPPN, das gewichtetes 2-3 Jeffress Netz aus Abbildung 6.4 annähert. Iteration 112. Fitness 0.995.

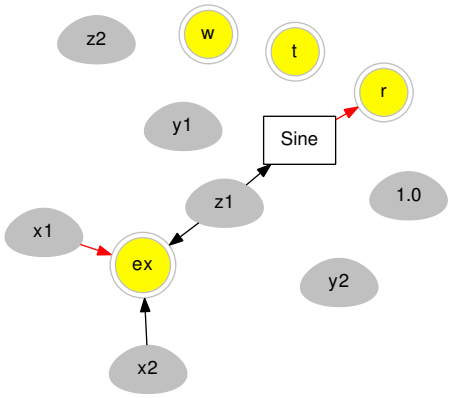
Lösung finden, die dem Zielgraph zu 99.5% entsprach, unabhängig von der verwendeten Netzgrösse. Während wir bei $N = 5$, $N = 10$ und $N = 100$ jeweils einen perfekten Graph erzeugen konnten, so kamen wir bei $N = 200$ nur auf 99.7% Ähnlichkeit. Dies lag an einer fehlenden Verbindung im Randbereich. Allerdings kamen wir bereits schon nach 17 Iterationen zu einer Lösung. Abbildung 6.6 zeigt die vier entstandenen CPPNs. Gemeinsames Merkmal dieser CPPNs ist, dass sie den LEO (*ex* im Bild) mit jeweils vertauschtem Vorzeichen mit x_1 und x_2 verschalten, d.h. eine Verbindung in Abhängigkeit des Abstandes der Neuronen auf der x-Achse realisieren.

Einige der CPPNs enthielten zudem weitere Knoten, wie z.B. einen Sinus-Knoten in Abbildung 6.6b. Diese hatten jedoch keinen Einfluss auf den LEO. Ein Netz bei $N = 5$ (hier nicht abgebildet) hatte sich darüber hinaus speziell an die Netzgrösse angepasst. Festzustellen war eine Erhöhung der Komplexität der Netze (mehr Knoten, mehr Verbindungen) mit fortschreitender Iteration. Dies ist auf die gewählten Wahrscheinlichkeiten für die strukturelle Mutation zurückzuführen, die das Erzeugen von Knoten und Verbindungen wahrscheinlicher macht, als das Entfernen.

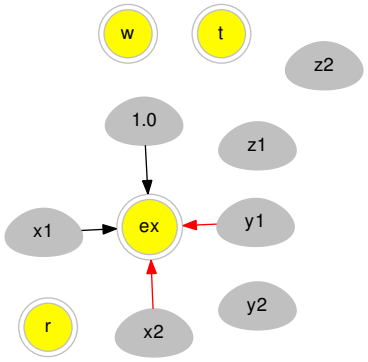
Eine Abhängigkeit der Anzahl benötigter Iterationen zu der Netzgrösse war *nicht* festzustellen.



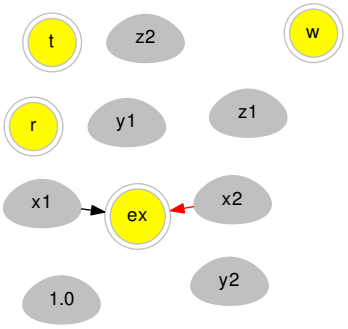
(a) N=5, Iteration 72, Fitness 1.0



(b) N=10, Iteration 75, Fitness 1.0



(c) N=100, Iteration 69, Fitness 1.0



(d) N=200, Iteration 17, Fitness 0.997

Abbildung 6.6.: Gefundene CPPNs bei Annäherung an bipartiten Graph

6.3.4. Bi-Fan

Bei der Annäherung an die Vervielfältigung des Bi-Fan Motives zeigte sich in den erzeugten CPPNs (siehe Abbildung 6.7) ein ähnliches Muster wie bei der Annäherung an einen bipartiten Graph. Während die CPPNs für $N = 5$, $N = 10$ und $N = 100$ (Abbildungen 6.7a bis 6.7c) für den LEO die Formel $-ax_1 + bx_2 + c$ wählten, mit $a, b, c \geq 0.0$, so realisierte das CPPN in Abbildung 6.7d die Formel $-ax_1 - bx_2$, d.h. x_1 und x_2 mit gleichem Vorzeichen. Dies führt dazu, dass der Eingangsknoten ganz links im Substrat mit dem Ausgangsknoten ganz rechts verbunden wird. Topologisch gesehen wird hingegen derselbe Graph erzeugt.

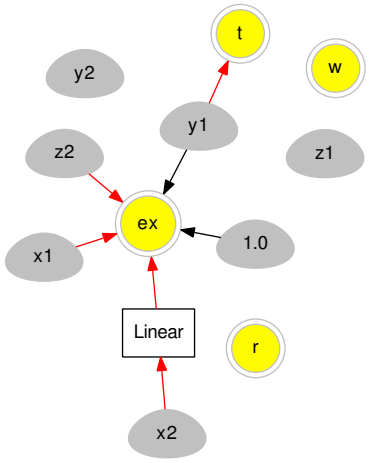
Bei einem x-Abstand der Knoten von 1.0 wird bei dem Bi-Fan Motiv eine Verbindung genau dann realsiert wenn gilt: $|x_1 - x_2| \leq 2.0$. Um die Symmetrie der Absolut-Funktion zu realisieren (dies entspricht der symmetrischen Verbindung nach links und rechts), verwenden einige CPPNs einem Offset c um den Mittelpunkt der Symmetrie zu wählen, während andere eine entsprechende Skalierung beider x-Koordianten verwenden.

6.3.5. Bi-Parallel

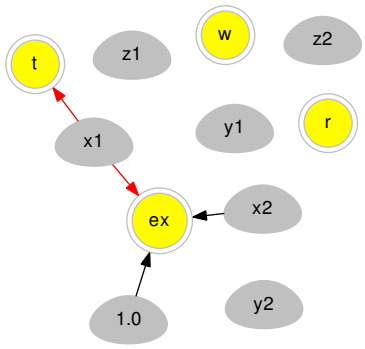
Das Bi-Parallel Motiv ist die Verallgemeinerung des bipartiten Graphen auf mehrere Schichten, in unserem Fall, auf drei Schichten. Wenn wir nur Verbindungen zwischen adjazenten Schichten erlaubten, d.h. keine direkte Verbindung zwischen der Eingangs- und Ausgangsschicht, so konnten wir das Bi-Parallel Motiv mit den gleichen CPPNs annähern die wir bereits für den bipartiten Graph gefunden hatten. Wenn wir diese Restriktion nicht trafen, dann konnten wir innerhalb 250 Iterationen keine Lösung finden. Erst innerhalb 2000 Iterationen war es uns möglich, nach mehrmaligen Versuchen die in Abbildung 6.8 gezeigten CPPNs zu finden. Wie erwartet ist in den so gefundenen CPPNs eine Abhängigkeit zwischen den x und z Koordinaten zu erkennen.

6.3.6. Feed-Forward Loop

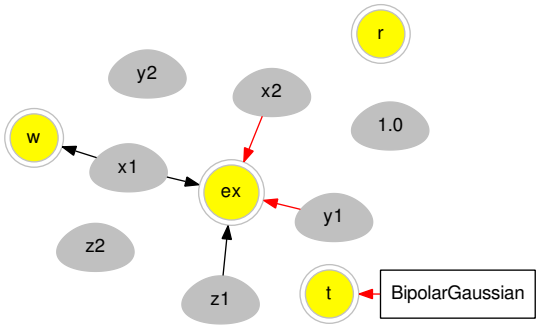
Das Feed-Forward Loop Motiv bei drei Schichten kann als vertikale Vollverdrahtung betrachtet werden, d.h. alle Neuronen mit derselben x-Koordinate werden miteinander verbunden. Unser Verfahren zeigte dasselbe Verhalten wie bei dem bipartiten Graphen. Innerhalb weniger Iterationen konnten wiederholbar, einfache CPPNs gefunden werden. Der LEO wir hierbei in Abhängigkeit der x_1 und x_2 Eingänge geschaltet.



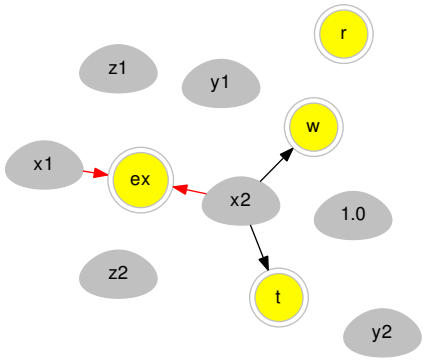
(a) N=5, Iteration 110, Fitness 1.0



(b) N=10, Iteration 83, Fitness 1.0

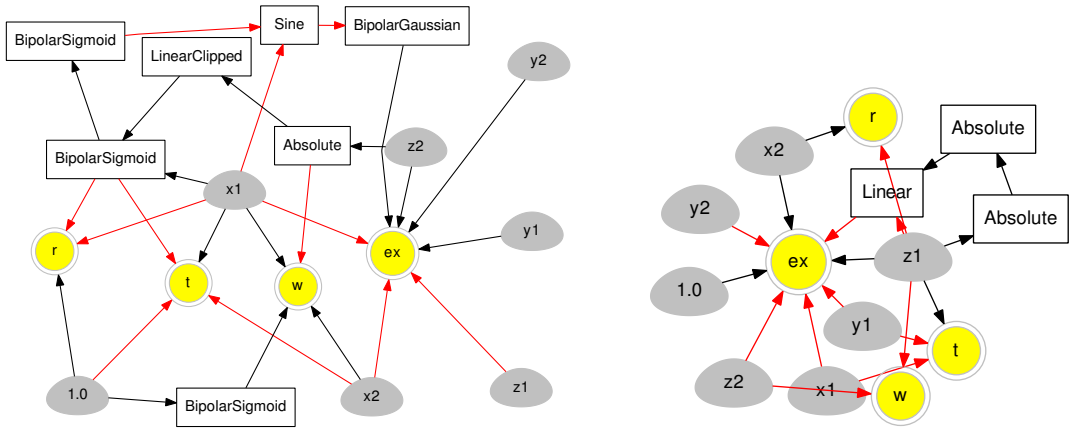


(c) N=100, Iteration 95, Fitness 0.995



(d) N=200, Iteration 67, Fitness 0.995

Abbildung 6.7.: Gefundene CPPNs bei Annäherung an Bi-Fan



(a) Iteration 1749, 3. Versuch **(b)** Iteration 1778, 8. Versuch

Abbildung 6.8.: Gefundene CPPNs bei Annäherung an Bi-Parallel Motiv. Ohne Beschränkung der Verbindungen zwischen logischen Schichten. Neustart bei 2000 Iterationen. N=5. Fitness 1.0

6.3.7. Netz des Sandskorpions Smeringurus mesaensis

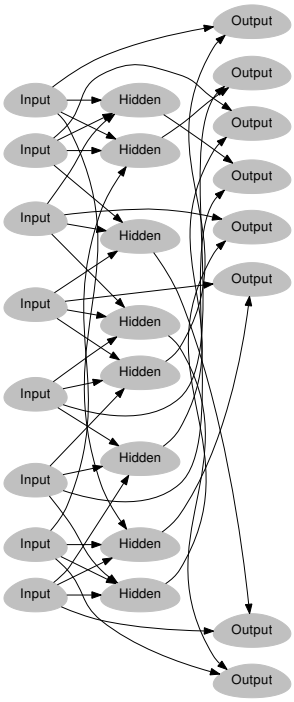
In [Kai04] wurde ein gepulstes neuronales Netz entworfen, dass die Positionsschätzung des Sandskorpions *Smeringurus mesaensis* nachbildet. Wir haben versucht dieses Netz (siehe Abbildung 6.9a) durch unseren Ansatz anzunähern.

Als anwendungsspezifische Fitnessfunktion kam hierbei Neighbor-Matching, unter Berücksichtigung der Kantengewichte, zum Einsatz. Die gewählte Populationsgrösse bei diesem Versuch betrug 100. Wir haben die Simulationen jeweils für 250 Iterationen laufen lassen und danach abgebrochen. Der Versuch wurde 100-mal wiederholt. Dabei haben wir insgesamt drei Netze gefunden, die über 95%-Ähnlichkeit, sowohl in der Topologie als auch in der Gewichtung, zu dem Zielgraph aufwiesen. Bei den restlichen Versuchen blieb der Algorithmus in einem lokalen Maximum gefangen. Eines der gefundenen Netze (siehe Abbildung 6.9b) hatte 97%-Ähnlichkeit zum Zielgraph. Das zugehörige CPPN ist in Abbildung 6.9c abgebildet.

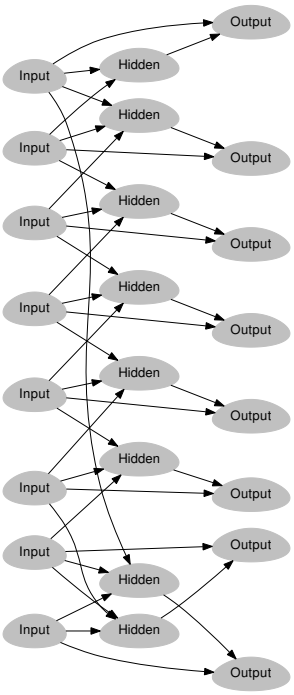
6.4. Laufzeit

Abbildung 6.10 zeigt die Laufzeit unseres Verfahrens pro Iteration in Abhängigkeit zur Netzgrösse. Angenähert wurde das Bi-Fan Motiv. Die verwendete Populationsgrösse betrug 1000, die Anzahl der Nachkommen ebenfalls 1000.

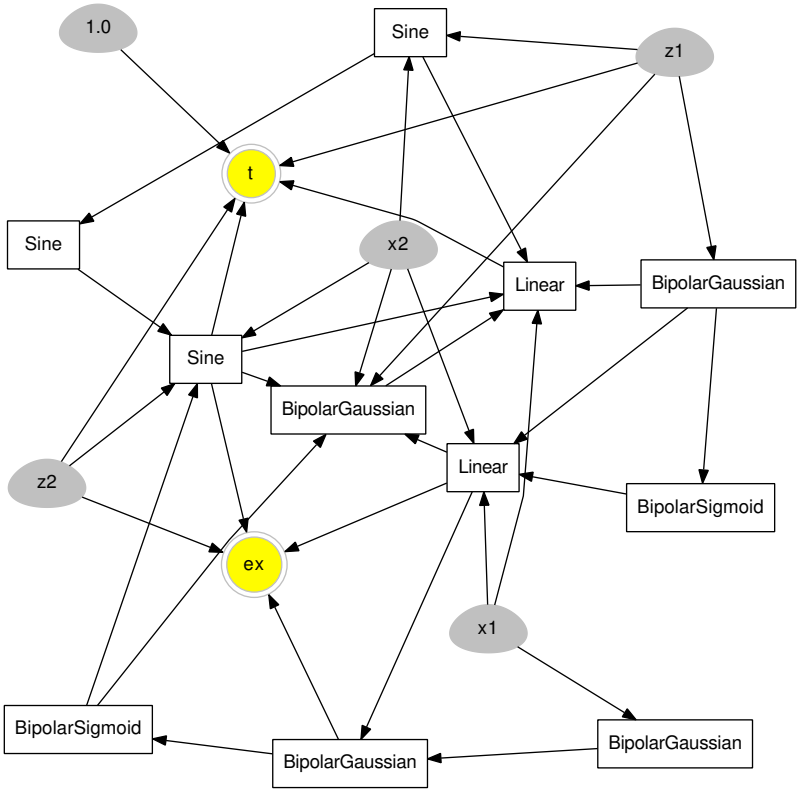
Die gemessene Laufzeit blieb bei allen getesteten Netzgrößen während der gesamten Simulationsdauer nahezu konstant. Die Verdopplung der Netzgrösse von 100 auf 200 erhöhte die durchschnittliche Laufzeit pro Iteration von 1.7 Sekunden auf 20 Sekunden. Dies geht auf die verwendete Fitnessfunktion (Neighbor-Matching) zu-



(a) Zielgraph (Sandskorpion)



(b) Durch CPPN erzeugter Graph



(c) CPPN

Abbildung 6.9.: Annäherung an das Netz des Sandskorpions *Smeringurus mesaensis*

rück, die auf dem Kuhn-Munkres-Algorithmus basiert (Komplexität von $\mathcal{O}(n^4)$ bzw. $\mathcal{O}(n^3)$). Die Simulation von gepulsten neuronalen Netzen hätte hingegen ein lineares Laufzeitverhalten.

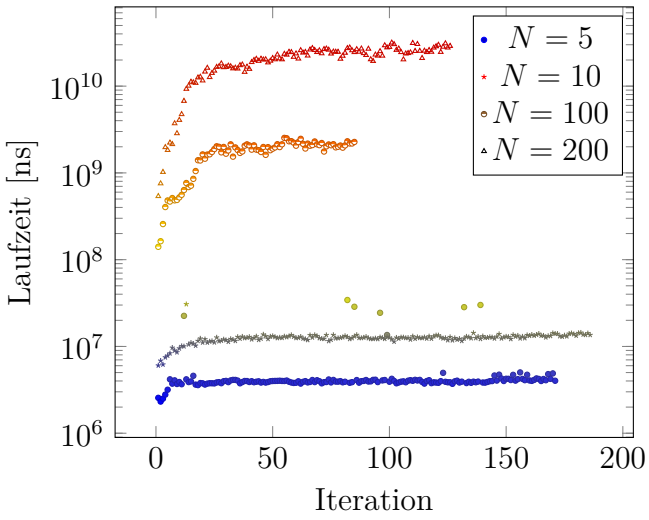


Abbildung 6.10.: Laufzeit in Abhängigkeit von der Netzgrösse N bei der Annäherung an das Bi-Fan Motiv. Populationsgrösse 1000. Anzahl Nachkommen 1000. Hardware Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz. 4 cores/8 threads.

6.5. Fazit

Wir konnten zeigen, dass sich mit unserem Verfahren, basierend auf der Kodierung durch CPPNs, einige in biologischen Systemen vorkommende Graphenmotive evolutionär annähern lassen, unabhängig von der Netzgrösse. Auch die Annäherung der Gewichtung gelang in einem kleinen getesteten Netz. Bei komplexeren Netzen hatte unser Verfahren allerdings erhebliche Probleme. Wir führen dies auf Mängel des evolutionären Algorithmus zurück, nicht auf die Kodierung der CPPNs als solche, da diese zum Teil minimal waren. Eine Mutationsoperation, die eine lokale Optimierung der Individuen durchführt, ohne diese dem Selektionsdruck des evolutionären Algorithmus auszusetzen, könnte hierbei helfen. Da die Platzierung der Neuronen auf dem Substrat ebenso eine grosse Rolle zu spielen scheint, wären zudem Methoden sinnvoll, die eine Optimierung der Platzierung übernimmt[3]. Eine Evaluierung der hier erzielten Ergebnisse anhand der Simulation von gepulsten neuronalen Netzen steht noch aus. Die Ergebnisse, die uns aus anderen Veröffentlichen bekannt sind, lassen allerdings vermuten, dass sich auch mit unserem Verfahren gepulste neuronale Netze erfolgreich evolutionär entwickeln lassen.

[3]Beispielsweise in Form von Evolvable-Substrate HyperNEAT [RS12]

7. Zusammenfassung

Diese Arbeit befasst sich mit der Kodierung und der evolutionären Optimierung von komplexen Netzen, wie sie für den Einsatz von gepulsten neuronalen Netzen bei deren Erzeugung benötigt werden. Basierend auf den Compositional Pattern Producing Networks (CPPN) haben wir ein Verfahren entwickelt, dass sich bei entsprechender Konfiguration für die Erzeugung von Netzen eignet, die sowohl Regelmässigkeiten als auch Variationen in ihrer Topologie und Gewichtung aufweisen können. Die Komplexität der Beschreibung der Netze ist hierbei unabhänig von der Netzgrösse selbst. Zumindest theoretisch lassen sich auf diese Art und Weise beliebig komplexe Netze darstellen. In der Praxis müssen die bestehenden Verfahren zur evolutionären Optimierung dieser Kodierung hingegen noch verbessert werden. Gegenüber der expliziten hierarchischen Beschreibung wie sie in [Sch09] zum Einsatz kommt, sehen wir in unserer Kodierung[1] unter anderem den Vorteil, dass sich damit Abhängigkeiten beliebiger Aspekte von (neuronalen) Netzen effizient ausdrücken lassen. Durch unsere Experimente konnten wir zudem zeigen, dass sich unser Verfahren insbesondere zur Erzeugung bipartiter Netze eignet.

7.1. Geleistete Arbeiten

Zu Beginn haben wir uns in Kapitel 3 ausführlich mit bestehenden Verfahren zur evolutionären Erzeugung von neuronalen Netzen auseinandergesetzt. Hierbei haben wir unter anderem Verfahren analysiert, die auf Lindenmayer-Systemen, Boolschen Netzwerken, CPPNs, zellularer Kodierung oder hierarchischen Ansätzen basieren. Weiterhin haben wir in Kapitel 4 die zwei Verfahren Space Colonization und Diffusionsbegrenztes Wachstum auf die Frage hin untersucht, ob bzw. wie gut sich damit neuronale Netze erzeugen lassen, mit dem Resultat, dass sich diese Verfahren aufgrund der hohen Laufzeit nicht eignen. Durch die genaue Analyse, sowie einige entwickelte Prototypen, sind wir daraufhin zu der Erkenntnis gelangt, dass sich, bezogen auf unsere Anforderungen, die Kodierung in Form von CPPNs am besten eignet.

Basierend auf der Kodierung von CPPNs haben wir in Kapitel 5 unseren eigenen Ansatz entwickelt. Hierfür wurden zuerst einige Verfahren genauer untersucht, die bereits CPPNs verwenden. Unser Ansatz unterscheidet sich von den bereits existierenden in einigen Punkten. Zum einen ermöglichen wir die Erzeugung von gepulsten

[1] die auf azyklischen Netzen basiert, mit mathematischen Funktionen als Knoten

neuronalen Netzen. Weiterhin verwenden wir bei der Kreuzung zweier Genome eine topologische Analyse, um Parameterwerte entsprechend zuordnen zu können. Dies dient der Verbesserung der Konvergenzeigenschaften. Darüber hinaus erweitern wir den multi-modalen evolutionären Algorithmus NGSA-II um eine Regulierung der Redundanz und ermöglichen, abhängig von der jeweiligen Anwendung, die Hinzunahme weiterer Kriterien zur Bewertung.

In Kapitel 6 haben wir unser Erzeugungsverfahren anhand einiger Verbindungsmotive, die häufig in biogischen Systemen auftreten, getestet. Hierbei wurde die Ähnlichkeit zu einem vorgegebenen Zielgraph als Fitnessfunktion verwendet. Erst der Einsatz der Graphenähnlichkeit als Fitnessfunktion hat es uns möglich gemacht, eine genauere Analyse der Erzeugungsqualitäten unseres Verfahrens durchzuführen, und dabei unabhängig von der zugrundeliegenden Konfiguration der gepulsten neuronalen Netze und der konkreten Anwendung zu sein. Hierzu mussten wir ein existierendes Verfahren (Neighbor-Matching) erweitert, um auch die Kantengewichtung in das Ähnlichkeitsmaß mit einfliessen zu lassen. Hilfreich war an dieser Stelle zudem ein von uns entwickeltes Tool zur graphischen Visualisierung der erzeugten Netze sowie zur Steuerung und Einflussnahme auf die Parameterisierung des evolutionären Algorithmus zur Laufzeit. Wir konnten zeigen, dass sich mit unserem Verfahren bestimmte Netze, unabhänig von der Netzgrösse, finden lassen. Bei den Netzen wo dies nicht funktionierte, führen wir die Ursachen auf die genaue Konfiguration des evolutionären Algorithmus zurück, nicht jedoch auf die zugrundeliegende Kodierung.

Die Implementierung unseres Verfahrens, sowie die der Prototypen, erfolgte vollständig in der Programmiersprache Rust. Neben einer hohen Abstraktion, sowie der Speicher- und Threadsicherheit, ermöglichte das eine sehr hohe Performanz.

7.2. Ausblick

Eine Validierung der hier erzielten Ergebnisse durch die Simulation von gepulsten neuronalen Netzen anhand einer konkreten Anwendung steht noch aus. Alle dafür notwendigen Softwaremodule sind bereits vorhanden. Weiterhin halten wir genauere Untersuchungen im Bereich der CPPNs für sinnvoll. Offene Fragen hierbei sind:

- Welche Aktivierungsfunktionen eignen sich besonders gut? Welche sind notwendig für die allgemeingültige Beschreibung von Netzen?
- Welchen Einfluss hat die Konfiguration des Substrates auf die Erzeugungsqualität des Verfahrens? Welche Geometrien eignen sich besonders gut?
- Wie lässt sich das Substrat evolutionär mitentwickeln?
- Kann die Problemgrösse eines existierenden CPPNs im Nachhinein durch Skalierung des Substrates angepasst werden, ohne dabei das CPPN verändern zu müssen?

- Wie kann am besten das Übersteuern der CPPNs vermieden werden?
- Ist eine Modularisierung von CPPNs sinnvoll und wie könnte diese aussehen?

Zudem ergeben sich einige Fragestellungen im Bereich der evolutionären Algorithmen. Diese sind:

- Wie wirkt sich die Qualität der Kreuzungsoperation auf das Ergebnis aus? Welchen Vorteil hat eine topologische Analyse zum Zweck des Alignments?
- Welcher Algorithmus eignet sich für die Bewertung anhand vieler Kriterien? NSGA-III?
- Wie können die Parametereinstellungen automatisiert an die Entwicklung der Population angepasst werden?
- Wie kann einer Verbesserung der Mutationsoperation erzielt werden? Ist eine lokale Suche, die den Selektionsdruck des evolutionären Algorithmus umgeht, sinnvoll?

Weiterhin halten wir eine Verbesserung der Visualisierung, z.B. der Arbeitsweise des evolutionären Algorithmus, für sinnvoll. Im Bereich der neuronalen Netze sehen wir zudem viele weitere interessante Forschungsfelder, z.B. in der adaptiven Entwicklung von gepulsten neuronalen Netzen [Lon11] oder die jüngst veröffentlichten Forschungsergebnissen [Güt16], prädiktive Lernverfahren betreffend.

A. Liste entwickelter Software

Die im Rahmen dieser Arbeit entstandenen Softwaremodule sind in Tabelle A.1 aufgelistet. Diese sind frei unter der folgenden URL verfügbar: `https://github.com/mneumann/Projektname`. Hierbei kam die Programmiersprache *Rust*[1] zum Einsatz.

Projektname	Beschreibung
`hypernsga`	Implementierung unseres eigenen Verfahrens
`lindenmayer-system`	Lindenmayer-Systeme
`graph-neighbor-matching`	Neighbor-Matching Algorithmus
`munkres-rs`	Kuhn-Munkres Algorithmus (Ungarischer Algorithmus)
`triadic-census-rs`	Triadischer Zensus
`izhikevich-neurons`	Neuronenmodell von Izhikevich (Simulator, STDP)
`space-colonization-rs`	Space Colonization
`dla-rs`	Diffusionsbegrenztem Wachstum (DLA)
`neat-rs`	NEAT und HyperNEAT
`nsga2-rs`	NSGA-II Algorithmus
`graph-layout-rs`	Fruchterman-Reingold Graph-Layouting
`graph-edge-evolution`	Graphgrammatik von Horby und Pollack
`graph-annealing`	Evolutionäre Entwicklung nach Hornby und Pollack
`artificial-genome-rs`	Graphgrammatik mit Genregulationsnetzwerk
`acyclic-network-rs`	Azyklischen Netze
`cppn-rs`	CPPNs basierend auf azyklischen Netzen

Tabelle A.1.: Entwickelte Softwaremodule

[1] `https://www.rust-lang.org/`

Abkürzungsverzeichnis

ANN Artifical Neural Network (deutsch *KNN*)

CA Cellular Automaton (deutsch *ZA*)

CPPN Compositional Pattern Producing Network

DLA Diffusionsbegrenztes Wachstum (englisch *Diffusion-limited Aggregation*)

EA Evolutionärer Algorithmus

ES-HyperNEAT Evolvable-Substrate HyperNEAT

GA Genetischer Algorithmus

GANN Genetic Algorithm Neural Network

GNN Gepulstes Neuronales Netz (englisch *SNN*)

GP Genetisches Programmieren

GRN Genregulationsnetzwerk (englisch *Gene Regulatory Network*)

HyperNEAT Hypercube-based NeuroEvolution of Augmented Topologies

kDNA Künstliche DNA

KNN Künstliches Neuronales Netz (englisch *ANN*)

L-System Lindenmayer System

NEAT NeuroEvolution of Augmented Topologies

NSGA Non-Dominated Sorting Genetic Algorithm

NSGP Non-Dominated Sorting Genetic Programming

SNN Spiking Neural Network (deutsch *GNN*)

ZA Zellularer Automat (englisch *CA*)

Literatur

[Aho+97] Isto Aho u. a. *Searching Neural Network Structures with L Systems and Genetic Algorithms*. Techn. Ber. International Journal of Computer Mathematics, 1997.

[BB84] RM Brady und RC Ball. „Fractal growth of copper electrodeposits“. In: *Nature* 309 (1984), S. 225–229.

[BF03] Jesper Blynel und Dario Floreano. „Exploring the T-maze: Evolving learning-like robot behaviors using CTRNNs“. In: *Applications of evolutionary computing*. Springer, 2003, S. 593–604.

[BKL02] Sander M Bohte, Joost N Kok und Han La Poutre. „Error-backpropagation in temporally encoded networks of spiking neurons“. In: *Neurocomputing* 48.1 (2002), S. 17–37.

[BKŠ09] Zdeněk Buk, Jan Koutník und Šnorek, Miroslav. „NEAT in HyperNEAT substituted with genetic programming“. In: *Adaptive and Natural Computing Algorithms*. Springer, 2009, S. 243–252.

[BM01] Vladimir Batagelj und Andrej Mrvar. „A subquadratic triad census algorithm for large sparse networks with small maximum degree“. In: *Social networks* 23.3 (2001), S. 237–243.

[Boe95] Egbert JW Boers. „Using l-systems as graph grammar: G2l-systems“. In: (1995).

[Bri07] Robert Bridson. „Fast Poisson disk sampling in arbitrary dimensions.“ In: *SIGGRAPH Sketches*. 2007, S. 22.

[BS02] Hans-Georg Beyer und Hans-Paul Schwefel. „Evolution strategies–A comprehensive introduction“. In: *Natural computing* 1.1 (2002), S. 3–52.

[Che02] Shu-Heng Chen, Hrsg. *Genetic Algorithms and Genetic Programming in Computational Finance*. Dordrecht: Kluwer Academic Publishers, Juli 2002. ISBN: 0-7923-7601-3. URL: `http://www.springer.com/west/home/business?SGWID=4-40517-22-33195998-detailsPage=ppmmedia%7Ctoc`.

[CML13] J Clune, J Mouret und H Lipson. *Data from: The evolutionary origins of modularity*. 2013. DOI: `doi:10.5061/dryad.9tb07`. URL: `http://dx.doi.org/10.5061/dryad.9tb07`.

[CRT13] H. Cuntz, M.W.H. Remme und B. Torben-Nielsen. *The Computing Dendrite: From Structure to Function*. Springer Series in Computational Neuroscience. Springer New York, 2013. ISBN: 9781461480945. URL: https://books.google.de/books?id=o-q3BAAAQBAJ.

[DA94] Kalyanmoy Deb und Ram B Agrawal. „Simulated binary crossover for continuous search space“. In: *Complex Systems* 9.3 (1994), S. 1–15.

[Deb+02] Kalyanmoy Deb u. a. „A fast and elitist multiobjective genetic algorithm: NSGA-II“. In: *Evolutionary Computation, IEEE Transactions on* 6.2 (2002), S. 182–197.

[DJ11] Kalyanmoy Deb und Himanshu Jain. „Parent to Mean-centric Self-adaptation in Single and Multi-objective Real-parameter Genetic Algorithms with Sbx Operator *“. In: 2011.

[DKS09] J. Drchal, J. Koutnik und M. Snorek. „HyperNEAT controlled robots learn how to drive on roads in simulated environment“. In: *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*. Mai 2009, S. 1087–1092. DOI: 10.1109/CEC.2009.4983067.

[Dol+06] Marisa Dolled-Filhart u. a. „Classification of breast cancer using genetic algorithms and tissue microarrays“. In: *Clinical cancer research* 12.21 (2006), 6459?6468. URL: http://clincancerres.aacrjournals.org/content/12/21/6459.short.

[DS12] Jan Drchal und Miroslav Snorek. „Distance measures for HyperGP with fitness sharing“. In: *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM. 2012, S. 545–552.

[Flo12] Răzvan V Florian. „The chronotron: a neuron that learns to fire temporally precise spike patterns“. In: *PloS one* 7.8 (2012), e40233.

[FP90] F. D. Fracchia und P. Prusinkiewicz. „Visualization of the Development of Multicellular Structures“. In: *Proceedings on Graphics Interface '90*. Halifax, Nova Scotia: Canadian Information Processing Society, 1990, S. 267–276. URL: http://algorithmicbotany.org/papers/multicellular.gi1990.pdf.

[GGK98] Felix Gers, Hugo de GARIS und Michael Korkin. „CoDi-1BitÄ CELLULAR AUTOMATA BASED NEURAL NET MODEL SIMPLE ENOUGH TO BE IMPLEMENTED IN EVOLVABLE HARDWARE“. In: (1998).

[GK02] Wulfram Gerstner und Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[Gru+94] Frédéric Gruau u. a. *Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm*. 1994.

[Gru92] F. Gruau. „Genetic synthesis of Boolean neural networks with a cell rewriting developmental process“. In: *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*. 1992, S. 55–74. DOI: 10.1109/COGANN.1992.273948.

[Gru94] Frédéric Gruau. *Efficient Computer Morphogenesis: A Pictorial Demonstration*. Working Papers. Santa Fe Institute, 1994. URL: http://EconPapers.repec.org/RePEc:wop:safiwp:94-04-027.

[Gru95] Frederic Gruau. *Automatic Definition of Modular Neural Networks*. 1995.

[GS06] Robert Gütig und Haim Sompolinsky. „The tempotron: a neuron that learns spike timing–based decisions“. In: *Nature neuroscience* 9.3 (2006), S. 420–428.

[GS07] Jason Gauci und Kenneth Stanley. „Generating Large-scale Neural Networks Through Discovering Geometric Regularities“. In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. GECCO '07. London, England: ACM, 2007, S. 997–1004. ISBN: 978-1-59593-697-4. DOI: 10.1145/1276958.1277158. URL: http://doi.acm.org/10.1145/1276958.1277158.

[GS08] Jason Gauci und Kenneth O Stanley. „A Case Study on the Critical Role of Geometric Regularity in Machine Learning.“ In: *AAAI*. 2008, S. 628–633.

[Güt16] Robert Gütig. „Spiking neurons can discover predictive features by aggregate-label learning“. In: *Science* 351.6277 (2016). ISSN: 0036-8075. DOI: 10.1126/science.aab4113. URL: http://science.sciencemag.org/content/351/6277/aab4113.

[H+01] Gregory S Hornby, Jordan B Pollack u. a. „Body-brain co-evolution using L-systems as a generative encoding“. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. 2001, 868?875. URL: http://www.cs.uvm.edu/~jbongard/2014_CS206/2001_Hornby.pdf.

[Haf10] Stefan Haflidason. „ON THE SIGNIFICANCE OF THE PERMUTATION PROBLEM IN NEUROEVOLUTION“. Diss. University of Manchester, 2010.

[Han92] James Scott Hanan. „Parametric L-systems and Their Application to the Modelling and Visualization of Plants“. AAINN83871. Diss. 1992. ISBN: 0-315-83871-X. URL: http://algorithmicbotany.org/papers/hanan.dis1992.pdf.

[HCM14] Joost Huizinga, Jeff Clune und Jean-Baptiste Mouret. „Evolving neural networks that are both modular and regular: HyperNeat plus the connection cost technique“. In: *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM. 2014, S. 697–704.

[HH52] Alan L Hodgkin und Andrew F Huxley. „A quantitative description of membrane current and its application to conduction and excitation in nerve“. In: *The Journal of physiology* 117.4 (1952), S. 500.

[HL90] Thomas C Halsey und Michael Leibig. „Electrodeposition and diffusion-limited aggregation“. In: *The Journal of chemical physics* 92.6 (1990), S. 3756–3767.

[Hol62] John H Holland. „Outline for a logical theory of adaptive systems“. In: *Journal of the ACM (JACM)* 9.3 (1962), S. 297–314.

[Hol75] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[Hor+06] Gregory S Hornby u. a. „Automated antenna design with evolutionary algorithms“. In: *AIAA Space*. 2006, 19?21. URL: http://arc.aiaa.org/doi/pdf/10.2514/6.2006-7242;%20http://www.researchgate.net/profile/Al_Globus/publication/228909002_Automated_antenna_design_with_evolutionary_algorithms/links/547375300cf216f8cfaff65a.pdf.

[IGE04] Eugene M Izhikevich, Joseph A Gally und Gerald M Edelman. „Spike-timing dynamics of neuronal groups“. In: *Cerebral cortex* 14.8 (2004), S. 933–944.

[IM08] Eugene M Izhikevich und Jeff Moehlis. „Dynamical Systems in Neuroscience: The geometry of excitability and bursting“. In: *SIAM review* 50.2 (2008), S. 397.

[Izh+03] Eugene M Izhikevich u. a. „Simple model of spiking neurons“. In: *IEEE Transactions on neural networks* 14.6 (2003), S. 1569–1572.

[Izh00] Eugene M Izhikevich. „Neural excitability, spiking and bursting“. In: *International Journal of Bifurcation and Chaos* 10.06 (2000), S. 1171–1266.

[Izh01] Eugene M Izhikevich. „Resonate-and-fire neurons“. In: *Neural networks* 14.6 (2001), S. 883–894.

[Izh04] Eugene M Izhikevich. „Which model to use for cortical spiking neurons?“ In: *IEEE transactions on neural networks* 15.5 (2004), S. 1063–1070.

[Joh04] Maritza Johnson. „Analysis of the C. Elegans Neuronal Network“. In: (2004). URL: http://dreuarchive.cra.org/2004/Johnson/MaritzaJohnson/finalPaper.pdf.

[KA05] Nadav Kashtan und Uri Alon. „Spontaneous evolution of modularity and network motifs“. In: *Proceedings of the National Academy of Sciences of the United States of America* 102.39 (2005), S. 13773–13778.

[Kai04] Florian Kaiser. „Modellierung der Positionsschätzung des Sandskorpions 'Smeringurus mesaensis' mit gepulsten Neuronalen Netzen“. Studienarbeit. Universität Karlsruhe, 2004.

[Kit90] Hiroaki Kitano. „Designing Neural Networks Using Genetic Algorithms with Graph Generation System“. In: *Complex Systems* 4 (1990), 461?476.

[Koe94] Philipp Koehn. *Combining Genetic Algorithms and Neural Networks: The Encoding Problem*. 1994.

[Koz92] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*. Bd. 1. MIT press, 1992.

[KR91] John R. Koza und James P. Rice. „Genetic generation of both the weights and architecture for a neural network“. In: *In International Joint Conference on Neural Networks*. IEEE, 1991, 397?404.

[Kuh55] Harold W Kuhn. „The Hungarian method for the assignment problem“. In: *Naval research logistics quarterly* 2.1-2 (1955), S. 83–97.

[Lee+13] Suchan Lee u. a. „Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings“. In: Hrsg. von Anna I. Esparcia-Alcázar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Kap. Evolving Gaits for Physical Robots with the HyperNEAT Generative Encoding: The Benefits of Simulation, S. 540–549. ISBN: 978-3-642-37192-9. DOI: 10.1007/978-3-642-37192-9_54. URL: http://dx.doi.org/10.1007/978-3-642-37192-9_54.

[LEM92] J David Schaffer Darrell Whitley Larry, J Eshelman und Briarcliff Manor Fort Collins Briarcliff Manor. „Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art“. In: *COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks, June 6, 1992, Baltimore, Maryland*. IEEE Computer Society. 1992,

[Lin68] Aristid Lindenmayer. „Mathematical models for cellular interaction in development: Parts I and II.“ In: *Journal of Theoretical Biology* 18 (1968).

[Lin87] Aristid Lindenmayer. „An introduction to parallel map generating systems“. English. In: *Graph-Grammars and Their Application to Computer Science*. Hrsg. von Hartmut Ehrig u. a. Bd. 291. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1987, S. 27–40. ISBN: 978-3-540-18771-4. DOI: 10.1007/3-540-18771-5_42. URL: http://dx.doi.org/10.1007/3-540-18771-5_42.

[Lon11] Lyle N Long. „An adaptive spiking neural network with Hebbian learning“. In: *Evolving and Adaptive Intelligent Systems (EAIS), 2011 IEEE Workshop on*. IEEE. 2011, S. 17–23.

[LR79] Aristid Lindenmayer und Grzegorz Rozenberg. „Parallel generation of maps: Developmental systems for cell layers“. English. In: *Graph-Grammars and Their Application to Computer Science and Biology*. Hrsg. von Volker Claus, Hartmut Ehrig und Grzegorz Rozenberg. Bd. 73. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1979, S. 301–316. ISBN: 978-3-540-09525-5. DOI: 10.1007/BFb0025728. URL: http://dx.doi.org/10.1007/BFb0025728.

[Luc06] Artur Luczak. „Spatial embedding of neuronal trees modeled by diffusive growth“. In: *Journal of neuroscience methods* 157.1 (2006), S. 132–141.

[Luc09] Claire Luciano. „Network Motifs and E. coli“. In: (2009). URL: http://biogrid.engr.uconn.edu/REU/Reports_09/Final_Reports/Claire_Luciano.pdf.

[LV10] Daniel Lobo und Francisco J. Vico. „Evolution of form and function in a model of differentiated multicellular organisms with gene regulatory networks“. In: *Biosystems* 102.2–3 (2010), S. 112–123. ISSN: 0303-2647. DOI: http://dx.doi.org/10.1016/j.biosystems.2010.08.003. URL: http://www.sciencedirect.com/science/article/pii/S0303264710001425.

[Maa99] Wolfgang Maass. „Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons“. In: *Institute for Theoretical Computer Science. Technische Universitaet Graz. Graz, Austria, Technical Report TR-1999-037.[Online]. Available: http://www. igi. tugraz. at/psfiles/90. pdf* (1999).

[Mil+02] Ron Milo u. a. „Network motifs: simple building blocks of complex networks“. In: *Science* 298.5594 (2002), S. 824–827.

[MMF87] Sasuke Miyazima, Paul Meakin und Fereydoon Family. „Aggregation of oriented anisotropic particles“. In: *Phys. Rev. A* 36 (3 Aug. 1987), S. 1421–1427. DOI: 10.1103/PhysRevA.36.1421. URL: http://link.aps.org/doi/10.1103/PhysRevA.36.1421.

[Moh+12] Ammar Mohemmed u. a. „Span: Spike pattern association neuron for learning spatio-temporal spike patterns“. In: *International Journal of Neural Systems* 22.04 (2012), S. 1250012.

[MTH89] Geoffrey F. Miller, Peter M. Todd und Shailesh U. Hegde. „Designing Neural Networks Using Genetic Algorithms“. In: *Proceedings of the Third International Conference on Genetic Algorithms*. George Mason University, USA: Morgan Kaufmann Publishers Inc., 1989, 379?384. ISBN: 1-55860-006-3. URL: http://dl.acm.org/citation.cfm?id=93126.94034.

[Neb+08] Antonio Jesús Nebro u. a. „A study of convergence speed in multi-objective metaheuristics“. In: *Parallel Problem Solving from Nature–PPSN X*. Springer, 2008, S. 763–772.

[Neu08] Michael Neumann. „Yinspire – A performance efficient simulator for spiking neural nets“. Studienarbeit. Universität Karlsruhe, 2008.

[Nie88] Harald Niederreiter. „Low-discrepancy and low-dispersion sequences“. In: *Journal of number theory* 30.1 (1988), S. 51–70.

[Nik12] Mladen Nikolic. „Measuring similarity of graph nodes by neighbor matching“. In: *Intelligent Data Analysis* 16.6 (2012), 865?878. URL: http://argo.matf.bg.ac.rs/publications/2011/similarity.pdf.

[NPW84] L Niemeyer, L Pietronero und HJ Wiesmann. „Fractal dimension of dielectric breakdown“. In: *Physical Review Letters* 52.12 (1984), S. 1033. URL: http://laplace.ucv.cl/Patterns/Referencias/Pietronero-prl52-1033.pdf.

[PL96] Przemyslaw Prusinkiewicz und Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. New York, NY, USA: Springer-Verlag New York, Inc., 1996. ISBN: 0-387-94676-4. URL: http://algorithmicbotany.org/papers/abop/abop.pdf.

[Pon+06] Sergei L Kosakovsky Pond u. a. „Automated phylogenetic detection of recombination using a genetic algorithm“. In: *Molecular biology and evolution* 23.10 (2006), 1891?1901. URL: http://mbe.oxfordjournals.org/content/23/10/1891.short;%20http://mbe.oxfordjournals.org/content/23/10/1891.full.

[Pon06] Filip Ponulak. „Supervised learning in spiking neural networks with ReSuMe method“. In: *Phd, Poznan University of Technology* 46 (2006), S. 47.

[Rad90] Nicholas J Radcliffe. *Genetic neural networks on MIMD computers*. University of Edinburgh, 1990.

[Rei99] Torsten Reil. „Dynamics of Gene Expression in an Artificial Genome - Implications for Biological and Artificial Ontogeny“. In: *Proceedings of the 5th European Conference on Artificial Life (ECAL'99), number 1674 in Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1999, S. 457–466.

[RLP07] Adam Runions, Brendan Lane und Przemyslaw Prusinkiewicz. „Modeling Trees with a Space Colonization Algorithm.“ In: *NPH* 7 (2007), S. 63–70.

[RS12] Sebastian Risi und Kenneth O Stanley. „A unified approach to evolving plasticity and neural geometry“. In: *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE. 2012, S. 1–8.

[Sch09] Johannes Schmidt. „Erzeugung Gepulster Neuronaler Netze mittels hierarchischer Beschreibungen“. Diplomarbeit. Universität Karlsruhe, 2009.

[Sch65] Hans-Paul Schwefel. „Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik“. In: *Master's thesis, Hermann Föttinger Institute for Hydrodynamics, Technical University of Berlin* (1965).

[SJW91] Wolfram Schiffmann, Merten Joost und Randolf Werner. „Performance evaluation of evolutionarily created neural network topologies“. English. In: *Parallel Problem Solving from Nature*. Hrsg. von Hans-Paul Schwefel und Reinhard Männer. Bd. 496. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1991, 274?283. ISBN: 978-3-540-54148-6. DOI: 10.1007/BFb0029764. URL: http://dx.doi.org/10.1007/BFb0029764.

[SJW93] W. Schiffmann, M. Joost und R. Werner. „Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons“. In: *Proc. of the int. conf. Artificial Neural Nets and Genetic Algorithms*. Springer, 1993, 675?682.

[SM01] Kenneth Stanley und Risto Miikkulainen. *Evolving Neural Networks Through Augmenting Topologies*. Techn. Ber. AI2001-290. Department of Computer Sciences, The University of Texas at Austin, 2001. URL: http://nn.cs.utexas.edu/keyword?stanley:utcstr01;%20http://www.bibsonomy.org/bibtex/23536bbac6a03e607a67de247016a7009/idsia.

[SM02] K. O. Stanley und R. Miikkulainen. „Efficient Evolution of Neural Network Topologies“. In: *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02*. CEC '02. Washington, DC, USA: IEEE Computer Society, 2002, 1757?1762. ISBN: 0-7803-7282-4. URL: http://dl.acm.org/citation.cfm?id=1251972.1252319.

[Sob] IM Sobol. *The distribution of points in a cube and the approximate evaluation of integrals, Zh. Vychisl. Mat. i Mat. Fiz. 7 (1967), 784–802.*

[Sol+08] Andrea Soltoggio u. a. „Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios“. In: (2008).

[Sta07] Kenneth O. Stanley. „Compositional Pattern Producing Networks: A Novel Abstraction of Development“. In: *Genetic Programming and Evolvable Machines* 8.2 (Juni 2007), S. 131–162. ISSN: 1389-2576. DOI: 10.1007/s10710-007-9028-8. URL: http://dx.doi.org/10.1007/s10710-007-9028-8.

[TI89] Simon J Thorpe und Michel Imbert. „Biological constraints on connectionist modelling“. In: *Connectionism in perspective* (1989), S. 63–92.

[Vog79] Helmut Vogel. „A better way to construct the sunflower head“. In: *Mathematical Biosciences* 44.3 (1979), S. 179–189. ISSN: 0025-5564. DOI: http://dx.doi.org/10.1016/0025-5564(79)90080-4. URL: http://www.sciencedirect.com/science/article/pii/0025556479900804.

[VS11] Phillip Verbancsics und Kenneth O Stanley. „Constraining connectivity to encourage modularity in HyperNEAT“. In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM. 2011, S. 1483–1490.

[Wat+10] Ukrit Watchareeruetai u. a. „Multi-objective genetic programming with redundancy-regulations for automatic construction of image feature extractors“. In: *IEICE TRANSACTIONS on Information and Systems* 93.9 (2010), S. 2614–2625.

[WDD91] D. Whitley, S. Dominic und R. Das. „Genetic Reinforcement Learning with Multilayer Neural Networks“. In: *Proc. of the Fourth International Conference on Genetic Algorithms*. San Diego, CA, 1991, S. 562–569.

[Whi89] Darrell Whitley. „The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best“. In: *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989, 116?121.

[Whi93] David W. White. „GANNET: A Genetic Algorithm for Searching Topology and Weight Spaces in Neural Network Design. The First Step in Finding a Neural Network Solution“. UMI Order No. GAX94-25153. Diss. College Park, MD, USA, 1993.

[Wol02] Stephen Wolfram. *A New Kind of Science*. 2002.

[WS81] TA Witten Jr und Leonard M Sander. „Diffusion-limited aggregation, a kinetic critical phenomenon“. In: *Physical review letters* 47.19 (1981), S. 1400.

[WSB90] Darrell Whitley, Timothy Starkweather und Christopher Bogart. „Genetic algorithms and neural networks: Optimizing connections and connectivity“. In: *Parallel computing* 14.3 (1990), S. 347–361.

[XM07] Ling Xu und David Mould. „Modeling dendritic shapes using path planning“. In: (2007).

[Zit+01] Eckart Zitzler u. a. *SPEA2: Improving the strength Pareto evolutionary algorithm*. 2001.