Ein Vergleich händischer und automatisch erzeugter Intelligenz im Bereich künstlichen Lebens

A comparison of manual and automatically generated intelligence in the field of artificial life

Bachelor-Thesis von Patrik Schmittat

November 2009

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering

Ein Vergleich händischer und automatisch erzeugter Intelligenz im Bereich künstlichen Lebens
A comparison of manual and automatically generated intelligence in the field of artificial life

vorgelegte Bachelor-Thesis von Patrik Schmittat

Gutachten: Johannes Fürnkranz

Tag der Einreichung: 18. Oktober 2009

Erklärung zur Bachelor-Thesis

Hiermit versichere ich die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 18. Oktober 2009

(P. Schmittat)

Abstract

Neuro-Evolutions-Algorithmen sind eine Form von maschinellem Lernen, die *evolutionäre Algorithmen* verwenden, um *künstliche neuronale Netze* zu trainieren. Diese Technik soll ein *Multi-Agenten-System* in einer *teilweise* bzw. *vollständig beobachtbaren Umgebung* trainieren. Die Umgebung wird mit einem *Regelwerk*, die dynamische Informationen enthält (Lebenskraftdynamik bei Events, Futtergenerierung und Sterbebedingungen), vom Benutzer vorgegeben. Die Agenten können *rundenbasiert* aus einer gegebenen Menge von *Aktionen* auswählen und diese ausführen. Typische Aktionen sind Bewegen, Stehenbleiben, Futteraufnahme und Reproduktion, somit wird ein primitiver Lebenszyklus ermöglicht.

Dieses System mit seinen Techniken soll in dieser Bachelorarbeit auf *Überlebensfähigkeit* getestet werden, das bedeutet, dass der Neuro-Evolutions-Algorithmus den Agenten die notwendigen Mittel (hier ein 'lernendes' neuronales Netz) zur Verfügung stellt, damit es ermöglicht wird eine steigende Population zu erlangen. Weiterhin soll untersucht werden ob *eigenständig modellierte Netze* den maschinell trainierten überlegen sind. Die Untersuchungen sollen an einer eigens erstellten Simulation durchgeführt werden. Die Implementierung der Simulation ist ebenfalls im Rahmen dieser Bachelorarbeit durchzuführen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungsverzeichnis

1 Einleitung

In diesem Kapitel soll eine Motivation für die Durchführung eines Vergleichs von händischer und maschineller Modellierung künstlich neuronaler Netze gegeben, sowie die erkennbaren Herausforderungen vorgestellt werden.

1.1 Motivation

Autonome Agenten sind Systeme, die innerhalb einer Umgebungen ein bestimmtes Ziel verfolgen und durch ihre Aktionen einen Einfluss auf eine von ihnen wahrgenommene Umgebung haben. Diese Systeme sind bereits vermehrt im Einsatz und eröffnen neue Möglichkeiten. Ein Beispiel für die Nutzung autonomer Agenten ist die Suchmaschine Google. Sie setzt diese Agenten ein, um Informationen aus dem Internet zu beschaffen. Ein nicht autonomer Ansatz wäre hier schlicht unvorstellbar bei der gegebenen Größe des Internets, der Umgebung in der gesucht werden soll.

Die Entwicklung solcher Systeme ist mit großem Aufwand verbunden, auch dann, wenn die Umgebung, in der der Agent agieren soll, vorher bekannt und komplett einsehbar ist. Daher ist es von Interesse die Entwicklung, also den Vorgang des „Lernens“ aus Sicht des Agenten, zu automatisieren. Eine Automatisierung hätte den Vorteil, dass lediglich ein Ziel inklusive einer Bewertungsfunktion spezifiziert werden muss und der schwierige Teil, nämlich die Verhaltensspezifikation, ausgelassen werden kann. Ein möglicher Ansatz um diesen Vorgang zu realisieren ist, künstliche neuronale Netze (KNN) mittels evolutionären Algorithmen anzulernen, das heißt der Agent kann mit diesen angelernten KNNs innerhalb der Umgebung das gewünschte Ziel erreichen.

Jedoch stellt sich an dieser Stelle die Frage, inwiefern diese Techniken gerechtfertigt sind oder ob Aufwand und Rechenzeit lieber in natürlichere Methoden, wie die händische Modellierung dieser Netze, investiert werden sollen. Diese Untersuchung wurde bisher nicht besonders ausgiebig durchgeführt und soll innerhalb dieser Arbeit aufgegriffen werden.

Zur Beantwortung dieser Frage wird zunächst überprüft, ob die Anwendung von evolutionären Algorithmen es ermöglicht, Agenten die notwendigen Mittel zur Verfügung zu stellen, damit sie performant agieren können. Das bedeutet, dass sie ihr gegebenes Ziel nicht nur erfüllen, sondern wenn möglich ihre Aufgabe bestmöglich abschließen. Des Weiteren soll untersucht werden, inwiefern sich händische Modellierung (durch Verhaltensregeln und vollständig eigenständig modellierten KNNs[1]) und automatisiertes Lernen unterscheiden. Mehrere Faktoren bestimmen diese Unterscheidung, hauptsächlich wie performant der Agent mit der gewählten Methode agiert. Weitere Faktoren sind Anpassungsfähigkeit (kleine Änderungen in der Umgebung und die Reaktion darauf), Komplexität (Aufwand der Implementierung und Größe der Lösung) und Fehleranfälligkeit (Ausfall von Sensoren oder motorischer Fähigkeiten).

Die Untersuchungen werden aus zeitlichen Gründen nur an einer Probleminstanz durchgeführt, deshalb muss eine Umgebung ausgesucht und zur Vereinfachung ein bestimmtes Ziel festgelegt werden. Dies ist dadurch lediglich ein Blick in eine bestimmte Richtung, liefert jedoch in diesem Bereich ein Ergebnis, das für weitere Betrachtungen verwendet werden kann. Als Umgebung wird hierfür eine künstliche Welt gewählt, die von Organismen bewohnt wird, welche die autonomen Agenten darstellen. Ihr Ziel ist es ein Überleben innerhalb der Welt zu sichern und dies soll durch die Größe der Population gemessen werden[2]. Weiterhin wird ein spezieller Algorithmus zum Vergleich gewählt. Hierbei handelt es sich um NeuroEvolution of Augmenting Topologies (NEAT)[8], der sich im Bereich von evolutionären Algorithmen bereits bewiesen hat und wie im Kapitel 3 „Stand der Forschung“ dargestellt, Anwendung in Projekten findet.

1.2 Herausforderungen

Künstliche neuronale Netze mit großen Mengen von Neuronen sind selbst nach langer Forschung weiterhin schwer einsehbar. Es ist aufwendig eine Übersicht über die Funktionalität und ihren Ablauf zu gewinnen[3]. Der Grund dafür ist, dass selbst einfache Strukturen wie XOR-Funktionen keine natürlich wirkende Translationen in Netze ergeben. Die erste

[1] Das bedeutet, dass bei der händischen Modellierung das gesamte Netz, daher Struktur sowie Gewichte, selbst erstellt wird.

[2] Eine genauere Beschreibung der Umgebung und deren Population wird im Abschnitt 2.3 „Simulationsumgebung“ gegeben

[3] Als Beispiel für so ein Netz kann im Anhang unter A.1 „Maschinelle Modellierung“ nachgeschaut werden.

Herausforderung wird somit sein: Ein „Gefühl“ zu bekommen, das es ermöglicht effektiv Netze für die Untersuchungen zu entwickeln.

Weitere Probleme können in Richtung Simulation auftreten. Es ist zu untersuchen ob bereits zufriedenstellende Simulationsumgebungen existieren, die die Analyse der Agenten und deren Netze unterstützt. Eine solche Umgebung ist dann zufriedenstellend, falls sie die Modellierung der Problemstellung, sowie einfache und vollständige Einsicht in die Ergebnisse ermöglicht. Sie sollte daher die Untersuchungen unterstützen und diese nicht stören oder gar verhindern.

Die bisher dargestellten Probleme sind von allgemeiner Natur, ein Problem, das durch die Wahl der Probleminstanz entsteht, ist die Modellierung eines Netzes, das Grundlage für eine steigende Population ist. Es gilt an dieser Stelle Richtlinien zu finden, die das Überleben der Agenten beeinflussen und diese in den Entwicklungsprozess der Netze einzubringen. Erst nach diesem Vorgang kann die eigentliche Aufgabenstellung, der Vergleich von automatisch angelernten und händisch erstellten Netzen, bearbeitet werden.

1.3 Ziele

Ziel dieser Arbeit ist es einen Vergleich beider Methoden durchzuführen und die jeweiligen Ansätze, händischer oder maschineller Modellierung, kritisch zu betrachten und zu bestimmen welcher der beiden besser geeignet ist. Es gilt daher folgende Fragen im Kontext des gewählten Beispiels zu beantworten:

- Welches Verfahren liefert Modellierungen, die besser ihre Aufgabe erfüllen?
- Wieviel Aufwand bringt das jeweilige Verfahren mit sich?
- Wie fehleranfällig ist das gewählte Verfahren hinsichtlich Implementierung sowie Anwendung[4]?
- Wie komplex sind die generierten Modelle?
- Wie hoch sind die Kosten für den Betrieb der Modelle?
- Wie hoch ist die Anpassungsfähigkeit der resultierenden Agenten?

Um einen fairen Vergleich durchführen zu können, gilt es die Aufgaben innerhalb dieser Arbeit in folgender Reihenfolge abzuarbeiten. Würde diese Ordnung nicht eingehalten werden, könnten maschinelle Modellierungen den händischen Prozess beeinflussen. Hierbei dienen die ersten Aufgaben vielmehr der Vorbereitung.

1. Simulationsumgebung aussuchen / entwickeln
2. Umgebung modellieren
3. Netzstruktur ermitteln
4. Händische Modellierung
5. Algorithmus aufsetzen
6. Maschinelle Modellierung
7. Vergleich beider Ansätze durchführen

Sollte die in Punkt (1) gewählte Simulationsumgebung eine der bereits öffentlich verfügbaren sein, wie die auf der Hauptseite von NEAT[5], so ist dies für nachfolgende Arbeiten von Vorteil, denn diese können somit auf die gleiche Simulationsumgebung zurückgreifen. Es gilt daher diese Simulationsumgebungen eigenen Implementierungen vorzuziehen.

Punkt (3) ist von besonderem Interesse, denn dieser bestimmt maßgeblich die Überlebensfähigkeit der Agenten. Es muss daher ausgeschlossen werden, dass die Wahl der Struktur die Resultate des Vergleiches beeinflusst. Daher sollte bei dem Entwurf der Netzstruktur darauf geachtet werden, dass die Netze die nötigen Informationen erhalten, um die momentane Situation des Agenten’ nachvollziehen zu können. Die Motoren sind ebenso wichtig, denn ohne eine geeignete Wahl von Ausgängen können die Agenten nicht sinnvoll reagieren.

Punkt (7) dagegen sollte kommende Entscheidungen und Projekte beeinflussen, somit ist es von Nöten die Dokumentation der aufgetretenen Probleme, die daraus resultierenden Entscheidungen und deren Diskussion besonders detailliert festzuhalten.

[4] Wie leicht entstehen Fehler, wie groß sind die Auswirkungen und wieviel Zeit beansprucht das Korrigieren dieser Fehler.

[5] `http://www.cs.ucf.edu/~kstanley/neat.html`

1.4 Übersicht

Im Folgenden wird zunächst eine Einführung in die Problemdomäne gegeben. Anschließend wird der aktuelle Stand der Forschung dargelegt um eine Übersicht zu bekommen. Danach folgen der Ansatz zur Lösung der Probleme sowie die Analyse der experimentellen Ergebnisse. Zum Schluss wird eine kurze Zusammenfassung gegeben, um einen abschließenden Überblick zu bekommen.

2 Einführung in die Problemdomäne

Die in diesem Kapitel vorhandenen Abschnitte sollen eine Grundlage für das Verständnis der Arbeit schaffen oder entsprechenden Verweise vorgeben.

2.1 Künstliche Neuronale Netze

Künstlich Neuronale Netze (KNN) sind mathematische Nachbildungen von Neuronen wie sie aus der Biologie bekannt sind. Der Grund für den Entwurf von KNNs war die Annahme, dass die Informationsverarbeitung unserer Hirne durch solche Netze ermöglicht wird. Ein simples Modell eines künstlichen Neurons ist in Abbildung 2.1 zu sehen.

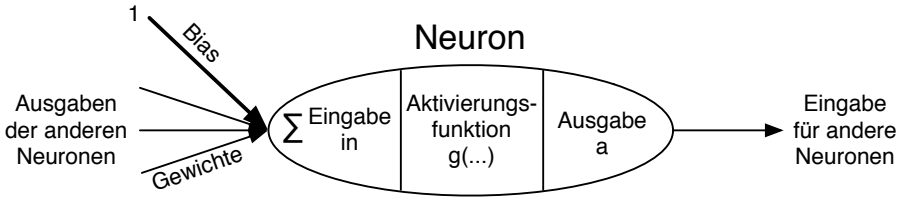


Abbildung 2.1: Einfaches mathematischen Modell eines Neurons. Entnommen aus Russel und Norvig [7].

Die KNNs sind aus mehreren solcher künstlichen Neuronen aufgebaut und können in drei Schichten unterteilt werden, dazu gehört die Eingabe- und Ausgabeschicht sowie die so genannte verdeckte Schicht. Jedes Neuron besitzt, wie in Abbildung 2.1 zu sehen ist, mehrere Eingänge die mit Ausgängen anderer Neuronen verbunden sind. Die Werte an den Eingängen a_j werden mit ihren entsprechenden Gewichten $W_{j,i}$ multipliziert und anschließend aufsummiert. Meistens wird zu dieser Summe ein fester Bias $a_0 = 1$ mit dem zugehörigen Gewicht $W_{0,i}$ addiert, um eine Verschiebung des Schwellwertes zu ermöglichen[1]. Das Ergebnis $in_i = \sum_{j=0}^{n} W_{j,i} a_j$ ist die Neuroneingabe, welche der Aktivierungsfunktion $g(...)$ übergeben wird. Diese Funktion dient dazu den Ausgang des Neurons zu steuern, sie bildet daher den Eingangswertebereich in_i auf einen Ausgangswertebereich $a_i = g(in_i)$ ab. Die Aktivierungsfunktion kann eine beliebige mathematische Funktion sein, wie zum Beispiel die Stufenfunktion oder die differenzierbare Sigmoidfunktion. Daraus resultiert das folgende Verhalten: Wenn der Wert von in_i einen gewissen Schwellwert übersteigt, wird das Neuron einen positiven Wert an den Ausgang a_j übergeben, dies nennt man auch das „Feuern“ des Neurons. In Abbildung 2.2 ist die Stufenfunktion und Sigmoidfunktion abgebildet, um sich einen Eindruck von Aktivierungsfunktionen bilden zu können.

Durch den Einsatz von mindestens einer verdeckten Schicht kann jede stetige Funktion und durch den Einsatz von mindestens zwei verdeckten Schichten jede Funktion ausreichend genau rekonstruiert werden [7, S. 744]. Das Einfügen

[1] Standardmässig liegt dieser Schwellwert bei 0, dieser kann jedoch durch das Verändern des Gewichtes $W_{0,i}$ auf einen Wert ungleich 0 verschoben werden. Damit ist die Modellierung von Vergleichen wie $x < 3$ ermöglicht.

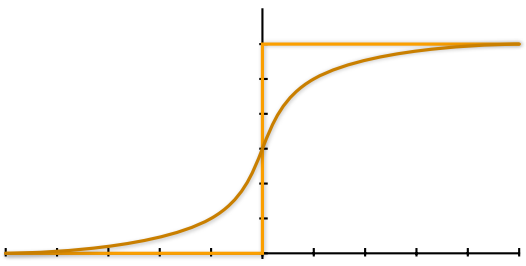


Abbildung 2.2: Verschiedene Aktivierungsfunktionen, hier die Stufenfunktion und Sigmoidfunktion als Beispiel.

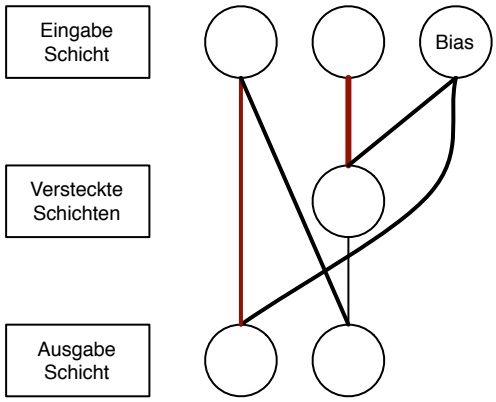
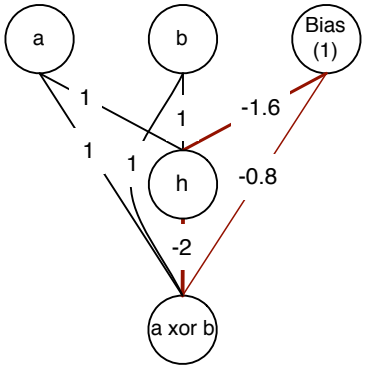


Abbildung 2.3: Ein einfaches KNN mit Eingabe-, Ausgabe- und versteckter Schicht. Rote Linien stellen hierbei negative Gewichte dar, während die dicke der Linien die absolute Größe des Gewichtes repräsentiert.



a	b	in_h	a_h	in_{xor}	a_{xor}
0	0	-1.6	0	-0.8	0
0	1	-0.6	0	0.2	1
1	0	-0.6	0	0.2	1
1	1	0.4	1	-0.8	0

Abbildung 2.4: Boolesche XOR-Funktion mittels Neuronalem Netz konstruiert. Bei diesem Netz wurde die Stufenfunktion als Aktivierungsfunktion verwendet.

von Schichten ermöglicht es, selbst schwierige Probleme wie das „Pole Balancing Problem“ [4] mittels KNN zu lösen[2]. Ein Beispiel einer nicht linearen Funktion ist die XOR-Funktion, diese kann bereits mit mindestens einer verdeckten Schicht nachgebaut werden. Das Netz zu dieser Funktion ist in Abbildung 2.4 zu sehen.

Weiterhin existiert eine besondere Art von KNN, die rekurrenten Netze. Diese Netze besitzen Rückkopplungen von Neuronen aus einer Schicht mit derselben oder einer vorrangehenden Schicht. Dies ermöglicht es Werte über mehrere Auswertungen des Netzes zu verwenden und in künftige Berechnungen einfließen zu lassen. Mittels dieser Technik ist es möglich einem Agenten eine Art von „Gedächtnis“ zu geben, womit er Erfahrungen speichern kann. Ein Beispiel einer rekurrenten Verbindung wäre es wenn das Neuron h aus Abbildung 2.4 mit Neuron b verbunden wäre. Ein größeres Beispiel eines rekurrenten Netzes kann aus Anhang A.1 entnommen werden.

Dies ist lediglich eine kleine Einleitung in das Gebiet der KNNs. Weiterführende Themen wie Perzeptronen oder Backpropagation können in dem Buch von Russel und Norvig [7][S. 736-748] nachgelesen werden, sind jedoch für diese Arbeit nicht von Interesse.

[2] Bei diesem Problem handelt es sich um einen beweglichen Wagen, der zwei unterschiedlich lange Stäbe stabilisieren soll und lediglich seitwärts beschleunigen kann. Die Stäbe besitzen eine gewisse Trägheit und reagieren daher unterschiedlich auf diese Bewegungen. `http://anji.sourceforge.net/polebalance.htm`

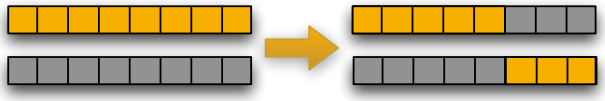


Abbildung 2.5: Rekombination in evolutionären Algorithmen. Hierbei handelt es sich um eine Rekombination mit nur einem (Schnitt)Punkt. Die Farben stellen hierbei die zwei verschiedenen Genome dar, um die Wirkung der Rekombination zu verdeutlichen.

Abbildung 2.6: Mutation in evolutionären Algorithmen. Hierbei handelt es sich um eine Mutation mit gleich mehreren mutierenden Punkten. Die Farben stellen hierbei die Gene dar, die durch die Mutation verändert wurden.

2.2 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) ist ein evolutionärer Algorithmus zur Entwicklung von KNNs. Die Einführung von NEAT setzt jedoch voraus, dass evolutionäre Algorithmen verstanden wurden, deshalb wird zuerst eine Übersicht in diesem Bereich gegeben.

Evolutionäre Algorithmen

Wie alle evolutionäre Algorithmen (EA) ist auch NEAT ein iteratives Optimierungsverfahren. EAs orientieren sich hierbei an der biologischen Evolution und arbeiten mit genetischen Techniken, um das Genom der Individuen weiter zu entwickeln[3]. Die Population nähert sich somit über Generationen hinweg einem Optimum an. Die verwendeten Techniken sind, wie in der natürlichen Evolution auch, Selektion, Rekombination und Mutation.

Selektion Auswahl einer Menge von Individuen anhand einer Selektionsbedingung, wie z.B. dem Wert der Fitnessfunktion. Die Fitnessfunktion wird anfangs jeder Generation auf jedes Individuum aus der Population angewandt. Anhand dieses Wertes kann eine Sortierung der Individuen durchgeführt werden und mit einer Tendenz zu Exemplaren mit höheren Fitnesswert eine Auswahl getroffen werden. Nur die Menge der ausgewählten Individuen wird dann mittels Rekombination und Mutation verändert und anschließend in die neue Generation übernommen.

Rekombination Zwei Individuen werden gegenseitig gekreuzt. Hierbei werden die Gene, die meistens als Zeichenketten dargestellt sind, an bestimmten Punkten „gekreuzt“, dabei übernimmt der entstehende Nachwuchs jeweils ein Teil jedes Individuums. Meistens werden hierbei zwei neue Gene erzeugt die dann dem entstandenen Nachwuchs zugeordnet werden. Eine mögliche Anwendung ist in Abbildung 2.5 zu sehen.

Mutation Direkt nach der Rekombination können die Gene des Nachwuchses einer Mutation unterlaufen, dadurch werden bestimmte Chromosomen (Teilstücke des Genes) in ihrer Wertigkeit verändert. Wenn die Gene als Binärzahl dargestellt werden bedeutet die Mutation das Invertieren eines oder mehreren zufällig ausgewählten Bits. Eine mögliche Anwendung ist in Abbildung 2.6 zu sehen.

Nach einem Durchlauf dieser drei Techniken entsteht dadurch eine neue Population mit dem erzeugten Nachwuchs der Rekombination und der teilweisen Hinzunahme bestimmter Individuen aus der vorherigen Population. Durch die ständige Bewertung der Individuen werden nur die besten Exemplare weiter entwickelt und somit nähert sich die Population dem Optimum. Der Algorithmus kann wie in Auflistung 2.1 zusammengefasst werden.

[3] Genome stellen hierbei den „Bauplan eines Individuums dar. Sie werden auch als Genotyp bezeichnet, während das Individuum Phenotyp heisst.

```
Initialisiere Anfangspopulation
Solange Optimum nicht erreicht:
  Bewertung der Individuen
  Selektiere beste Individuen
    Rekombiniere
    Mutiere
  Ersetze Population durch Auswahl
```

Auflistung 2.1: Der Algorithmus eines einfachen evolutionären Algorithmus'

NEAT

Aufbauend auf dem Prinzip der evolutionären Algorithmen nähert NEAT auf die gleiche Art und Weise KNNs einem Optimum an. NEAT ist jedoch nicht der einzige Algorithmus mit dieser Idee. Im Paper von Stanley und Miikkulainen [8][S. 100-106] können andere Techniken nachgelesen werden, die auf einem ähnlichen Ansatz aufgesetzt haben. Es bildeten sich jedoch bei vorherigen Techniken folgende Probleme heraus:

- Wahl der anfänglichen Population
- Kodierung der Netze (Struktur und deren Gewichte)
- „Competing Conventions"-Problem[4]
- Sichern der Innovationen von Strukturen

Die Besonderheit von NEAT gegenüber anderen evolutionären Algorithmen ist, dass NEAT diese Probleme angeht und lösen kann. Dadurch gewinnt dieser Algorithmus einen Vorsprung gegenüber seinen Vorgängern. Die Lösungswege dieser Probleme werden im Folgenden erklärt und machen NEAT in seiner Funktionsweise aus.

Wahl der anfänglichen Population: Die anfängliche Population bei NEAT sind Netze, die lediglich aus Eingangs- und Ausgangsschicht bestehen. Die Gewichte der Verbindungen sind zufällig und bieten somit einen gewissen Unterschied und dadurch Variation. Aufbauend auf dieser Grundlage werden durch Mutation und Rekombination komplexere Netze gebildet. Dieser Ansatz gewährleistet, dass NEAT immer nur dann Innovationen in der Struktur einführt, wenn sie gefordert sind und nicht mit einer Struktur startet die unnötige Verbindungen oder versteckte Schichten enthält.

Der Vorteil der dadurch entsteht ist, dass die Anzahl der untersuchten Netze möglichst klein gehalten wird und vergleichbar kleine Netze konstruiert werden. Die Anzahl der Netze macht sich in der Geschwindigkeit bemerkbar, die besser als bei anderen Algorithmen ist.

Kodierung der Netze: Die Kodierung eines Netzes wird in zwei Bereiche aufgeteilt, zum einen die Geninformation der Knoten und zum anderen die der Verbindungen zwischen den Knoten. Die Knoten werden jeweils mit Typ (Eingangs-, Ausgangs- oder versteckter Knoten) und ihrer Identifikationsnummer gespeichert. Im Bereich der Verbindungen stehen die beiden adjazenten Knoten, das Gewicht, ob diese Verbindung aktiviert ist und die Innovationsnummer, die widerspiegelt, wann diese Verbindung eingeführt wurde (wird im folgenden Paragraph näher erklärt). Ein Beispiel einer Kodierung kann aus Abbildung 2.7 entnommen werden.

Competing Conventions: Bei den Problem der Competing Conventions handelt es sich um eine nicht eindeutige Kodierung bezüglich der Netze. Somit ist nicht gewährleistet, dass ein Netz nur eine Kodierung besitzt. Das daraus entstehende Problem ist, wenn zwei unterschiedliche Kodierungen, die das gleiche Netz beschreiben, rekombiniert werden es zu einem defekten Nachwuchs kommen kann. Ein defekter Nachwuchs zeichnet sich durch seine große Verminderung des Fitnesswertes gegenüber den fiteren Eltern aus und ist daher nicht erwünscht. Was erreicht werden sollte ist, das die Rekombination identischer Eltern (aus der Sicht der Genome) ein Kind erzeugt, das den Eltern gleicht, denn es werden eigentlich keine unterschiedlichen „Eigenschaften" bei der Rekombination gekreuzt.

Um dieses Problem in den Griff zu bekommen wurden die Innovationsnummern eingeführt. Es handelt sich hierbei um eine globale Identifikationsnummer, die als historischer Marker angesehen werden kann, darunter kann man sich Folgendes vorstellen: Jedes mal wenn eine neue Strukturänderung eingeführt wird[5], erhält diese eine eindeutige Zahl für die spätere Referenzierung. Durch diese Information ist es später möglich komplett unterschiedliche Topologien gegenüber zu stellen und die Struktur zu vergleichen, dies wird bei der Rekombination benötigt.

[4] Zwei unterschiedliche Kodierungen (Genotypen) für das gleiche Netz (Phenotyp).

[5] Darunter zählt nicht das Reaktivieren von Verbindungen, die bereits im Gen vorhanden sind.

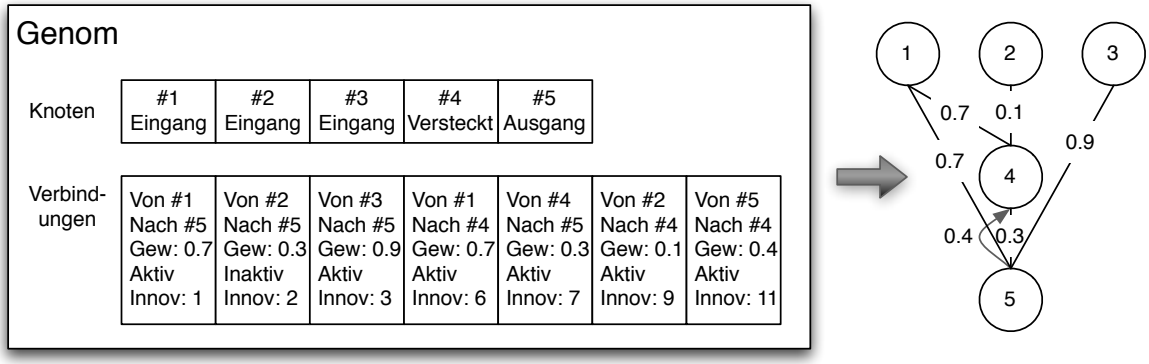


Abbildung 2.7: Darstellung der Kodierung von Netzen bei NEAT. Wie hier zu sehen ist kann NEAT auch mit rekurrenten Verbindungen arbeiten (Verbindungen in Richtung der Eingangsschicht).

Anhand von Innovationsnummern können die Gene eines Netzes sortiert werden und bieten daher eine eindeutige Kodierung pro Netz und umgehen das Problem der Competing Convetions. Ein Beispiel einer Rekombination mit Innovationsnummern ist in Abbildung 2.8 zu sehen.

Sichern der Innovation von Strukturen: Das Aufteilen der Individuen auf bestimmte Spezies ermöglicht es Innovationen zu sichern. Der Vorteil dieser Aufteilung ist, dass die Individuen sich nicht mit der ganzen Population vergleichen müssen sondern lediglich innerhalb ihrer eigenen Spezies. Dadurch sind neuartige Strukturänderungen, die mit hoher Wahrscheinlichkeit anfänglich die Fitness verschlechtern[6], geschützt und werden nicht von bereits optimierteren Strukturen dominiert. Dies wird dann notwendig, wenn der Algorithmus auf ein lokales Optima zusteuert und dadurch die Erzeugung neuer Spezies, mit niedrigeren Fitnesswerten, verhindern würde. Jedoch sind genau diese Neuerungen wichtig, um das lokale Optima zu verlassen.

Die Aufteilung von Individuen auf bestimmte Spezies wird durch die Einführung der Innovationsnummern vereinfacht. Je weniger Innovationsnummern zweier Gene übereinstimmen desto „distanzierter“ sind sie zueinander, wenn dieser Wert eine bestimmte Schwelle überschreitet, dann werden diese zwei unterschiedlichen Spezies zugeordnet.

$$\delta = c_1 \frac{U}{N} + c_2 W$$

Dies ist eine vereinfachte Formel für das oben vorgestellte Entfernungsmaß. Hierbei steht N für die Anzahl der Verbindungen des größeren Genoms, W für den Gesamtunterschied in den Gewichten, U für die Anzahl der Verbindungen, die dem anderen Gen unbekannt sind und c_i sind Stellschrauben für die Gewichtung dieser Faktoren.

Dies war lediglich ein Überblick von NEAT, für ausführliche Erklärungen und experimentelle Ergebnisse sollte die Hauptseite[7] besucht werden oder das Paper von Stanley und Miikkulainen [8] gelesen werden.

2.3 Simulationsumgebung

Durch die Wahl der Probleminstanz muss eine entsprechende Umgebung definiert werden, in der die Agenten sich bewegen können. Die Umgebung ist eindeutig definiert durch die Struktur der Welt, die Sensoren und Motoren der Agenten, sowie das Regelwerk, welches den Ablauf steuert.

Eine kurze Zusammenfassung wird jedoch im Voraus gegeben, damit ein Gesamteindruck entsteht, bevor auf Details eingegangen wird. Bei den Agenten handelt es sich um „Erops“, die auf einer zyklischen „Welt“[8] leben und mit dieser interagieren können. Jedes Erop hat neben seiner aktuellen Position Lebenspunkte, sowie ein gewisses Alter, das die Lebensdegeneration jeder Runde positiv beeinflusst. Jeder Erop kann sich teilen, wodurch der alte Erop getötet wird, jedoch entstehen zwei neue junge Erops mit jeweils halben Lebenspunkten. Pro Erop wird eine Instanz eines KNNs zur Verfügung gestellt, damit jedes Erop sein eigenes Gedächtnis besitzt. Weiterhin existieren Nahrungsquellen (hier

[6] Dies geschieht, da eine zufällige Einführung von Strukturinformationen mit ebenso zufällig gewählten Gewichten meistens nicht direkt optimiert startet und fehlerhaftes Verhalten vorbringt.

[7] `http://www.cs.ucf.edu/~kstanley/neat.html`

[8] Falls sich Entitäten über den Rand hinaus bewegen, werden sie auf der gegenüberliegende Seite wieder eingesetzt.

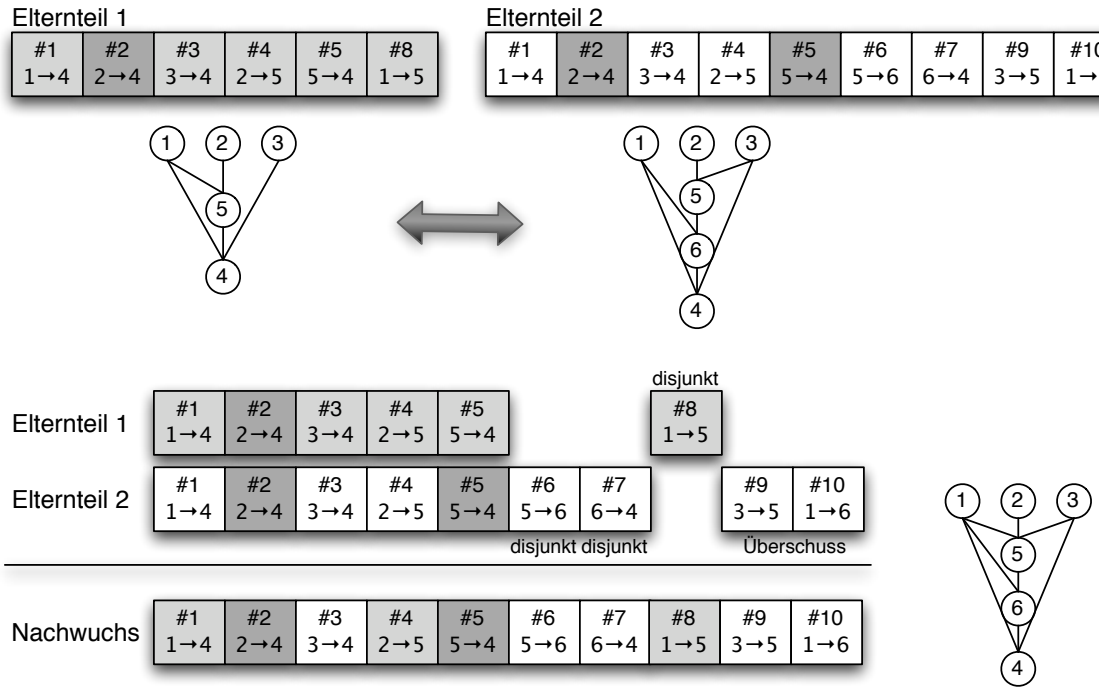


Abbildung 2.8: Darstellung der Rekombination von Netzen bei NEAT. Innovationsnummern zeigen die gleichen Strukturteile auf. Gleiche Gene werden zufällig übernommen, während disjunkte (Gene die in der Mitte nicht übereinstimmen) und überschüssige Gene (Gene die am Ende nicht übereinstimmen) von dem Genom übernommen werden, das den höheren Fitnesswert besitzt. Bei diesem Beispiel wurde angenommen, dass beide Genome die gleiche Fitness besitzen.

„Shrubs“), die es den Erops ermöglichen Lebenspunkte aufzufüllen, indem auf der gleichen Position die Aktion „Essen“ ausgeführt wird. Shrubs füllen ihren Stand von Nahrung periodisch wieder auf und können dadurch mehrmals verzehrt werden. Eine weitere Entität, die „Predators“, können sich in der Welt frei bewegen. Sie jagen die Erops und können diese vernichten, falls sie sich auf dem gleichen Feld befinden. Shrubs können auch vom Predator auf die gleiche Weise zerstört werden, jedoch wird immer eine konstante Anzahl an Shrubs in der Welt gesichert, dies geschieht durch das Einfügen neuer Shrubs an einer zufälligen Position.

Struktur der Welt

Bei der Welt handelt es sich um ein diskretes kugelförmiges Gitter, in dem innerhalb einer Zelle sich mehrere Erops, Predators und Shrubs befinden können. Die Größe der Welt wird während der Simulationen auf eine Begrenzung von 20 Einheiten Breite und Höhe festgelegt. Ebenso werden am Anfang zufällig Shrubs, Predators und Erops platziert um eine lebensfähige Anfangspopulation zu gewinnen, die Anzahl beider Einheiten ist über alle Simulationen gleich, lediglich die Position variiert, um eine Anpassung zu verhindern.

Sensoren und Motoren

Jedes Erop sieht einen Teil der Welt um sich herum, dabei handelt es sich um eine 8-Nachbarschaft. Neben der lokalen Sicht bekommt jeder Erop seine aktuellen Lebenspunkte und das Alter übergeben. Anhand dieser Informationen müssen die Motoren gesteuert werden. Dabei können Erops sich innerhalb der 8-Nachbarschaft frei bewegen, Nahrung von einem Shrub, an der gleichen Position wie der Erop selbst, zu sich nehmen und eine Zellteilung durchführen. Pro Runde eines Erops kann nur eine Aktion eingesetzt werden, dadurch muss abgewägt werden, welche der Aktionen in dieser Situation angebrachter sind.

Predators erhalten die gleiche Sichtweise ihrer Welt und werden mit einer gegebenen Wahrscheinlichkeit sichtbare Erops und Shrubs verfolgen. Die einzige Aktion, die ein Predator ausführen kann, ist das Bewegen innerhalb der 8-

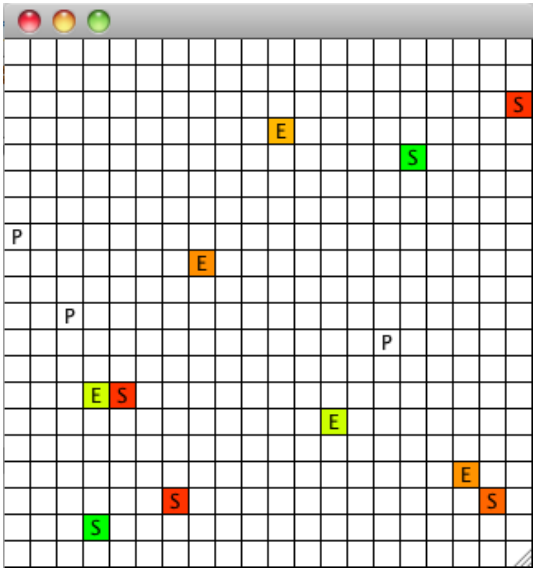


Abbildung 2.9: Visualisierung der Welt einer Simulationsumgebung. Die Welt ist in ein Raster unterteilt und jede Zelle stellt maximal eine Einheit dar. Die Einheiten werden durch *E* für Erop, *S* für Shrub und *P* für Predator dargestellt. Farben zeigen die momentanen Lebenspunkte eines Erops beziehungsweise den Füllstand eines Shrubs an.

Nachbarschaft und das anschließende Zerstören aller Entitäten auf diesem Feld. Wie auch bei den Erops gilt für Predators eine Aktion pro Simulationsschritt, jedoch beschränkt sich die Auswahl auf die Richtung der Bewegung.

Regelwerk

Die Simulation wird zeitlich diskret ablaufen, daher ist zu jedem Zeitpunkt nur eine Aktion eines Erops möglich und alle Erops vollführen ihre jeweiligen Aktionen gleichzeitig. Nach jeder Aktion werden die Lebenspunkte entsprechend des Alters angepasst und die Lebensdegeneration steigt proportional zum Alter des Erops. Gleichzeitig regeneriert jeder Shrub einen Teil seines Nahrungsmittelstandes. Die Aktion der Zellteilung lässt den teilenden alten Erop sterben und erzeugt zwei neue Erops mit jeweils halben Lebenspunkten und einem definierten Anfangsalter; diese Erops befinden sich danach in der Zelle des geteilten Erops.

Ein einfache Visualisierung einer solchen Welt ist in Abbildung 2.9 zu sehen.

3 Stand der Forschung

Seit 1943 wurden im Bereich KNN viele Themen bereits ausgiebig bearbeitet. Angefangen bei den ersten simplen mathematischen Modellen [2], bis hin zu detaillierten und realistischen Modellen, die zu dem Bereich der Computational Neuroscience geführt haben. Darüber hinaus sind bei KNNs auch die abstrakten Eigenschaften enorm wichtig geworden, dabei handelt es sich um: verteiltes Rechnen, Fähigkeit zu Lernen und Toleranz gegenüber verrauschten Eingaben. Diese Faktoren sind der Grund warum KNNs heute noch eine der effizientesten und populärsten Lernsysteme sind [7, S. 737].

Im Bereich des künstlichen Lebens sind KNNs vielseitig eingesetzt [6, 3, 5, 1] und brachten bereits praktische Projekte hervor, darunter zählen zum Beispiel NERO[1] und GAR[2]. Beide sind momentan noch in der Entwicklung, zeigen jedoch große Fortschritte.

Bei NERO handelt es sich um den Versuch NEAT in real-time anzuwenden. Das Ganze ist in die Umgebung eines Spiels eingebettet, bei dem zwei Parteien versuchen die Herrschaft auf einer Karte zu gewinnen. Der Kampf findet indirekt über Einheiten statt und es gilt diese vorher zu trainieren, da direkte Befehle nicht möglich sind. Das Training wird mittels „Reinforcement Learning“ durchgeführt, verändert die KNNs der Einheiten und somit das Verhalten das daraus resultiert.

GAR hingegen verwendet EAs für das Evolvieren von Verhalten eines Partikelsystems, das in die Waffen von Raumschiffen einer Weltraumsimulation integriert ist. Der Einsatz von EAs erlaubt es hier neue Grafiken innerhalb des Spiels zu entwickeln ohne diese mühselig mit Grafiktools entwerfen zu müssen. Dadurch wird während der Laufzeit ein stetiges Kontingent neuer Waffen erzeugt.

[1] http://nerogame.org/

[2] http://gar.eecs.ucf.edu/

4 Ansatz

Im Folgenden sollen die Probleme und Ansätze sie zu lösen dargestellt werden. Diese können in drei Bereiche aufgeteilt werden: Die Struktur der neuronalen Netze, die händische sowie die maschinelle Modellierung und die zeitliche Planung der Modellierungstypen.

4.1 Struktur der Netze

Im Folgenden wird der Grundaufbau der beiden Netztypen dargestellt, mit Netztypen sind hierbei die händische und maschinellen Netze gemeint. Die Ausgangs- und Eingangsneuronen bei beiden Typen werden gleich sein, ohne diese Voraussetzung könnten die Ergebnisse nicht miteinander verglichen werden.

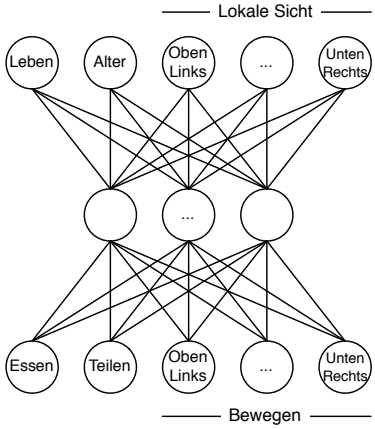


Abbildung 4.1: Allgemeine Struktur der Netze basierend auf der gewählten Simulation und deren Eigenschaften

Es gilt die Netze mit den Informationen, die bereits in Abschnitt 2.3 „Simulationsumgebung“ beschrieben wurden, zu versorgen. Daher muss eine Netzstruktur gewählt werden, die diese Informationen als Eingabe übernimmt und als Ausgabe alle Motoren anspricht. Die hier gewählte Struktur ist in Abbildung 4.1 zu sehen. Die Eingabeschicht erhält die Ausgabe der Sensoren, die vorher auf ein Wertebereich von -1 bis 1 normalisiert werden, dadurch ist gewährleistet, dass alle Sensoren am Anfang mit gleicher Priorität Einfluss nehmen. Die Anpassung findet wie im Folgenden statt:

Leben: Aktuelle Lebenspunkte dividiert durch die maximale Anzahl der Lebenspunkte.

Alter: Das Alter befindet sich bereits in einem normalisierten Zustand.

Lokale Sicht: Aufteilung der lokalen Sicht auf die einzelnen Zellen, daher in neun Neuronen

Zelle: 0 falls die Zelle keine Nahrung enthält, -1 falls sich auf dieser Zelle ein Predator befindet, ansonsten der Füllstand des Shrubs

Die Ausgabeschicht enthält elf Neuronen, davon sind neun Neuronen für die Navigation zuständig, die Aktion Bewegen in eine Richtung und jeweils ein Neuron für Nahrung und Zellteilung. Als Auswahlkriterium wird die Aktion gewählt dessen Neuron den höchsten Ausgabewert hat, somit das Neuron mit der höchsten Präferenz.

Die versteckte Schicht kann beliebig groß werden und ist nicht auf eine Anzahl von Neuronen oder Anzahl von Schichten limitiert. Dadurch ist es möglich beliebige Funktionen abzubilden, jedoch steigt dadurch auch die Anzahl der durchsuchten Netze.

4.2 Händische Modellierung

Für die Modellierung gilt es zunächst Zusammenhänge bei den Eingabeneuronen zu finden. Diese können nur durch eine vorausgehende Analyse der Problemstellung gewonnen werden, genauer gesagt muss zu dem gegebenen Regelwerk der Simulation ein entsprechendes Verhalten, das die Zielfunktion maximiert, modelliert werden. Hierbei ist jedoch darauf zu achten, dass das Verhalten nicht direkt auf dieses Regelwerk angepasst sondern abstrahiert wird, damit trotz Regeländerungen weiterhin zufriedenstellendes Verhalten beobachtbar ist. Danach muss das modellierte Verhalten in ein neuronales Netz transferiert werden.

Zwei Faktoren beeinflussen somit das Ergebniss, dies ist zum einen wie performant das geplante Verhalten der Agenten innerhalb der Simulation (gegeben dem Regelwerk) ist und zum anderen wie genau die Transformation zum KNN gelungen ist. Damit die Transformation begutachtet werden kann, wird zusätzlich eine dritte Art der Modellierung eingeführt. Hierbei handelt es sich um den Einsatz von Verhaltensregeln, die mittels der Programmiersprache JAVA direkt implementiert werden. Dadurch ist der zweite Transformationsschritt geringfügiger, da es sich hierbei um eine Hochsprache handelt, im Gegensatz zu den KNNs, die weniger intuitiv modellierbar sind. Die Verhaltensregeln und deren Ergebnisse können aus dem Anhang A.2 entnommen werden.

4.3 NEAT

Für die automatisierte Modellierung der Netze muss, im Gegensatz zu der händischen Modellierung, nur eine geeignete Fitnessfunktion gefunden werden. Diese muss eine feine Granularität besitzen, damit die verschiedenen Netze in ihrer Performanz eindeutig verglichen werden können. Die in dieser Untersuchung gewählte Fitnessfunktion entstand aus einer Reihe von kleinen Simulationen, die dazu dienten unerwünschtes Verhalten mit Änderungen an der Funktion auszuschließen. Die endgültige Funktion kann folgendermaßen abgebildet werden:

$$Fitness(Chromosom) = \frac{1}{n} \sum_{i=0}^{i<n} sim_i + anz_i \cdot 10 + leb_i$$

Die Anzahl der Auswertungen nach einer Simulation mittels Fitnessfunktion ist hier durch n gegeben und wurde auf 5 gesetzt, um kleine Schwankungen durch Zufall auszugleichen. Die Simulation endet sobald kein Erop mehr lebt oder die maximale Schrittanzahl erreicht wurde, diese beträgt 200. Die letztendlich erreichte Schrittanzahl der Simulation wird in sim_i festgehalten. Der Wert anz_i stellt die Anzahl von lebenden Erops am Ende der Simulation dar, währenddessen leb_i die Summe der Lebenspunkte dieser Erops widergibt. Die Fitnessfunktion bewertet somit Vermehrung und Überleben und erzwingt Erforschung der Umgebung sowie Ausnutzung des gewonnenen Wissens.

Die Netze bestehen Anfangs lediglich aus den Neuronen aus Eingabe- und Ausgabeschicht und besitzen keinerlei Verbindungen. Dies wurde einem vollständigen Netz vorgezogen, damit nur benötigte Verbindungen in das Netz aufgenommen werden und somit der Suchraum möglichst klein bleibt.

Als Implementierung des Algorithmus wurde das ANJI-Framework[1] gewählt. Dieses ist in Java verfasst und erlaubt es durch Konfigurationsdateien einfache Änderungen an Parametern durchzuführen, welche in Abschnitt 5.1 „Simulationsaufbau“ näher erklärt werden. Die Wahl des Frameworks fiel auf ANJI, weil dieses eine entsprechende API besitzt um die Simulation zu integrieren und eine überzeugendere Dokumentation lieferte, während diese bei der Konkurrenz dürftig ausfiel (dazu gehörte unter anderem JNEAT). Zudem liefert ANJI interessante grafische Darstellungen der Egebnisse und ermöglicht, nach der Evolution eines Simulationslaufes, den Zugriff auf jedes Genom einzeln.

4.4 Zeitliche Planung der Modellierungstypen

Wichtig für den Vergleich händischer und maschineller Modellierung ist es, dass die händische Modellierung nicht durch bereits durchgeführte Analysen oder Ergebnisse der maschinellen Modellierung beeinflusst wird. Um dies zu gewährleisten müssen zwei unabhängig Analysen stattfinden, da dies jedoch durch den Rahmen einer Bachelorarbeit nicht möglich ist, wird die händische zeitlich vor der maschinellen Modellierung angesetzt. Damit ist gewährleistet, dass Ergebnisse durch den Einsatz von NEAT nicht die Ergebnisse der händischen Modellierung beeinflussen. Der gröbere Ablauf wurde bereits im Abschnitt 1.2 „Herausforderungen“ vorgestellt, die detailliertere Planung der Modellierungen und Analysen ist somit wie folgt:

[1] http://anji.sourceforge.net/

...

4.1 Analyse der Simulationsumgebung und deren Regelwerk für die händische Modellierung

4.2 Modellierung der ersten Netze und Simulation

4.3 Implementierung der Verhaltensregeln und Simulation

4.4 Netze an veränderte Umgebungen und Regelwerke testen

4.5 Verhaltensregeln an veränderte Umgebung und Regelwerk testen

4.6 Auswertung der Resultate und Analyse der Ergebnisse

...

6.1 Ausführung der maschinellen Modellierung

6.2 Simulation der erzeugten Netze

6.3 Netze an veränderte Umgebungen und Regelwerke testen

6.4 Auswertung der Resultate und Analyse der Ergebnisse

7.1 Vergleich der Auswertungen und Gegenüberstellung der Resultate

5 Analyse

In diesem Kapitel soll auf den Simulationsaufbau sowie die Ergebnisse der Analyse genauer eingegangen werden. Anschließend werden die daraus gewonnen Resultate diskutiert. Die Parameter der Frameworks oder der Simulation werden im Folgenden `monospace` dargestellt, dies soll zur besseren Erkennung dieser Werte dienen.

5.1 Simulationsaufbau

Für eine allgemeine Beschreibung der Simulationen sollte unter dem Abschnitt 2.3 „Simulationsumgebung“ nachgelesen werden. In diesem Abschnitt wird genauer auf den detaillierten Ablauf und die Konfiguration eingegangen.

Pro Simulation, die durchgeführt wird, kann lediglich ein Netz getestet werden. Die Netze werden für beide Modellierungen die gleiche Struktur besitzen[1]. Jedes Erop innerhalb dieser Simulation besitzt eine Instanz des Netzes, dies wurde vorgesehen, weil dadurch jedes Erop ein eigenes Gedächtnis bilden kann. Würde man für alle Erops ein globales Netz vorsehen hätte man ein Schwarmgedächtnis und nicht das eines Individuums.

Jede Simulation erhält eine zufällig generierte Umgebung, die anfänglich mit Shrubs und Erops besiedelt ist. Die Anzahl der Shrubs ist durch `world.shrub.amount` gegeben, die Anzahl der Predators hingegen durch `world.predator.amount.start`, während die Menge an Erops durch `world.erops.amount.start` festgelegt wird. Die Höhe und Breite der Umgebung kann durch `world.width` und `world.height` vor der Simulation gesetzt werden. Die Simulationen werden solange berechnet bis alle Erops ausgestorben sind oder die maximale Schrittanzahl erreicht wird, die Schrittanzahl ist durch `simulation.steps.max` definiert. Nach Abschluss der Simulation wird das Netz mit einem Fitnesswert versehen[2]. Dieser Vorgang wird mehrmals wiederholt um einen Durchschnittswert zu erhalten, die Anzahl der Wiederholungen ist durch `simulation.fitness.trials.max` festgelegt.

Die Shrubs verhalten sich für jeden Schritt in der Simulation gleich, sie regenerieren ihre Nahrung um einen Wert der in `shrub.amount.step.increase` vorgesehen ist und dies bis zu einem Maximum von `shrub.amount.max`. Der anfängliche Wert von Nahrung bei der Generierung der Umgebung liegt bei `shrub.amount.start`. Die Erops besitzen einen Alterungsprozess, der es verhindert ewig zu überleben. Dieser Prozess kann durch folgende beiden Werte `erop.age.life.decrease` und `erop.age.increase` gesteuert werden. Die Formel für die genaue Berechnung pro Schritt:

$$\begin{aligned} aktuellesAlter &\mathrel{+}= erop.age.increase \\ aktuellesLeben &\mathrel{-}= \lceil erop.age.life.decrease \cdot aktuellesAlter \rceil \end{aligned}$$

Hierbei besitzt das Alter keinerlei obere Schranke, wobei die Lebenspunkte durch `lifeform.life.maximum` nach oben hin beschränkt sind. Predators besitzen zwar auch Lebenspunkte, können jedoch nicht von Erops angegriffen werden und sind somit unzerstörbar. Sie verfolgen standardmässig jeden sichtbaren Erop oder Shrub mit einer Wahrscheinlichkeit von `predator.follow.possibility`, andernfalls bewegen sie sich zufällig.

In Auflistung 5.1 wird eine Zusammenfassung aller Parameter für Simulation, Regelwerk und Umgebung dargestellt. Diese Parameter waren lediglich für die vorgestellte Simulationsumgebung gültig, weiterhin existieren noch Einstellungsmöglichkeiten für den Algorithmus von NEAT. Hierfür wurden die in Auflistung 5.2 aufgeführten Werte genommen. Die Auswahl der Werte erfolgte durch Analyse der in [8] vorgestellten Experimente und deren Parameterwerte. Die daraus ermittelten Werte wurden lediglich im Bereich der Populationsgröße und Anzahl von Generationen angepasst, um eine deutliche Konvergenz von Fitnesswerten und gleichzeitig eine annehmbare Rechenzeit zu erhalten. Eine detaillierte Erklärung der Werte findet sich im Paper [8] oder auf der Seite von ANJI[3].

[1] Diese wurde bereits im Abschnitt 4.1 „Struktur der Netze“ vorgestellt

[2] Die dafür zugrundeliegende Fitnessfunktion wurde im Abschnitt 4.3 „NEAT“ vorgestellt

[3] http://anji.sourceforge.net/

```
world.erops.amount.start = 20
world.predator.amount.start = 3
world.shrub.amount = 10
world.width = 20
world.height = 20
simulation.steps.max = 200
simulation.fitness.trials.max = 5
shrub.amount.step.increase = 5
shrub.amount.start = 30
shrub.amount.max = 100
lifeform.life.maximum = 100
erop.age.life.decrease = 10
erop.age.increase = 0.01
predator.follow.possibility = 0.8
```

Auflistung 5.1: Eine Zusammenfassung der gewählten Parameterwerte, die die Simulation beeinflussen

```
num.generations = 100
popul.size = 75
add.connection.mutation.rate = 0.008
remove.connection.mutation.rate = 0.01
remove.connection.max.weight = 100
add.neuron.mutation.rate = 0.005
weight.mutation.rate = 0.75
survival.rate = 0.2
selector.elitism = true
selector.roulette = false
initial.topology.activation = sigmoid
initial.topology.fully.connected = false
initial.topology.num.hidden.neurons = 0
initial.topology.activation.input = linear
recurrent = best_guess
recurrent.cycles = 1
```

Auflistung 5.2: Die festgelegten Parameterwerte von NEAT zusammengefasst

5.2 Auswertung

Jedes entwickelte Netz wird auf zwei Arten bewertet, darunter fällt zum einen der Fitnesswert und zum anderen eine persönliche Begutachtung der Simulation. Die Auswertung aller Netze auf diese Weise ist jedoch aus zeitlichen Gründen nicht möglich, daher wird eine Vorauswahl durch den Fitnesswert getroffen. Dieser Prozess findet automatisch bei der Verwendung von NEAT statt und wird auch teilweise bei der händischen Modellierung eingesetzt. Bei den Verhaltensregeln wird ebenfalls der zugehörige Fitnesswert ermittelt, dies geschieht jedoch lediglich zur Vollständigkeit des Vergleiches und weniger wegen der Vorauswahl. Die Begutachtung wird auf die gleiche Art durchgeführt und abstrahiert daher vom zugrundeliegenden Modell.

Während der Fitnesswert schnell und ohne Beihilfe ermittelt werden kann, muss die persönliche Begutachtung für jedes Netz oder jede Verhaltensregel sorgsam durchgeführt werden. Dies geschieht indem folgende Faktoren jeweils einzeln untersucht werden und der Gewichtung[4] nach summiert werden:

Populationsdynamik 40% Wie schwankt die Population über die Zeit? Gibt es Zeitpunkte an denen ein Aussterben droht?

[4] Die Gewichtungen wurden nach eigenem Ermessen gewählt und spiegeln das gesetzte Ziel dieser Untersuchung wider. Kleine Faktoren sollen hier eine mögliche Unterscheidung bieten, falls die Ergebnisse der großen Faktoren sich zu sehr ähneln.

Lebenspunkte 15% Wieviel Lebenspunkte besitzt die restliche Population? Wie verhielten sich die Lebenspunkte während der Simulation?

Effizienz 15% Werden alle Shrubs gleichmässig ausgenutzt? Bewegen sich die Erops in der 8-Nachbarschaft zielstrebig? Werden Nachkommen zu sinnvollen Zeitpunkten erzeugt (Lebenspunkte / Alterverhältnis)?

Alter 10% Wie hoch ist das Durchschnittsalter der überlebenden Population? Wie verhielt sich das Durchschnittsalter während der Simulation?

Sterbeanzahl 10% Wieviel Erops sind während der Simulation gestorben?

Unbekanntes 10% Wie verhalten sich die Erops bei veränderten Konditionen?

Die besten Netze aus den jeweiligen Netztypen werden gegenüber gestellt und verglichen, dies geschieht einerseits mittels direktem Vergleich der Bewertung, sowie einigen ökonomischen Faktoren wie Zeitkosten, Ressourcenanforderungen und Fehleranfälligkeit. Hierbei spielen jedoch die ökonomischen Faktoren im Vergleich zur Bewertung eine eher kleinere Rolle, jedoch sollte diese nicht unbeachtet bleiben.

Die Verhaltensregeln ermöglichen hierbei noch eine zusätzliche Möglichkeit des Vergleiches. Wie bereits angesprochen könnte die Transformation in die KNNs Probleme verursachen. An der Begutachtung der Verhaltensregeln kann erkannt werden inwiefern eine Änderung des Modells, zu Gunsten des Modellierers, eine Verbesserung der Resultate hervorbringt. Hierbei ist jedoch ein direkter Vergleich der Ergebnisse nicht mehr repräsentativ, daher kann dies nur als Anmerkung dienen.

5.3 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Simulationen vorgestellt. Hierbei werden zunächst die Ergebnisse der händischen Modellierung vorgestellt und anschließend die der maschinellen Modellierung.

Die Zusammenfassung der Ergebnisse der beiden Netztypen und die der Begutachtung sind aus Tabelle 5.2 am Ende dieses Abschnitts zu entnehmen. Für die Ergebnisse der Verhaltensregeln sollte im Anhang unter A.2 „Ergebnisse der Verhaltensregeln“ nachgeschaut werden.

Händische Modellierung

Bei dieser Modellierung stellte sich vor allem eines heraus, der Verdacht gegen Anfang, dass KNNs schwer zu modellieren sind, bestätigte sich. Typisches Verhalten wie zufallsbasiertes Bewegen (wandering), das für die Erforschung von Gebieten genutzt werden kann, ist schwer in einem KNN zu reproduzieren. Ebenso ist die Sigmoid-Funktion als Wahl einer Aktivierungsfunktion ungünstig, da das resultierende Verhalten schwer vorhersehbar ist. Aus diesem Grund wurde für das Netz, das in Abbildung 5.1 zu sehen ist, auch eine lineare Aktivierung gewählt. Ausserdem ist zu erkennen, dass keine rekurrenten Verbindungen vorhanden sind. Der Grund dafür ist, dass es unverhältnismässig schwer war, das dadurch entstehende Verhalten zu kontrollieren.

An dieser Stelle soll eine kurze Erklärung zu dem gewählten Verhalten und dem resultierenden Netz aus Abbildung 5.1 gegeben werden. Der rechte Block aus Neuronen dient zur Navigation des Agenten. Das Verhalten der Motorik ist hier simpel. Falls der Agent durch einen Predator bedroht wird, bewegt dieser sich in entgegengesetzter Richtung. Wenn er unbedroht sein sollte, hält er nach Shrubs in der 8-Nachbarschaft Aussicht und bewegt sich darauf zu. Falls keines dieser Fälle zutrifft, bewegt er sich in den Norden. Der linke Block aus den fünf Neuronen steuert das Verhalten bezüglich Futteraufnahme und dem Teilungsprozess. Die Futteraufnahme ist durch die Lebenspunkte, sowie das Alter gesteuert. Bei zu vielen Lebenspunkten oder zu hohem Alter wird der Agent kein Futter aufnehmen. Die Teilung hingegen wird nur bei einem hohen Alterswert ausgeführt.

Maschinelle Modellierung

Bei dieser Modellierung zeigte sich eine gute Anpassung an die gewählten Parameterwerte, vorallem wenn der Wert `world.predator.amount.start = 0` gewählt wurde. Ebenso war die Aufteilung der Erops über die Shrubs gleichmässig und somit wurden alle Ressourcen optimal genutzt. Scheinbar ist jedoch kein gutes Fluchtverhalten bei anwesenden Predators angelernt worden und einige Erops wurden dadurch zerstört. Die Konvergenz der Netze war erstaunlich schnell und deutete auf eine eher zu hoch gewählte Anzahl auf Generationen hin, verlangsamte sich jedoch bei steigender Gegneranzahl. Die resultierenden Netze sind groß und erlauben daher nur eine eingeschränkte Darstellung dieser Netze[5]. Eine

[5] Das beste Netz der Modellierung ist in Abbildung A.1 im Anhang unter A.1 „Maschinelle Modellierung“ zu sehen.

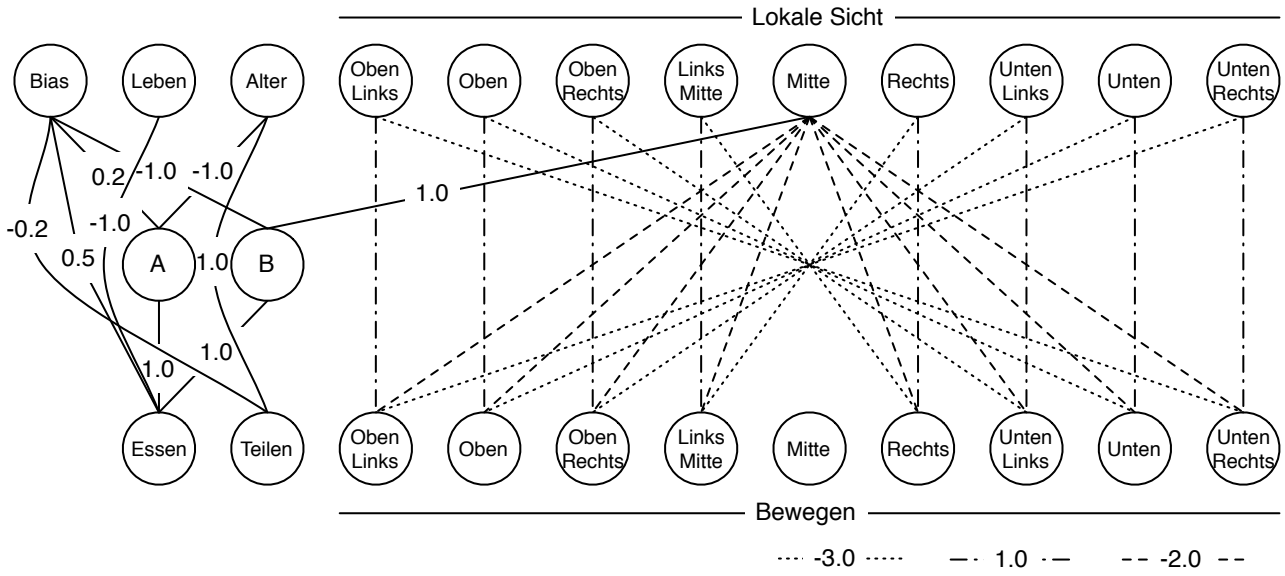


Abbildung 5.1: Neuronales Netz der händischen Modellierung. Hierbei sind die gestrichelten Linien eine Gruppe von Verbindungen, die die gleiche Gewichtung besitzen. *A* und *B* sind hierbei versteckte Neuronen, die eine Art Vergleich durchführen. Alle Aktivierungsfunktionen sind hier linear konfiguriert und der Bias beträgt 1.

Besonderheit dieser Netze ist, dass sie rekurrente Verbindungen besitzen und daher Informationen speichern können, damit ist auch zu erklären wie die zufallsbasierten Bewegungen realisiert wurden, die die Agenten aufweisen.

Übersicht

Aus der Tabelle 5.2 können die Ergebnisse entnommen werden, hierbei werden Werte objektiv gegenüber gestellt und im Abschnitt 5.4 „Diskussion“ näher diskutiert. Des Weiteren stellen die Abbildungen 5.2 und 5.3 den Verlauf des Fitnesswerts und der Komplexität der Netze während einer mechanischen Modellierung dar.

5.4 Diskussion

In diesem Abschnitt werden die Eigenschaften der Modellierungen und die zusammengefassten Ergebnisse verglichen und diskutiert. Es soll somit geklärt werden, welcher der vorgestellten Modellierungstypen besser für das gegebene Problem geeignet ist.

Bevor auf die Ergebnisse eingegangen wird, soll an dieser Stelle die Modellierungsphase betrachtet werden. Es stellte sich heraus, dass beide Methoden ihre Tücken aufweisen. Die händischen Modellierung brachte teilweise komplexe Entwurfsprobleme mit sich, die sich in der Umsetzung von unbeabsichtigtem Verhalten in der Bewegung äusserte. Ein Ansatz für eine Lösung konnte im Nachhinein in den Netzen der maschinellen Modellierung ausgemacht werden, dazu wurden mehrere rekurrente Verbindungen genutzt, um Zustände und Abläufe umzusetzen. Es wird lediglich ein Ansatz genannt, da eine Nachmodellierung dieser Strukturen mit einer erheblichen Schwierigkeitsstufe verbunden ist. Das entstandene rekurrente Netz kann im Anhang unter A.1 „Maschinelle Modellierung“ näher betrachtet werden und zeigt sichtlich dieses Problem auf.

Gleichzeitig zeigte sich die maschinelle Modellierung genauso problematisch bezüglich Vorbereitungen und Verbesserung der Ergebnisse. Frühere Fitnessfunktionen, die allein auf Simulationsstufe und Anzahl der Erops bewerteten, führten zu einer rapiden Vermehrung und einer dauerhaft sehr niedrigen Anzahl der Lebenspunkte. Gleichzeitig erhöhten subtile Parameterwerte wie `initial.topology.fully.connected = true` die Zeit der Konvergenz und lieferte größere Netze. Die endgültig gewählten Parameterwerte forderten kleinere Simulationen, um optimale Belegungen zu ermitteln. Letztendlich ist somit die Zeit der Vorbereitung und Modellierung wenig unterschiedlich und erlaubt es lediglich bei der maschinellen Modellierung eine kurze Pause während den Simulationen einzusetzen. Bei dem Verhalten der beiden Netztypen stellte sich ein größerer Unterschied heraus.

	Händische Modellierung		Maschinelle Modellierung	
Durschn. Fitnesswert	534	1192[6]	562	996[7]
Populationsdynamik	o (12 - 40)	Leicht schwankend, manchmal gefärdet	+ (18 - 45)	Schwankend
Lebenspunkte	+	Ausreichend, stabil	o	Große Unterschiede, manchmal durchgehend niedrig
Effizienz	+	Nicht alle Shrubs dauerhaft genutzt, Bewegung optimal, Teilung selten; nur bei hohem Alter.	o	Shrubs nur beim Vorbeigehen genutzt, dauerhafte Wanderung, Teilung bei sinnvollen Zeitpunkten.
Alter	o (0.1 - 1.9)	Hohe obere Schranke, oszillierendes Verhalten	+ (0.4 - 0.9)	Konstantes Alter, niedriger Durchschnittswert
Sterbeanzahl	+ (120)	Nur altersschwache Erops	- (208)	Viele durch Predator gestorben
Unbekanntes	+	Optimales Verhalten ohne Predators, höhere Nahrungsmenge ausgenutzt, überlebensfähig bei schnellerer Alterung	-	Schlechtes Verhalten ohne Predators, reagiert gut auf höhere Nahrungsmenge, aussterben bei schnellerer Alterung
Zeitaufwand (Modellierung)	-	Mehrere Testläufe notwendig	+	Angemessener Zeitaufwand, starten und auf Ergebnis warten
Vorbereitungsaufwand	o	Feingefühl für KNNs wichtig	-	Parameter für Algorithmus finden, aufsetzen des Frameworks und Einarbeitung
Fehleranfälligkeit	-	Viele Tests notwendig, ungewünschte Nebeneffekte beim Verhalten	+	Bei Nutzung von Frameworks
Größe der Modelle	+	Druckbar und verständlich	o	> 70 Verbindungen, schwer verständlich

[7] Fitnesswert ohne Predators

Tabelle 5.2: Zusammenfassung der numerischen Ergebnisse von den Simulationen. Hierbei bedeuten die Symbole +, o und - ein gutes, neutrales beziehungsweise schlechtes Ergebnis.

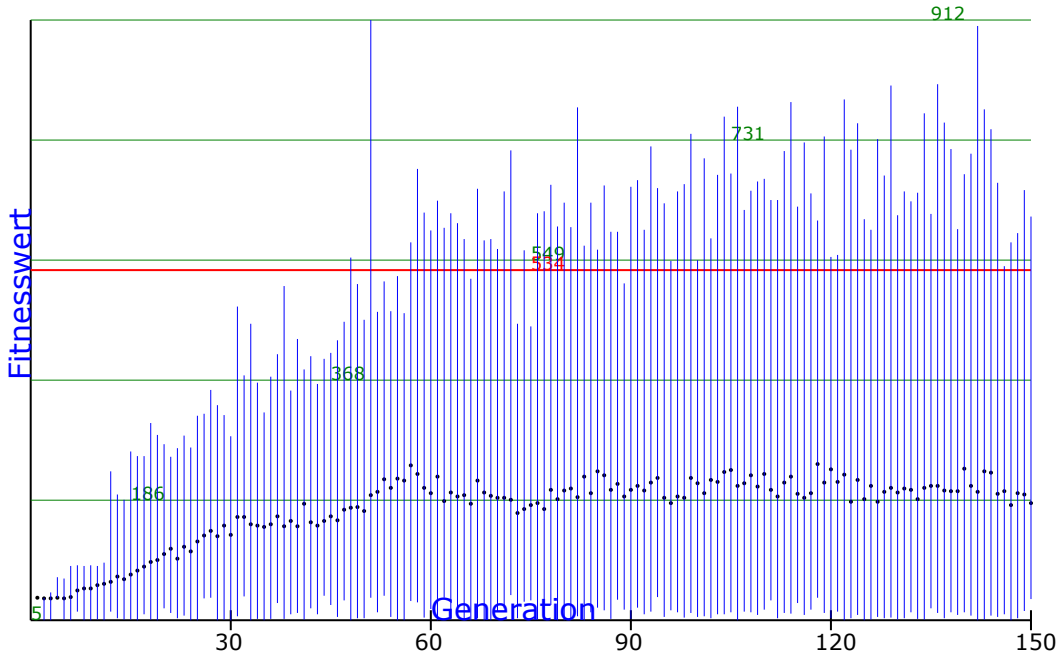


Abbildung 5.2: Verlauf des Fitnesswerts während der mechanischen Modellierung. Hierbei stellen die rote Linie den Fitnesswert des besten händischen Netzes als Vergleich und die schwarzen Punkte den Durchschnittswert der gesamten Population dar.

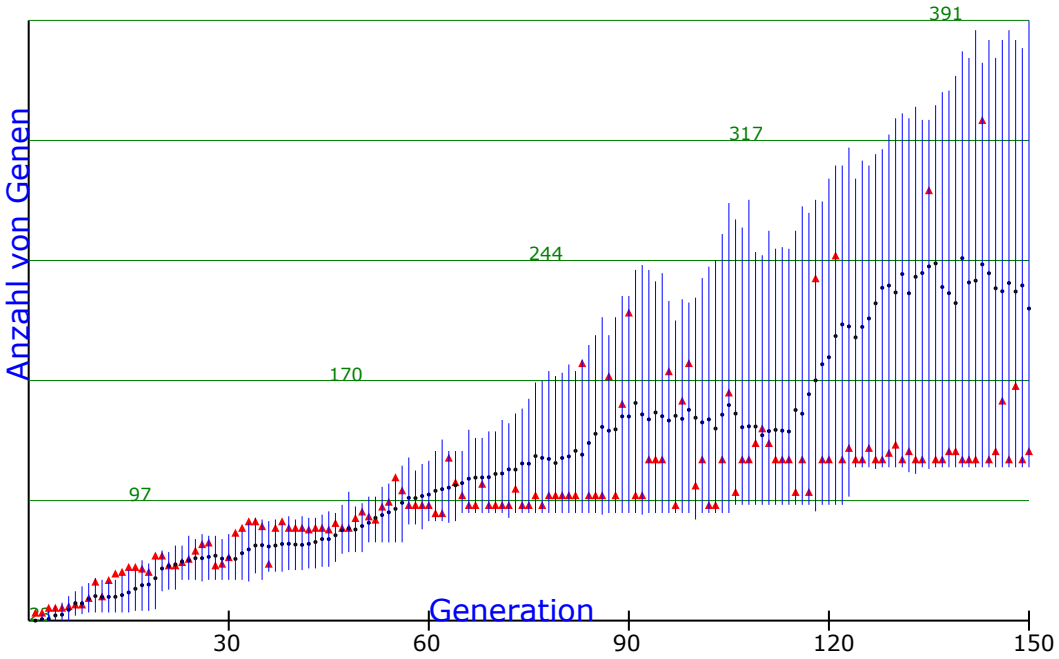


Abbildung 5.3: Verlauf der Komplexität der Netze während der mechanischen Modellierung. Hierbei stellen die schwarzen Punkte den Durchschnittswert der gesamten Population und die roten den Wert des fitesten Individuums dar.

- Die Abstraktion der händisch modellierten Netze war eindeutig höher. Diese reagierten angemessener auf Änderungen in den Parametern der Simulation. Das beste Beispiel hierfür war die Erhöhung der Alterungsrate und die damit fast sofort aussterbende Population der maschinellen Modellierung, während die händische eine, wenn auch schwache, Population aufrecht erhalten konnte.
- Die Größe der Netze spiegelt die Abstraktionsstufe wieder und zeigt, dass auch hier die händisch modellierten Netze überlegen sind. Dies betrifft Komplexität und Anschaulichkeit der Netze.
- In der Performanz hinsichtlich des Fitnesswertes sowie der Populationsgröße zeigte das maschinelle Netz bessere Ergebnisse. Es ermöglichte ein sicheres Überleben der Population und darüber hinaus besaßen die Erops immer ein durchschnittlich niedriges Alter. Bei maschinellen Modellierungen ohne Predators wurde sogar eine herausragende Nutzung aller Shrubs festgestellt.
- Subjektives Beobachten des Verhalten der einzelnen Modellierungen lässt zum Schluss kommen, dass die händisch modellierten Netze besser abschneiden sollten. Die Netze zeigen auf den ersten Blick ein effizienteres Verhalten und optimierte Bewegungsabläufe.

Ein eindeutiges Ergebnis lässt sich somit nicht feststellen. Die Wahl der besten Modellierung ist stark davon abhängig welche Bedingungen an das resultierende Netz gestellt wird. Geht man jedoch nicht über die Grenze des Problems (die Fitnessfunktion sollte dies ausdrücken) hinaus, dann ist die Wahl eine maschinelle Modellierung zu benutzen vorteilhaft, weil damit selbst subtile Faktoren durch die Optimierung berücksichtigt werden, die unter Umständen bei einer händischen Modellierung unbemerkt bleiben und somit das Ergebnis verschlechtern.

Manchmal wird der Einsatz von nicht-funktionalen Anforderungen notwendig und es bleibt nicht bei einer solch einfachen Aufgabenstellung, dazu gehört unter Anderem eine gewünschte Anpassungsfähigkeit, um auch bei sich veränderten Rahmenbedingungen gute Leistungen vollbringen zu können. In solchen Fällen muss die maschinelle Modellierung dies berücksichtigen, kann jedoch dazu führen, dass sich keine Konvergenz einstellt. Händische Modellierungen sind an dieser Stelle von Vorteil, da sie entsprechend durchgeführt werden können und das gewünschte Verhalten direkt umgesetzt werden kann.

Eine eindeutige Entscheidung zum Vorteil von maschinellen Modellierungen kann jedoch dann getroffen werden, wenn Problemklassen derart komplex werden, dass Verhaltensregeln aufwendig zu implementieren sind und ebenso fehleranfällig werden. Ab diesem Punkt bieten sich die automatischen Methoden an, nur stellt sich die Frage wie leistungsstark die vorgestellten Algorithmen bei diesen Problemklassen arbeiten.

Umgebungen, die sich in ständiger Veränderung befinden, erzwingen andere Herangehensweisen, hierfür bietet sich die Untersuchung eines weiteren Algorithmus' an. Der Ansatz verbindet das automatische Lernen von NEAT mit der Anforderung online neue Rahmenbedingungen zu erlernen. Es handelt sich dabei um das, in der Einleitung bereits genannte, Projekt NERO und dem Algorithmus real-time NEAT (rtNEAT)[7]. An dieser Stelle könnte aufgegriffen und ein Vergleich aller drei Ansätze durchgeführt werden, um zu ermitteln ob der erweiterte NEAT-Algorithmus den wechselnden Anforderungen gewachsen ist.

[7] http://nn.cs.utexas.edu/?stanley:ieeetec05

6 Zusammenfassung

In dieser Arbeit wurde die Präsenz zweier Methoden der Modellierung, händisch und maschinell, kritisch hinterfragt. Dies geschah nicht im Allgemeinen, sondern am Beispiel eines Problems. Dabei handelte es sich um die Modellierung künstlichen Lebens, das in einer unbekannten Umgebung überleben sollte. Für den Bereich der maschinellen Modellierung wurde mit NEAT ein evolutionärer Algorithmus vorgestellt und näher erklärt. Die Modelle, die generiert wurden waren Künstlich Neuronale Netze, kurz KNN, die als Eingabe die Sensoren der Agenten verwendeten. Die Agenten der künstlichen Welt wurden Erops genannt und jedes dieser Erops besitzt eine eigene Instanz eines KNNs. Die Ausgabe der Netze diente als Grundlage für das Ansprechen der Motoren und steuerte somit das Verhalten jedes Erops. Die Umgebung selbst war eine zyklische Welt, vorstellbar als Kugel, mit einer diskreten Unterteilung in Zellen und rundenbasierter Aktionen der Erops. Innerhalb dieser Welt werden zufällig Nahrungsquellen platziert, die periodisch ihren Nahrungsstand regenerieren, diese Entitäten wurden Shrubs genannt. Shrubs dienten den Erops als Grundlage für das Auffrischen von Lebenspunkten, die jedem Erop über Zeit proportional zum Alter abgezogen werden. Als letztes wurde eine gewisse Anzahl an Jägern hinzugefügt, die Predators hießen und Erops sowie Shrubs zerstören könnten, um eine Dynamik und Komplexität zu gewährleisten.

Am Beispiel dieser Probleminstanz wurde ein Vergleich durchgeführt, wobei die händische Modellierung zeitlich vor der maschinellen angeordnet war, um Einflüsse der maschinellen Modellierung zu verhindern. Der Vergleich basierte auf den Ergebnissen der Simulationen und war einerseits maschinell, via Bewertungsfunktion einzelner Netze, sowie durch persönliche Begutachtungen erstellt. Anschließend wurde eine Diskussion geführt, die ergab, dass maschinelle und händische Modellierung beidermaßen ihre Vorzüge besitzen. Die Wahl eines Vorgehens wird hauptsächlich von den nicht-funktionalen Faktoren bestimmt, wie zum Beispiel Anpassungsfähigkeit. Einfache Probleminstanzen wie das oben vorgestellte Szenario ermöglichen den Einsatz von maschinellen Modellierungen, um effizient Modelle zu erstellen. Bei komplizierteren Rahmenbedingungen, wie Anpassungsfähigkeit oder ästhetischem Verhalten[1], sind händische Modellierungen vorzuziehen, da sie bessere Ergebnisse erzielen.

Zuletzt wurden weitere offene Punkte vorgestellt, die näher untersucht werden können. Unter anderem handelte es sich dabei um einen dritten Ansatz, der durch die Einführung von rtNEAT entstanden ist und das Problem der Anpassungsfähigkeit zu lösen versucht. Zusätzlich dazu wurde die Frage vorgestellt, inwiefern komplexere Problemklassen, die bereits schwer durch Verhaltensregeln zu lösen sind, durch Algorithmen, wie zum Beispiel NEAT, bewältigt werden können.

[1] Damit ist Empfinden gemeint, dass bei der Beobachtung der Agenten entsteht. Bei maschinell erstellten Netzen kann ein Eindruck entstehen, Agenten würden „chaotisch“ agieren.

Literaturverzeichnis

[1] A. Berlanga, A. Sanchis, P. Isasi, and J. M. Molina. Neural network controller against environment: A coevolutive approach to generalize robot navigation behavior. *Journal of Intelligent and Robotic Systems*, 33(2):139–166, 2002.

[2] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[3] O. Michel. An artificial life approach for the synthesis of autonomous agents. In *Selected Papers from the European conference on Artificial Evolution (AE '95)*, pages 220–231, London, UK, 1996. Springer-Verlag.

[4] D. E. Moriarty and R. Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22(1-3):11–32, 1996.

[5] S. Nolfi and D. Parisi. Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior*, 5:75–98, 1996.

[6] S. Nolfi and D. Parisi. Neural networks in an artificial life perspective. In *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks*, pages 733–737, London, UK, 1997. Springer-Verlag.

[7] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[8] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

A Anhang

A.1 Maschinelle Modellierung

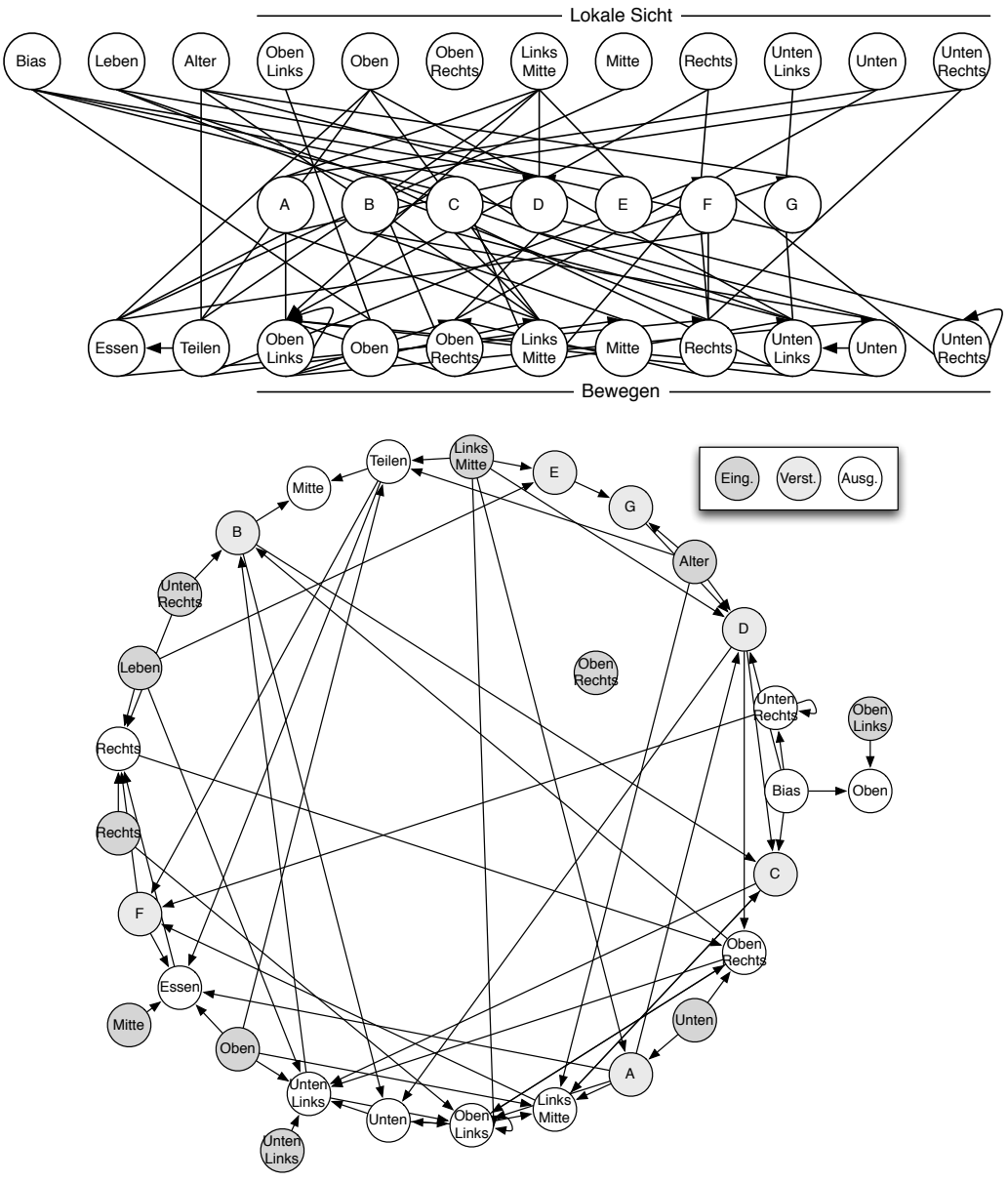


Abbildung A.1: Neuronales Netz der maschinellen Modellierung. Es handelt sich bei beiden Darstellungen um das gleiche Netz, lediglich die Anordnung hat sich verändert. A bis G sind hierbei versteckte Neuronen. Alle Aktivierungsfunktionen sind hier sigmoid (ausser die Eingangsneuronen) konfiguriert und der Bias beträgt 1.

A.2 Ergebnisse der Verhaltensregeln

Die unter 4.2 „Händische Modellierung“ angesprochene Verhaltensregeln wurden wie in Auflistung A.1 implementiert und entsprechend für die Simulationen verwendet.

```
Falls Predator sichtbar
        Laufe in entgegengesetzter Richtung
Falls Alter > 0.15 oder Leben > 80
        Teile dich
Falls Leben < 50 und Shrub an deiner Position
        Esse von Shrub
Falls Leben < 50 und Shrub sichtbar
        Bewege dich zu Shrub
Ansonsten
        Bewege dich in zufälliger Richtung
```

Auflistung A.1: Die Verhaltensregeln aus der Modellierung

Die daraus resultierenden Ergebnisse werden in Tabelle A.2 entsprechend zusammengefasst und mit den anderen Modellierungen verglichen.

	Händische Modellierung		Verhaltensregeln		Maschinelle Modellierung	
durschn. Fitnesswert	534	1192[1]	796	1125[1]	562	996[1]
Populationsdynamik	o (12 - 40)	Leicht schwankend, manchmal gefärdet	+ (25 - 78)	Gesicherte Populations	+ (18 - 45)	Schwankend
Lebenspunkte	+	Ausreichend, stabil	o	Durchwegs mittelmässig	o	Große Unterschiede, manchmal durchgehend niedrig
Effizienz	+	Nicht alle Shrubs dauerhaft genutzt, Bewegung optimal, Teilung selten; nur bei hohem Alter.	+	Alle Shrubs sinnvoll genutzt, Bewegung optimal, Teilung bei sinnvollen Zeitpunkten	o	Shrubs nur beim Vorbeigehen genutzt, Dauerhafte Wanderung, Teilung bei sinnvollen Zeitpunkten.
Alter	o (0.1 - 1.9)	Hohe obere Schranke, oszillierendes Verhalten	o (0.1 - 1.3)	Oszillierendes Verhalten	+ (0.4 - 0.9)	Konstantes Alter, niedriger Durchschnittswert
Sterbeanzahl	+ (120)	Nur altersschwache Erops	o (277)	Angemessene durchschnittliche Sterbeanzahl	- (208)	Viele durch Predator gestorben
Unbekanntes	+	Optimales Verhalten ohne Predators, höhere Nahrungsmenge ausgenutzt, überlebensfähig bei schnellerer Alterung	+	Optimales Verhalten ohne Predators, höhere Nahrungsmenge ausgenutzt, überlebensfähig bei schnellerer Alterung	-	Schlechtes Verhalten ohne Predators, reagiert gut auf höhere Nahrungsmenge, aussterben bei schnellerer Alterung
Zeitaufwand (Modellierung)	-	Mehrere Testläufe notwendig	+	Angemessener Zeitaufwand	+	Angemessener Zeitaufwand, starten und auf Ergebnis warten
Vorbereitungsaufwand	o	Feingefühl für KNNs wichtig	+	Kaum Vorbereitung notwendig	-	Parameter für Algorithmus finden, aufsetzen des Frameworks und Einarbeitung
Fehleranfälligkeit	-	Viele Tests notwendig, ungewünschte Nebeneffekte beim Verhalten	+	Wegen Hochsprache leicht modellierbar	+	Bei Nutzung von Frameworks
Größe der Modelle	+	Druckbar und verständlich	+	Leicht verständlich sequentiell ausgewertete Regeln	o	> 70 Verbindungen, schwer verständlich

[1] Fitnesswert ohne Predators

Tabelle A.2: Zusammenfassung der numerischen Ergebnisse von den Simulationen inklusive der Verhaltensregeln. Hierbei bedeuten die Symbole +, o und - ein gutes, neutrales beziehungsweise schlechtes Ergebnis.