Rheinische Friedrich-Wilhelms-Universität Bonn
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Verteilte, evolutionäre Optimierung von Schwärmen

Diplomarbeit zur Erlangung des akademischen Grades Diplom-Informatiker

Bonn, 26. März 2009

vorgelegt von: David Kriesel

Erstgutachter: Prof. Dr. Sven Behnke
Zweitgutachter: Prof. Dr. Wolfgang Alt

„Allein, der Nutzen hat vielerlei Gestalt. Die Ameisen, die auf ihrer Wanderschaft an einen toten Philosophen geraten, ziehen daraus auch ihren Gewinn.“

— Stanislaw Lem, 1921 – 2006

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine weiteren Hilfsmittel, als die im Rahmen der Arbeit angegebenen, benutzt zu haben. Ausschnitte meiner Arbeit, die dem Wort oder dem Sinn nach anderen Werken entnommen wurden, sind unter Angabe der Quelle kenntlich gemacht.

Ich versichere weiter, dass Illustrationen und Videos – soweit nicht anders gekennzeichnet – Eigenproduktionen sind.

Bonn, 26. März 2009

David Kriesel

Danksagung

Zu Anfang dieser Arbeit möchte ich mich bei allen bedanken, die mein Studium und diese Arbeit begleitet haben und mir in den letzten Jahren zur Seite standen. Ohne die genannten Personen wäre mein Studium und mein Werdegang an entscheidenden Stellen signifikant anders verlaufen. Sie und euch im Studium und dieser abschließenden Arbeitsphase zur Seite zu haben, war meine Inspiration und Bereicherung!

Als erstes möchte ich Prof. Dr. Sven Behnke danken: Er hat mir ermöglicht, diese experimentelle Diplomarbeit in seiner Arbeitsgruppe durchzuführen und sie intensiv begleitet. Ihm sind weiterhin große Teile der Rechner zu verdanken, die ich für diese Diplomarbeit zu einem Grid zusammenschließen durfte. Ohne ihn wäre diese Diplomarbeit in dieser Weise nicht möglich gewesen. Genauso möchte ich mich bei Prof. Dr. Wolfgang Alt bedanken für viele lange Gespräche, für seine herzliche Aufnahme in die theoretische Biologie und nicht zuletzt dafür, dass er sich als Zweitprüfer für diese Arbeit zur Verfügung stellt und diese nach Kräften unterstützt.

Professor Dr. Michael Clausen möchte ich ganz besonders meine Dankbarkeit aussprechen. Sein Interesse, seine Unterstützung, Ermutigung und Weitsicht, die Gespräche und Diskussionen mit ihm waren und sind für mich in jeder Hinsicht von unschätzbarem Wert.

Durch seine Unterstützung in Prüfungsangelegenheiten und speziell die einfache Ermöglichung meines Nebenfachwechsels zur theoretischen Biologie hat auch Prof. Dr. Rainer Manthey seinen Teil dazu beigetragen, dass mein Studium so verlaufen konnte, wie es dann verlaufen ist – dafür möchte ich mich sehr herzlich bedanken!

Besonderer Dank geht an Dr. Nils Goerke, der mich überhaupt erst der Wissenschaft nahegebracht und meine Leidenschaft für sie geweckt hat. Er war es, der mir bereits im Grundstudium den Antrieb gegeben und mich gelehrt hat, mich nicht nur auf Vorlesungen und Veranstaltungen zu konzentrieren, sondern mich vielmehr in Inhalte einzuarbeiten, die mich interessieren und begeistern. Seine wichtigste Lehre an mich war, dass es auch für sehr spezialisierte Inhalte im Studium nie zu früh ist, wenn man Begeisterung für sie empfindet – und durch seine vehemente Art, diese Lehre zu vermitteln, saß ich bereits im frühen Grundstudium in der kleinen Bibliothek der damaligen Abteilung Neuroinformatik bis spät Abends an verschiedensten Büchern. Er war stets zur Stelle, wenn ich fachliche Hilfe oder einfach nur eine klare Ansage brauchte.

Prof. Hod Lipson von der Cornell University danke ich für die außergewöhnliche Einladung zum Forschungsstipendium, welches mein Leben nach meiner Überzeugung nachhaltig verändert hat. Trotz meiner wissenschaftlichen Jungfäulichkeit hat er mir die Freiheit gegeben, zu erforschen, was ich wollte. Er hat mir Unterstützung und Anleitung zuteil werden lassen, wo ich ihn danach fragte, und mir Freiheit gewährt, wo ich es nicht tat. Im Kontext des Forschungsstipendiums möchte ich mich auch bei Prof. Dr. Rolf Eckmiller bedanken, der sich im Vorhinein dazu viel Zeit genommen und mich beraten hat. Michael Arcan sei für die laufende, kommunikative Begleitung dieses Aufenthalts gedankt.

Prof. Dr. Ralf Salomon von der Universität Rostock danke ich sehr für die Zeit, die ich in seiner Arbeitsgruppe verbringen durfte, und in der ich viel über künstliche Evolutionen und ihre Anwendung auf neuronale Netze lernte. Auch wenn die Zeit dort kurz war, habe ich den einen oder anderen Augenöffner mit auf den Weg bekommen („Du denkst ja wie ein GA-ler, lass das!“). In diesem Kontext möchte ich mich auch bei Stefan Goldmann und dessen Gastfreundschaft in Rostock bedanken.

Für Verbesserungsvorschläge, gewissenhafte Korrekturen und kritische Rückfragen möchte ich mich bedanken bei Christiane Schultze, Sarah Meyer, Bernd Schönbach und Andreas Huber. Dafür, dass sie mir die Rechenkapazität ihrer Computer zur Verfügung gestellt haben, möchte ich bei Joachim Nock, Ralf Waldukat und Diana von Gallera bedanken.

Für die viele Unterstützung, die bereits vor Studienbeginn eingesetzt hat, möchte ich mich bei der gesamten Mann- und Frauschaft des Notariates Dr. Kemp Dr. Kolb in Bonn bedanken, bei der ich mich immer gut aufgehoben fühle. Insbesondere möchte ich hier Herrn Dr. Kemp und Christiane Flamme nennen.

Den letzten, sozusagen den Ehrenplatz, müssen sich zwei Parteien bzw. drei Personen teilen. Meinen Eltern Friedrich-Wilhelm Kriesel und Christiane Kriesel möchte ich für die Unterstützung danken, die sie mir über die Entstehung dieser Arbeit und mein ganzes Studium hinweg haben zuteil werden lassen. Ich möchte mich auch für ihr Vertrauen und ihre Unterstützung bedanken, mit der sie meine Wünsche und Entscheidungen begleitet haben.

Meiner Freundin Verena Thomas danke ich für die Beistand und Ermutigung, sowohl im Studium, besonders der Diplomarbeit, als auch im Leben. Ich danke ihr auch für das kritische Gegenlesen dieser Arbeit. Sie hat mich dazu verleitet, die ganze oben beschriebene Unterstützung auch tatsächlich zur Erreichung konkreter Ziele einzusetzen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ziel dieser Diplomarbeit ist die Entwicklung von grundlegenden Techniken und Werkzeugen für die freie evolutionäre Synthese von Schwarmverhalten mithilfe der Verteilung von Rechenlast auf mehrere Computer. Daran anschließend soll eine Familie von Experimenten im Bereich der Schwarmevolution durchgeführt werden.

Seit jeher war der Mensch fasziniert und zutiefst beeindruckt von abertausenden synchron blinkenden Glühwürmchen zur Sommerzeit oder der eleganten Bewegungsweise eines Schwarms von Fischen. Auf andere Weise beeindrucken komplizierte und als technisch ausgereift anmutende Bauten von verschiedenen staatenbildenden Insektenarten, deren Perfektion und Durchdachtheit man erst langsam beginnt zu begreifen. Schwarmhafte Verhaltensweisen wie die genannten sind in der Biologie seit langem bekannt. Zunächst beispielsweise von Maeterlinck 1927 [43] auf eher poetische Weise beschrieben, legte man in späteren Arbeiten viel Augenmerk auf die Mechanismen, welche sich Schwärme zunutze machen, um alltägliche, jedoch nichttriviale Probleme zu lösen, wie beispielsweise die effiziente Lokalisation und Ausbeutung von Nahrungsquellen. Hierbei stellte man überrascht fest, dass die Komplexität des beobachteten, *emergenten* Verhaltens des Schwarms oft in krassem Gegensatz zur relativen Einfachheit der Individuen steht, aus denen der Schwarm besteht. Das Gesamtsystem scheint mehr zu sein als die Summe seiner Teile. Während die Individuen nur zu relativ einfachen Verhaltensweisen in der Lage sind, zeigt der Schwarm höchst komplexes, flexibles und robustes Verhalten – man spricht von *Schwarmintelligenz* oder schlicht *Schwarmverhalten*. Es sei weiter erwähnt, dass emergentes komplexes Verhalten durch lokale Interaktion einfacher Bausteine nicht nur auf das Reich der Tiere beschränkt ist. Arbeiten wie z.B. [24, 38] identifizieren vergleichbare Phänomene in verschiedenen Bereichen der Physik und Chemie.

In neuerer Zeit – insbesondere ermöglicht durch die kontinuierlich ansteigende Rechenkraft neu entwickelter Computertechnik – ist Schwarmverhalten nicht nur in der Natur zu beobachten, sondern auch zu *synthetisieren*. Exemplarisch seien hier die von Reynolds im Jahre 1987 publizierten, viel zitierten *Boids* genannt [54]: Boids sind rein synthetische Agenten, deren individuelles Verhalten auf wenigen, sehr einfachen Regeln basiert. Durch das Zusammenspiel dieser Agenten in einem *Schwarm* entsteht dann ein Verhalten, welches beispielsweise Vogelschwärmen in der Natur verblüffend ähnlich sieht (Abb. 1.1 auf der folgenden Seite) – und gleichzeitig darstellt, wie einfach derartige Verhaltensweisen dezentral organisiert sein können, obwohl sie für den außenstehenden Betrachter den Anschein erwecken, als seien sie von einer zentralen Instanz gesteuert. Methoden aus dem Bereich der Schwarmintelligenz brachten seither Arbeiten in vielen Disziplinen hervor. Über dezentral gesteuerte Roboterschwärme, Routing von Datenverkehr in großen Kommunikationsnetzwerken bis hin zu idealisiertem synthetisiertem Schwarmverhalten, mit dessen Hilfe sinnvolle Lösungen für komplexe Optimierungsprobleme gefunden werden können, scheinen die Möglichkeiten unbegrenzt.

Populäre Verdienste wissenschaftlicher Arbeiten im Bereich des Schwarmverhaltens inspirierten dann Romanautoren zu Bestsellern: Auf der Seite des synthetisierten Schwarmverhaltens sei der Roman *Prey* (Beute) von Michael Crichton [13] genannt, in welchem das Verhalten von Schwärmen

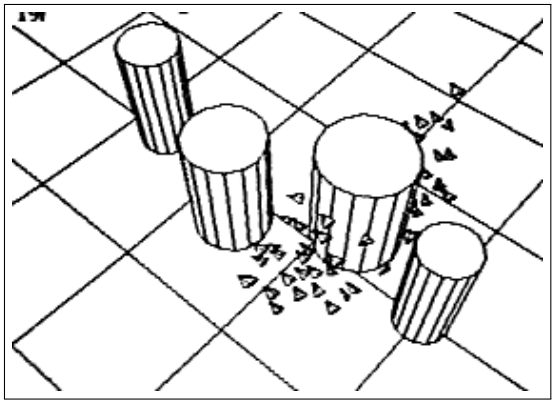


Abbildung 1.1: Boids bei Umgehung von Hindernissen. Entnommen von der Boids-Internetseite von C.W. Reynolds, `http://www.red3d.com/cwr/boids/`.

von Mikrorobotern außer Kontrolle gerät. Mehr durch die Biologie motiviert ist Frank Schätzings Roman *Der Schwarm* [57], der eine Bedrohung der Menschheit durch eine bisher unentdeckte, maritime Schwarmintelligenz beschreibt.

Weiter wachsende Rechenkraft ermöglicht schließlich die Synthese von Schwarmverhalten auf *evolutionäre* Weise: Man bildet Mechanismen der natürlichen Evolution nach, und lässt auf diese Weise künstliches Schwarmverhalten entstehen. Mittels moderner Computer- und Simulationstechnik ganze Evolutionen von Schwärmen durchführen zu können, eröffnet ganz neue Möglichkeiten. Allerdings sind künstliche Evolutionen von Schwärmen bis jetzt meist in signifikanter Art und Weise in ihrer Verhaltensentstehung beschränkt gewesen. Im Rahmen der Diplomarbeit sollen einige dieser Beschränkungen zunächst identifiziert und ihnen anschließend begegnet werden, um umfangreichere, freiere Evolutionen zu erlauben. Da die Vermeidung von Beschränkungen in der Regel den Parameterraum eines Problems signifikant vergrößert, soll der bei der Suche im Parameterraum entstehende Rechenaufwand auf mehrere Rechner verteilt werden. Dies führt uns zur Aufgabenstellung, welche im Rahmen der Diplomarbeit bearbeitet werden soll.

1.1 Aufgabenstellung

Die Aufgabenstellung gliedert sich in zwei primäre Bestandteile: Die Implementierung einer Softwareinfrastruktur, um Experimente zu ermöglichen – und die Durchführung von Experimenten auf Basis dieser Infrastruktur.

1.1.1 Implementierung einer Softwareinfrastruktur

Im Rahmen der Diplomarbeit soll zunächst eine universelle Softwareinfrastruktur für die Durchführung von Experimenten zur Schwarmevolution implementiert werden. Diese Softwareinfrastruktur ist zudem so zu konzipieren, dass Experimente mit geringem Programmieraufwand realisiert und durch Parallelisierung auf Rechnercluster zeiteffizient und robust durchgeführt werden können. Dies soll insbesondere große Experimente mit freien Evolutionen ermöglichen. Die zu implementierenden Softwarebestandteile beinhalten u.a.

- die Erweiterung einer vorhandenen Evolutionssoftware zur Client-Server-Applikation,
- ein durch evolutionäre Algorithmen einstellbares neuronales Netz als Kontrollstruktur der Individuen in einem Schwarm,
- eine effiziente, abstrakt gehaltene Software zur Schwarmsimulation,

- konkrete Schwarmexperimente auf Basis dieser Simulationssoftware sowie
- eine Testsuite zur teilweise automatisierten Analyse entstehenden Schwarmverhaltens.

Die skizzierte Softwareinfrastruktur soll mit Blick auf die Weiterverwendung und Erweiterung in späteren Arbeiten möglichst generisch gehalten werden.

1.1.2 Durchführung von Experimenten

Auf der Basis dieser Softwareinfrastruktur soll dann eine Familie exemplarischer Schwarmevolutionsexperimente durchgeführt werden. Zu diesem Zweck soll ein Modellindividuum mit naturinspirierten Eigenschaften entwickelt werden, aus welchem sich die Schwärme zusammensetzen sollen. Im Rahmen dieser Evolutionen sollen Simulationen zur Evaluation der Schwärme dienen. Um die Möglichkeit der Entwicklung von Spezialfertigkeiten miteinzubeziehen, soll sich der gesamte Schwarm aus einer festen, kleinen Anzahl von jeweils homogenen Subschwärmen zusammensetzen. Zur effizienten Durchführung dieser Experimente muss eine Anzahl von Computern akquiriert werden, auf die der Rechenaufwand verteilt werden kann.

1.2 Überblick

Der Rest dieser Diplomarbeit ist wie folgt gegliedert: In Kapitel 2 wird ein Überblick über den Stand der Forschung in den Bereichen der Beobachtung und Synthese von Schwarmverhalten vermittelt. Weiterhin werden Einblicke in die Anwendungsbereiche synthetisierten Schwarmverhaltens und in verschiedene verbreitete Varianten der Synthese gegeben. Es erfolgt eine Diskussion der vorgestellten Forschungsbemühungen auf abstrakter Ebene, Beschränkungen werden herausgestellt und Maßnahmen zur Vermeidung dieser vorgeschlagen. In Kapitel 3 werden Grundlagen von evolutionären Algorithmen und neuronalen Netzen vermittelt, welche im Rahmen der Diplomarbeit eingesetzt werden. Im Anschluss daran beschreibt Kapitel 4 die Modellierung des Versuchsaufbaus samt Modellindividuum unter Einbeziehung der vorgestellten Konzepte. Diese Beschreibung erfolgt völlig losgelöst von der technischen Realisierung, auf welche dann Kapitel 5 eingeht. Im Rahmen dieses Kapitels wird die Implementierung der oben genannten Softwarebestandteile auf semantischer Ebene beschrieben. Kapitel 6 widmet sich dann Experimenten auf Basis der genannten Softwareinfrastruktur und deren Resultaten. In Kapitel 7 wird eine Zusammenfassung über die geleistete Arbeit gegeben und weitere, auf diese Arbeit aufbauende Forschungswege aufgezeigt.

1.3 Technische Hinweise zur beigelegten DVD

Der Diplomarbeit ist eine DVD beigelegt, auf welcher kommentiertes, geschnittenes Videomaterial präsentiert wird und die implementierte Software archiviert ist.

1.3.1 Videomaterial

Das Videomaterial veranschaulicht insbesondere die in Kapitel 6 dokumentierten, evolutionär entstandenen Schwarmverhaltensweisen. Zum Abspielen dieses Videomaterial ist ein PC mit aktuellem Browser (z.B. Firefox ab Version 2, Internet Explorer ab Version 6 oder Opera ab Version 9) und aktuellem Flash-Player (ab Version 9) erforderlich. Einzelne Videos können über ein intuitiv bedienbares Menü angewählt werden. Dieses wird gestartet, indem man die Datei `index.html` im Unterverzeichnis `videos` der CD im Browser betrachtet.

1.3.2 Software und Experimentergebnisse

Auf der DVD befindet sich ebenso die implementierte Software in einem Format, wie sie von dem IDE *Eclipse* gelesen werden kann, sowie die Ergebnisse durchgeführter Experimente. Nähere Informationen hierzu sind der Datei `info.txt` im Hauptverzeichnis der DVD zu entnehmen.

Kapitel 2

Einordnung und verwandte Arbeiten

Es gibt eine Fülle von Arbeiten in verschiedenen Teilbereichen des Schwarmverhaltens, über die dieses Kapitel einen kurzen, repräsentativen Überblick geben soll. Viele Arbeiten ergründen natürliches Schwarmverhalten (Abschnitt 2.1), andere synthetisieren es (Abschnitt 2.2). Idealisierte Formen von Schwärmen finden beispielsweise in Optimierungsverfahren Anwendung (Abschnitt 2.2.1). Weitere automatisieren die Synthese mittels evolutionärer Algorithmen (Abschnitt 2.2.3). Meist werden im Rahmen der Synthese homogene Schwärme betrachtet.

Es sollen nun einige für das Umfeld der Diplomarbeit repräsentative Arbeiten vorgestellt werden. Schließlich sollen diese Arbeiten in Abschnitt 2.3 auf abstrakter Ebene diskutiert werden.

2.1 Schwarmverhalten in der Natur

Das Reich der Lebewesen ist auf jeder seiner Ebenen – vom hochentwickelten Organismus wie dem des Menschen bis auf die Ebene der Einzeller – geprägt von schwarmhaften Verhaltensweisen. Vielen dieser Verhaltensweisen ist gemein, dass die Komplexität des Verhaltens nicht etwa auf der Ebene des *Individuums*, sondern erst durch lokale Interaktionen zwischen den Individuen und ihrer Umwelt entsteht. Ein *Schwarm* muss für die Erzeugung solch komplexen Verhaltens also – im Vergleich zur Verhaltenskomplexität – nur aus simplen Individuen zusammengesetzt sein. Man spricht hier von der *Emergenz* komplexen Verhaltens – davon, dass das Ganze, wie bereits in der Einleitung angedeutet, mehr als die Summe seiner Teile ist.

Auf der Ebene der Mikroorganismen zeigen beispielsweise Kugelalgen (*Volvox*) [32] Schwarmverhalten: Sie erzeugen kugelartige Kolonien (Abb. 2.1 auf der folgenden Seite), welche aus tausenden individueller Zellen bestehen können, die die Oberfläche der Kugel bilden. Im Inneren der Kugel befindet sich eine von den Zellen erzeugte Gallerte. Anhand der Kolonien kann man einen mehrstufigen Vermehrungsvorgang beobachten. Die Kolonie selbst wächst durch Vermehrung auf Zellebene: Die einzelnen Zellen teilen sich, die Kugeloberfläche wächst. Vermehrung auf Kolonieebene findet hingegen durch spezialisierte Zellen auf der Kugeloberfläche statt, welche im Innenraum der Elternkolonie weitere Jungkolonien durch Zellteilung erzeugen. Sobald die Elternkolonie abstirbt, bricht ihre Kugelhülle auf und die Jungkolonien werden automatisch in die Freiheit entlassen. *Zellkolonien* wie diese können als Zwischenstufe zwischen einem *Schwarm von Einzellern* und einem *Mehrzeller* angesehen werden.

Äußerst ausgeprägte Formen von Schwarmverhalten zeigt das Reich der staatenbildenden Insekten [2, 9, 21] – die Zahl der Mitglieder solcher Staaten kann je nach Spezies um viele Größenordnungen variieren. Diese Zusammenarbeit und Spezialisierung im Schwarm kann so weit gehen, dass der individuelle Organismus darüber seine Überlebensfähigkeit aufgibt. Nester der Ameisenart *Myrmecocystus melliger* können beispielsweise mehr als tausend *Honigtöpfe* enthalten [27]: Honigtöpfe sind Ameisen, deren Kropf derart voll Honig gepumpt ist, dass ihr Hinterleib bis auf Erbsengröße anschwillt, so dass sie bewegungsunfähig als lebender Futterspeicher unter der Nestdecke hängen (Abb. 2.2 auf Seite 15). Trotz so extremer, für die Lebensfähigkeit einzelner

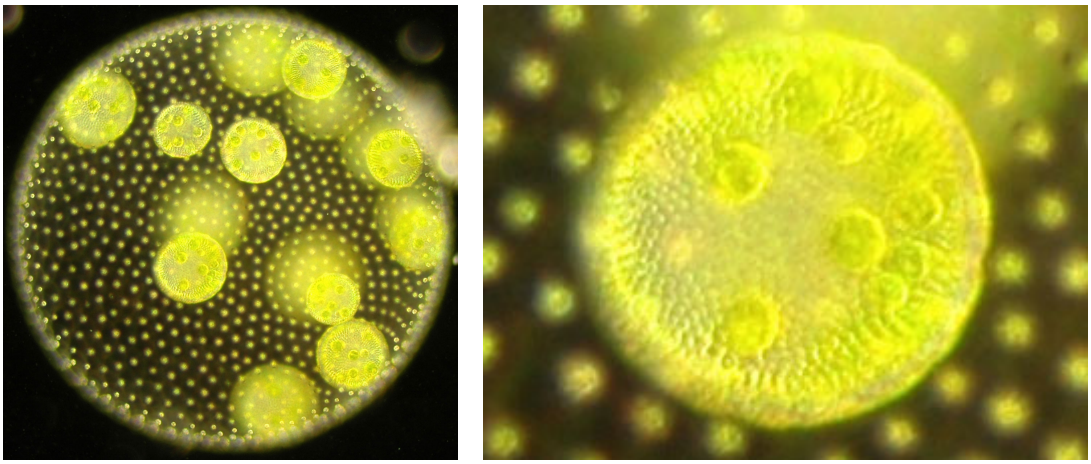


Abbildung 2.1: Links: Kugelalge der Art *Volvox carteri* mit mehreren Jungkolonien. Der Focus der Kamera ist so eingestellt, dass die Jungkolonien schärfer abgebildet sind als die Mutterkolonie, so dass bei den Jungkolonien die Zellstruktur besser sichtbar ist. **Rechts:** Vergrößerung einer solchen Jungkolonie. Bildvorlage entnommen aus der Bilder-Internetseite des Department of Applied Mathematics and Theoretical Physics, University of Cambridge, `http://www.damtp.cam.ac.uk/user/gold/movies.html`. Dieses Bild diente auch als Vorlage für das Themenbild des Kapitels auf Seite 13 oben.

Individuen nachteiliger Spezialisierungen ist der Schwarm als Superorganismus in der Lage, komplexe Aufgaben zu bewältigen. Zu diesen gehört beispielsweise, Behausungen mit ausgeprägten Strukturen zu errichten [61], welche eine effiziente Verteidigung gegen Fressfeinde und verschiedene Klimazonen für die Brut sowie die Lagerung oder gar den *Anbau* von Nahrung ermöglichen [68]. Da die Bewältigung solcher Aufgaben *keiner zentralen Steuerung* bedarf, erweist sie sich als *flexibel* und *fehlertolerant* [29].

Die Mitgliedschaft eines Individuums in einem Schwarm kann mit ganz unterschiedlichen Intensitäten der Individualitätsaufgabe verbunden sein. So sei als weiteres, weniger bis zur Invididualitätsaufgabe führendes Beispiel das Schwarmverhalten verschiedener Fischarten genannt [50] (Abb. 2.3 auf der rechten Seite): Die Individuen eines solchen Schwarms sind zwar durchaus alleine lebensfähig, allerdings ist durch die Mitgliedschaft im Schwarm ein gesteigerter Schutz vor Fressfeinden gegeben.

2.2 Synthetisiertes Schwarmverhalten

Beobachtungen und Dokumentationen von Verhaltensweisen wie den oben beschriebenen dienen als Inspiration für Studien, die zum Ziel haben, ähnliche Verhaltensmerkmale – und die dadurch induzierten Vorteile – auf künstliche Systeme zu übertragen. Hierzu werden die Individuen eines Schwarms durch künstliche Agenten modelliert. Die folgenden Abschnitte sollen einen Überblick über verschiedene Pfade bieten, die diese Studien eingeschlagen haben. Hierbei soll mit der Betrachtung idealisierten Schwarmverhaltens begonnen werden, bei welchem die Individuen nur auf ein sehr einfaches Repertoire an Basisverhaltensweisen zurückgreifen können.

2.2.1 Idealisierte Synthese

Algorithmen, die auf idealisiertem Schwarmverhalten basieren, finden mittlerweile erfolgreich als *Metaheuristiken* Anwendung, mit denen man versucht, sinnvolle Lösungen für komplexe Optimierungsprobleme zu finden. Explizit seien die beiden Optimierungsverfahren *Particle Swarm Optimization (PSO)* [31, 51] und *Ant Colony Optimization (ACO)* [16, 17] genannt.

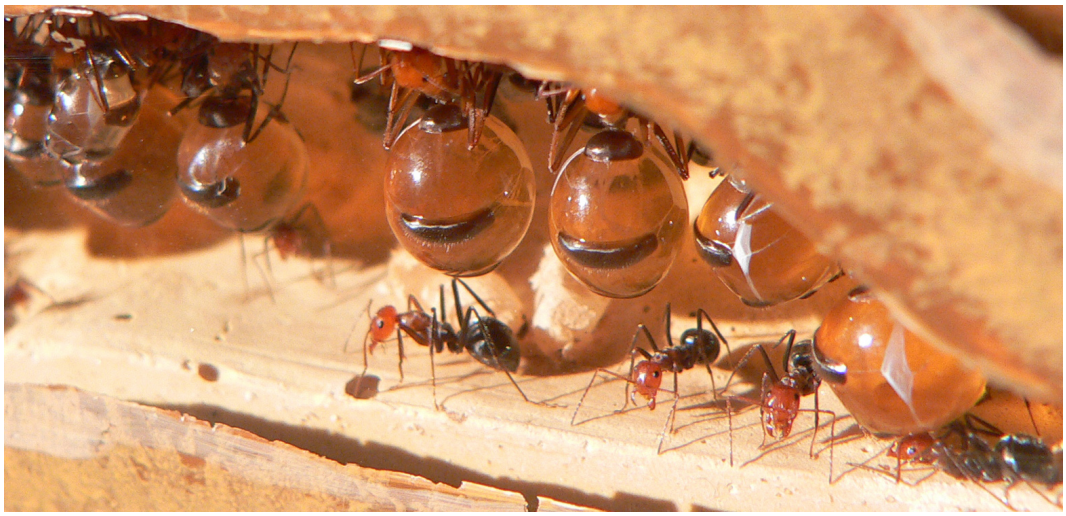


Abbildung 2.2: Honigtopfameisen (*Myrmecocystus melliger*) mit geschwollenen Hinterleibern, als lebender Futterspeicher bewegungsunfähig unter der Nestdecke hängend (oben im Bild). Unten im Bild ermöglichen andere Ameisen der Kolonie einen Vergleich der Hinterleibgröße zwischen aufgeschwollenem Zustand und Normalzustand. Quelle: Wikimedia Commons.

Abbildung 2.3: Schwarmverhalten von Fischen, illustriert anhand der auffällig gefärbten Gattung der Husarenfische. Quelle: Wikimedia Commons.

Der PSO-Algorithmus modelliert einen Schwarm von Individuen (*Partikeln*), die sich durch den Suchraum des zu optimierenden Problems bewegen. Die Partikel markieren diejenigen Punkte im Suchraum, an denen die Fitnessfunktion des zu optimierenden Problems ausgewertet wird. Die Lösungssuche, also die Bewegung eines jeden Partikels durch den Suchraum, basiert dann sowohl auf der Verarbeitung von *partikelspezifischen Informationen* (z.B. dem partikelspezifischen Fitness-Bestwert bis zu diesem Zeitpunkt), als auch auf *Interaktionen zwischen Partikeln* (z.B. können gute Fitnesswerte anderer Partikel die Bewegung ebenfalls beeinflussen). Das Ziel des Algorithmus ist, dass der Partikelschwarm sich – ähnlich eines futtersuchenden Vogelschwarms – auf diese Weise nahe zu einem Fitnessoptimum im Parameterraum bewegt. Mittlerweile wurde der PSO-Algorithmus um verschiedene Varianten erweitert: Als Beispiel sei ein von Wei et al. entwickeltes Hybridverfahren zwischen evolutionären Algorithmen und PSO genannt [66], welches für bestimmte Optimierungsprobleme sehr effizient arbeitet.

Der ACO-Algorithmus hingegen ist inspiriert von der Futtersuche nach Art der Ameisen: Ameisen beuten Futterquellen aus, indem eine Ameise, die Futter gefunden hat, von dort eine Spur von Duftstoffen (*Pheromonen*) zurück zum Nest legt. Dieser Pheromonspur können dann andere Ameisen vom Nest aus direkt folgen, und so ohne Suchaufwand helfen, die Futterquelle auszubeuten. Die vormals verschlungenen, zufällig anmutenden Suchpfade der Ameisen wandeln sich zu Pfaden, welche auf optimalerem Weg zum Futter führen. Mehr noch: Da auf kürzeren Wegen zu derselben Futterquelle öfter pro Zeit hin- und hergelaufen werden kann, werden evtl. gefundene kürzere Wege zum Futter stärker mit Pheromonen gekennzeichnet – und demzufolge automatisch bevorzugt. Diese Mechanismen bildet ACO nach und wendet sie auf graphenbasierte Optimierungsprobleme an.

Auch spezialisiertere Algorithmen als Metaheuristiken bauen auf idealisiertem Schwarmverhalten auf, wie beispielsweise schwarmbasierte *Clusteringverfahren* [26] oder Algorithmen im Umfeld der *Graphenpartition* [37].

2.2.2 Traditionelle Synthese

Im Rahmen der theoretischen Biologie wird versucht, in der Natur beobachtetes Schwarmverhalten direkt zu synthetisieren: Man leitet aus den Beobachtungen ein *Verhaltensmodell* für die einzelnen Individuen eines Schwarms ab, implementiert künstliche Individuen in Form von z.B. simulierten oder realen Robotern mit denselben Verhaltensregeln und überprüft, ob ein Schwarm dieser Individuen ähnliches Verhalten wie das natürliche Vorbild zeigt [12, 54, 60].

Hierbei kann von Interesse sein, *Entscheidungsprozesse* des Schwarms zu simulieren und zu analysieren: Auf welche Weise kommt der Superorganismus – aus vielen, sehr simplen Individuen mit jeweils lokaler Wahrnehmung bestehend – zu einer Ausschlussentscheidung, welche Futterquelle oder welcher neue Nistplatz genutzt werden soll oder auf welche Weise vorhandene Arbeit geteilt wird? Beispielsweise ist die Ameisenart *Lasius niger* dafür bekannt, solche Ausschlussentscheidungen zu treffen [4], weiterhin existieren Untersuchungen zu ähnlichen Ausschlussentscheidungen bei der Nistplatzsuche von Bienenschwärmen [58].

Außerdem bietet synthetisiertes Schwarmverhalten die Möglichkeit, das eigentlich beobachtete Verhalten in einen *Kontext von abgeänderten Varianten seiner selbst* zu setzen. So kann analysiert werden, welche beobachteten einzelnen Verhaltensmerkmale besonders zur Effizienz des Gesamtschwarms beitragen: Ein Beispiel hierfür findet sich in [33], wo Robotern die Möglichkeit gegeben wurde, nach dem Vorbild verschiedener Ameisenvölker aktiv weitere Roboter zu *rekrutieren*, was die Effizienz in der Futtersuche erhöht hat.

2.2.3 Evolutionäre Synthese

Ein anderer Ansatz als Verhalten manuell zu implementieren oder nachzubilden ist, es evolutionär entstehen zu lassen. Hierbei kommen evolutionäre Algorithmen zum Einsatz. Dies ist z.B. immer dann interessant, wenn man im Vergleich zum manuellen Design mehr Unvoreingenommenheit im Lösungsdesign wünscht oder explizit beobachten will, welchen Einfluss veränderte

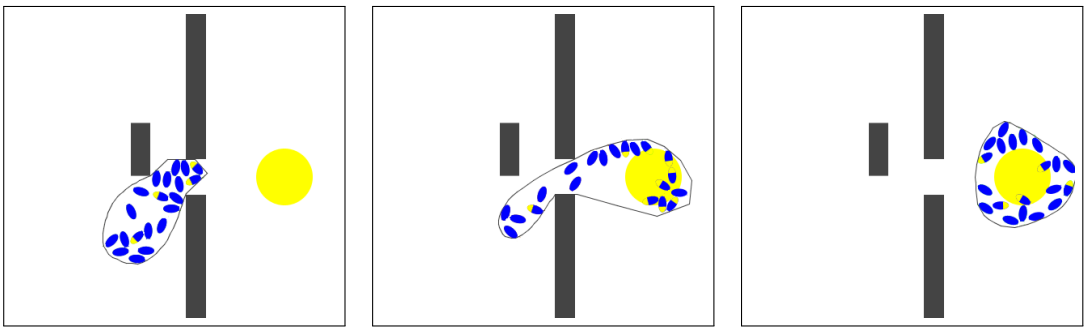


Abbildung 2.4: Simulation im Rahmen von „Beanbag Robotics“. Mikroroboter ohne eigene Navigationsfähigkeit in einer passiven Membran lernen evolutionär, im Schwarm zu navigieren und so an Hindernissen vorbei das Ziel (gelb dargestellt) zu erreichen.

Experimentbedingungen auf die Art der Lösung nehmen. Der evolutionäre Ansatz findet weite Verbreitung in den relativ jungen Gebieten der *evolutionären Robotik* und *Design-Automatisierung* und kann auch auf Schwarmverhalten und Schwarmrobotik übertragen werden. An dieser Stelle soll zunächst auf einige Arbeiten in der evolutionären Robotik eingegangen und anschließend zum evolutionären Schwarmverhalten übergegangen werden.

Im Bereich der evolutionären Robotik synthetisiert man Verhalten, indem man Kontrollstrukturen für z.B. autonome Roboter durch einen evolutionären Algorithmus einstellt, was oft zu unerwarteten, aber robusten Lösungen führt. Beeindruckende Resultate erzielten hier beispielsweise Pollack und Lipson, die im Rahmen des *GOLEM-Projektes* [40, 52] nicht nur die Kontrollstruktur, sondern zeitgleich auch die Morphologie von Robotern durch evolutionäre Algorithmen erzeugten und so in gewisser Weise künstliche Lebensformen schufen. Diese Arbeit trug nicht nur zur evolutionären Robotik, sondern auch zum Feld der Design- und Fertigungsautomatisierung bei, weil entstandene Roboter im Anschluss an die Evolution mittels Rapid-Prototyping-Techniken (umgangssprachlich: „3D-Druck“) gebaut wurden. So wurde die Hürde von Simulation zur Realität auf eine zum größten Teil automatisierte Weise überwunden.

Weitere Arbeiten nutzen evolutionäre Verhaltenssynthese, um für einen autonomen Roboter im Falle einer Beschädigung automatisiert neue Kontrollstrukturen entstehen zu lassen [6], die es ihm ermöglichen, im beschädigten Zustand zunächst ein neues Modell seiner selbst, und anschließend dazu passende, neue Arten der Fortbewegung zu erlernen (ähnlich einem Lebewesen mit gebrochenem Bein, welches sich hinkend fortbewegt).

Ein Beispiel aus dem Bereich der evolutionären Schwarmrobotik stellen unsere Arbeiten *Beanbag Robotics* [35, 36] dar, in welchen neuronale Kontrollstrukturen für Individuen eines Schwarms evolutionär synthetisiert werden. Diese Individuen sind extrem simpel modelliert, u.a. können sie *nicht lenken*. Hierdurch können sie äußerst klein und preiswert gebaut werden. Ein Schwarm dieser minimalistisch modellierten Roboter erlernt dann evolutionär, wieder Navigationsfähigkeit zu erlangen (Abb. 2.4). Ziel der Arbeit ist, durch Schwarmverhalten ein frei verformbares, redundantes robotisches System aus einfachsten Bestandteilen herzustellen.

Dorigo et al. hingegen schufen einen Roboter *Swarm-Bot*, welcher seinerseits aus mehreren einzelnen autonomen Robotern (*s-Bots*, welche im Unterschied zu z.B. den Individuen der Arbeit Beanbag Robotics voll navigationsfähig sind) zusammengesetzt ist [18, 62]. Die s-Bots können sich mittels spezieller Aktorik aneinanderketten, so zu einem Swarm-Bot verschmelzen und u.a. durch Evolution lernen, verschiedene Aufgaben im Schwarm besser zu lösen als auf der Individualebene.

Im Bereich *frei evolvierten* Schwarmverhaltens liefern Ward et al. [64] eine repräsentative Arbeit: Hier wurde das Verhalten einfacher Schwärmer in einer simulierten Umwelt in einer freien Evolution erzeugt, wobei kleine neuronale Netze als evolvierbare Kontrollstrukturen genutzt wurden. Die Fitness eines Individuums wird hier ähnlich wie in der freien Natur ermittelt: Sie steigt z.B. an, wenn das Individuum in der Umwelt verteiltes Futter zu sich nimmt. Die Arbeit beinhal-

tet allerdings keine Experimente, inwiefern beispielsweise verschiedene Umwelten das evolvierte Verhalten oder die evolutionäre Nähe der Schwärmer zueinander beeinflussen.

2.3 Zusammenfassung, Einschränkungen und Einordnung

Zusammenfassend lässt sich sagen, dass Schwarmverhalten evolutionär wie traditionell bis jetzt zumeist sehr zielgerichtet synthetisiert wurde. Ziele waren beispielsweise die Nachbildung von Verhalten in der Natur oder auch die Ausbildung konkreter Fertigkeiten im Bereich der Robotik.

Tendenziell weniger Veröffentlichungen sind im Hinblick auf frei – also naturinspiriert – evolviertes Schwarmverhalten bekannt, in denen Individuen nicht auf ein explizites Ziel hin optimiert werden, sondern ihre einzige Aufgabe – wie in der Natur – die Entwicklung einer robusten Population ihrer Spezies ist. Beispielsweise sind die Möglichkeiten der evolutionären Verhaltensentstehung meist signifikant eingeschränkt hinsichtlich eines, mehrerer oder gar aller der folgenden Aspekte:

Sensorik, Aktorik und Morphologie von Individuen wird meist statisch definiert, also nicht evolviert, z.B. bei der Verhaltensevolution für im Vorfeld fest definierte Roboter.

Kommunikations- und Interaktionsmöglichkeiten zwischen Individuen sind meist entweder nicht vorhanden (bis auf z.B. möglicherweise auftretende Kollisionen) oder stark eingeschränkt.

Kontrollstrukturen der Individuen sind meist nur stark eingeschränkt evolvierbar, z.B. bei der reinen Evolution gewichteter Verbindungen zwischen Sensoren und Aktor, Kontrollstrukturen ohne innere Zustände, der Verwendung von Single Layer Perceptrons bzw. vergleichbarer neuronaler Modelle oder neuronalen Netzen mit statischer Topologie.

Homogenität: Meist werden nur Schwärme mit homogenen Individuen berücksichtigt, was Effekte wie Spezialisierungen bestimmter Individuen völlig ausklammert.

Speziell vorgegebene Evolutionsziele, z.B. robotische Systeme bestimmter Art zu erhalten oder aber partikuläre Aufgaben zu lösen (wie die Überwindung eines bestimmten Hindernisses), lassen meist keine allgemeine Verhaltensentwicklung zu.

Umfang von Simulationen: Wenn eine Simulation als Evaluationsfunktion der Evolution gewählt wird, so wird die Simulationszeit oft aus Gründen des Rechenaufwandes sehr kurz gewählt, da Evaluationsfunktionen naturgemäß sehr oft ausgeführt werden müssen.

Experimente, welche in den genannten Aspekten weniger beschränkt evolvierbar sind, wären von großer Bedeutung, um eventuelle Zusammenhänge zwischen Experimentparametern (insbesondere wechselnden Umwelten) und entstehenden Verhaltensparadigmen wie Spezialisierung und Kooperation im Schwarm auszumachen. Es könnte mit diesen Mitteln auf synthetische und daher genau analysierbare Art und Weise die Entstehung von Schwarmverhalten aller Art erforscht werden – einem Phänomen, welches in sämtlichen Lebensbereichen allgegenwärtig ist.

Ein Schritt in diese Richtung soll im Rahmen dieser Diplomarbeit gemacht werden. Es sollen Erfahrungen in Bezug auf freiere Evolutionen von Schwärmen exemplarischer Modellschwärmer mit naturinspirierten Eigenschaften gewonnen werden. Die Verhaltensentstehung im Rahmen der Evolutionen soll im Hinblick auf die oben identifizierten Aspekte weniger eingeschränkt sein.

Kapitel 3

Grundlagen

In diesem Kapitel sollen Grundlagen von wichtigen, in dieser Diplomarbeit verwendeten Paradigmen beschrieben werden. Hierzu gehören künstliche neuronale Netze, die als Kontrollstrukturen der Schwärmer zum Einsatz kommen sollen (Abschnitt 3.1). Da beabsichtigt ist, Schwarmverhalten durch evolutionäre Optimierung entstehen zu lassen, folgt in Abschnitt 3.2 eine Beschreibung der diesbezüglichen Grundlagen.

3.1 Neuronale Netze

Biologische Nervensysteme bestehen aus einzelnen Nervenzellen, den biologischen Neuronen. Die Neuronen verarbeiten auf sehr einfache Weise Informationen, was eine Dateneingabe und eine Datenausgabe voraussetzt. Als Eingaben dienen elektrische Potentiale, die beispielsweise von anderen Neuronen kommen können: Neuronen können also miteinander *verbunden* sein. Die durch Neurone und ihre Verbindungen geschaffene Struktur nennt man auch *neuronale Netze*.

In einem Neuron werden die eintreffenden elektrischen Potentiale zunächst aufkumuliert. Sobald das aufkumulierte Potential einen gewissen Schwellenwert überschreitet, sendet das Neuron seinerseits ein Potential aus, das wiederum von anderen Neuronen verarbeitet werden kann. Sendet ein Neuron ein Signal aus, so spricht man davon, dass es *aktiviert* wurde. Ein Neuron bildet also gewissermaßen eine vektorielle Eingabe mittels einer nichtlinearen Abbildung auf eine skalare Ausgabe ab.

Ein biologisches Nervensystem besteht aus vielen solcher Neuronen [30], die untereinander mittels synaptischer Verbindungen verbunden sein können, oder auch nicht. Ein von einem Neuron erzeugtes Potential kann ein anderes Neuron nur dann erreichen, wenn eine Verbindung vom Erzeugerneuron zum Empfängerneuron existiert. Die genannten Verbindungen sind gerichtet. Die koordinierte Erzeugung solcher Potentiale von vielen parallel arbeitenden, einfachen Neuronen bildet dann eine sehr leistungsfähige informationsverarbeitende Struktur, wie man sie z.B. beim menschlichen Gehirn antreffen. Dort arbeiten ca. 10^{11} Neurone parallel [10, 30].

Solch ein biologisches Nervensystem ist nicht nur redundant ausgelegt und demzufolge läsionstolerant (der Verlust einzelner Nervenzellen hat wenig Einfluss auf das Gesamtsystem), sondern auch lernfähig: Die angesprochenen Verbindungen zwischen Neuronen können verstärkenden bzw. hemmenden Einfluss auf das Zielneuron haben. Die Art und die Stärke dieses Einflusses kann im Laufe der Zeit verändert werden, so dass das Gesamtsystem Informationen anders verarbeitet als zuvor. Die Veränderung der Informationsverarbeitung auf der Zeitachse bezeichnet man als *lernen*.

3.1.1 Bausteine künstlicher neuronaler Netze

Künstliche neuronale Netze[1] (im Folgenden ebenfalls als *neuronale Netze* bezeichnet) [34,72] sind informationsverarbeitende Strukturen, welche grob den biologischen Nervensystemen nachempfunden sind. Sie sind universelle Funktionsapproximatoren und geschaffen worden, um die Vorteile der genannten biologischen Systeme, insbesondere die massive Parallelität, die Läsionstoleranz und die Lernfähigkeit in künstlichen Systemen nutzbar zu machen. Sie bestehen ähnlich ihren biologischen Vorbildern aus kleinen, informationsverarbeitenden Einheiten (*Neuronen*) sowie Verbindungen zwischen den Neuronen, welche auch als *Synapsen* bezeichnet werden. Synapsen sind mit einer Zahl *gewichtet*: Ein negatives Vorzeichen des Gewichtes markiert hierbei eine Synapse als hemmend, ein positives Vorzeichen als verstärkend. Die Funktionsweise und Bedeutung von Neuronen und Synapsen soll nun näher betrachtet werden.

Wie bei den biologischen Neuronen lässt sich die Informationsverarbeitung der künstlichen Neurone in zwei Phasen unterteilen: Das Aufkumulieren eines Vektors von Eingaben (den aufkumulierten Wert bezeichnet man als *Netzeingabe*) sowie das Berechnen einer skalaren Ausgabe aus der Netzeingabe. Diese Aufgaben werden von zwei mathematischen Funktionen übernommen: Der *Propagierungsfunktion* und der *Aktivierungsfunktion*. Für beide Funktionen existiert eine Vielzahl von Definitionsmöglichkeiten. Exemplarisch soll jeweils eine Definitionsmöglichkeit genannt werden, welche auch in dieser Diplomarbeit Verwendung findet.

Im Vorhinein soll noch etwas Nomenklatur betrieben werden: Sei j das Neuron, von dem nun die Informationsverarbeitung betrachtet wird. Sei I_j die Menge derjenigen Neurone, welche eine Verbindung mit Neuron j als Ziel besitzen und i ein Neuron aus I_j. Das Gewicht einer Verbindung von einem Neuron i zu einem Neuron j sei benannt mit $w_{i,j}$. Sei net_i die Netzeingabe eines Neurons i und sei o_i dessen skalare Ausgabe.

Die Propagierungsfunktion $f_{\mathrm{prop}}(I_j)$ des betrachteten Neurons j errechnet die *Netzeingabe* des Neurons, net_j. Als Propagierungsfunktion wird im Folgenden die *gewichtete Summe* verwendet, welche in der Literatur breite Verwendung findet:

$$\mathrm{net}_j = f_{\mathrm{prop}}(I_j) = \sum_{i \in I_j} w_{i,j} \cdot o_i.$$

Die Aktivierungsfunktion errechnet aus der Netzeingabe net_j des betrachteten Neurons dessen Ausgabe o_j. Als Aktivierungsfunktion $f_{\mathrm{act}}(\mathrm{net}_j)$ wird im Folgenden der ebenfalls an vielen Stellen verwendete Tangens Hyperbolicus (Abb. 3.1 auf der rechten Seite) verwendet:

$$o_j = f_{\mathrm{act}}(\mathrm{net}_j) = \tanh(\mathrm{net}_j).$$

Die Stelle der stärksten Steigung der Aktivierungsfunktion ist in der Regel die 0. Durch Subtraktion eines pro Neuron j individuellen Schwellwertes Θ_j (*Bias*) von der Netzeingabe net_j lässt sich diese Stelle – und damit der „Schaltpunkt“ eines Neurons – beliebig entlang der net_j-Achse verschieben (in diesem Fall auf den Punkt Θ_j).

Ähnlich den natürlichen Vorbildern lernen auch künstliche neuronale Netze durch die Modifikation der Gewichte $w_{i,j}$. Dies geschieht mittels verschiedener Lernverfahren. Ein kurzer Überblick über verschiedene Arten von Lernverfahren wird in Abschnitt 3.1.5 gegeben.

Bei der Implementierung eines neuronalen Netzes realisiert man den Schwellwert Θ_j eines Neurons j oft durch eine Verbindung mit dem Gewicht $-\Theta_j$ von einem virtuellen *Biasneuron* zum Neuron j. Das Biasneuron hat keine Eingaben und seine Ausgabe beträgt immer 1. Da man also die Schwellwerte als Verbindungsgewichte auffassen kann, werden sie hier nicht weiter explizit betrachtet.

[1] Im Rahmen dieses Abschnittes wird nur der Aufbau und die Funktion „perzeptronartiger“ Netze motiviert und beschrieben – präziser ausgedrückt, Multilayer-Perzeptrone, welche zusätzlich aber noch über Rekurrenzen verfügen dürfen. Dem Diplomanden ist bewusst, dass noch etliche weitere Arten künstlicher neuronaler Netze existieren. Es würde jedoch zu weit führen, die hier verwendeten (und daher in diesem Abschnitt beschriebenen) neuronalen Netze in ihren vollen Kontext zu stellen.

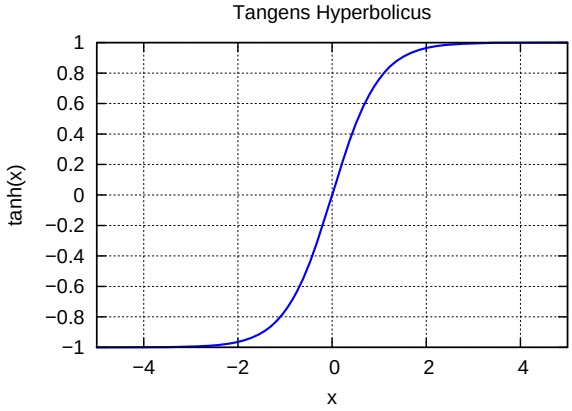


Abbildung 3.1: Der hier als Aktivierungsfunktion verwendete Tangens hyperbolicus. Für die Variable x wird die Netzeingabe net_j eines Neurons j eingesetzt.

3.1.2 Perzeptrone und Abkürzungsverbindungen

Die graphenähnliche Struktur aus Verbindungen und Neuronen impliziert, dass ein neuronales Netz verschiedene *Topologien* innehaben kann und ferner, dass die Art der Topologie essentiellen Einfluss auf die Datenverarbeitung im neuronalen Netz hat. Im folgenden sollen einige häufige Topologien vorgestellt werden. Hierbei werden Neurone, die Eingaben in das Netz entgegennehmen, mit dem Buchstaben i bezeichnet, Neurone, welche Ausgaben aus dem Netz heraus abgeben mit Ω, sowie von außen nicht sichtbare, also versteckte (hidden) Neurone mit h. Diese Art der Benennung von Neuronen folgt derjenigen aus [34]. Hierbei ist zu beachten, dass Eingabeneurone in der Regel die Eingabe nur weitergeben, also die Identitätsfunktion repräsentieren.

Eine der einfachsten denkbaren Arten neuronaler Netze besteht aus einer Menge Eingabeneuronen und einer Menge Ausgabeneuronen, sowie Verbindungen von Eingabe- zu Ausgabeneuronen. Versteckte Neuronen existieren in dieser Art neuronaler Netze nicht. Aufgrund der Darstellungsweise eines solchen neuronalen Netzes haben sich für diese Neuronenmengen die Namen *Eingabeschicht* bzw. *Ausgabeschicht* etabliert. Da ein Netz dieser Art genau eine Gewichtsschicht besitzt, auf die ein Lernverfahren angewendet werden kann, wird es auch als *einschichtiges Perzeptron* bezeichnet. Einschichtige Perzeptrone können nur linear separierbare Funktionen repräsentieren [47].

Analog zu den einschichtigen Perzeptronen existieren *mehrschichtige Perzeptrone*, eine sehr verbreitet verwendete Art neuronaler Netze. Sie enthalten in weiteren Neuronenschichten zwischen Eingabe- und Ausgabeneuronenschicht noch beliebig viele *versteckte Neuronenschichten*, wobei die Verbindungen wieder an den Schichten orientiert sind, bzw. immer von einer zur nächsten Schicht verlaufen (in Abb. 3.2 auf der folgenden Seite gepunktet dargestellt). Mehrschichtige Perzeptrone sind nicht mehr auf linear separierbare Funktionen beschränkt.

Verbreitet sind auch mehrschichtige Perzeptrone mit sogenannten *Abkürzungsverbindungen*, also Verbindungen, welche mindestens eine Neuronenschicht überspringen (in Abb. 3.2 durchgezogen dargestellt). Solche Verbindungen erhöhen zwar nicht die Mächtigkeit eines Netzes, können aber helfen, Probleme für das Netz einfacher erlernbar zu machen.

3.1.3 Rekurrente neuronale Netze

Bis jetzt wurden nur vorwärtsgerichtete neuronale Netze betrachtet: Verbindungen existieren aus Richtung Eingabeschicht in Richtung Ausgabeschicht. Netze dieser Art sind in der Lage, eine Eingabe einer eindeutig bestimmten Ausgabe zuzuführen.

Ein neuronales Netz kann jedoch auch mehr Verbindungstypen als nur die vorwärtsgerichteten besitzen: *Laterale* Verbindungen existieren zwischen Neuronen innerhalb derselben Schicht, weiter

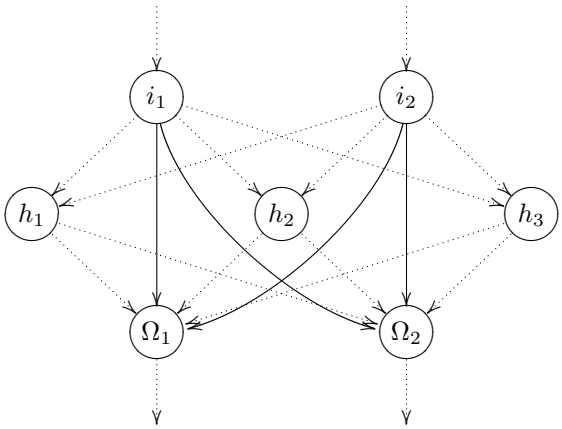


Abbildung 3.2: Ein mehrschichtiges Perzeptron mit durchgezogen dargestellten Abkürzungen. Verbindungen zu Eingabeneuronen markieren Eingaben, Verbindungen von Ausgabeneuronen aus markieren Ausgaben.

sind *rückwärtsgerichtete* Verbindungen, *Rückwärts-Abkürzungen* oder auch *Selbstverbindungen* denkbar (Abb. 3.3 auf der rechten Seite). Existieren solche Verbindungen, so nennt man ein neuronales Netz *rekurrent*: Seine Ausgabe hängt nicht mehr nur von der Eingabe, sondern auch von seinem inneren Zustand ab. Solche Netze sind in der Regel ungleich schwieriger zu trainieren und zu designen, da sie beachtliche Dynamiken bis hin zu chaotischem Verhalten entwickeln können. Auch sind hier (mit Ausnahme von Eingabeschicht und Ausgabeschicht) die Neuronenschichten nicht mehr eindeutig anhand der Topologie definiert. Die Aktivierungsreihenfolge wird meist so definiert, dass zunächst die Eingabeneurone aktiviert werden (also die Daten in das Netz eingegeben werden), im Anschluss in einer (meist implementierungsabhängigen) Reihenfolge die versteckten Neurone aktiviert werden, und zum Schluss die Ausgabeneurone.

3.1.4 Propagierung von Daten durch neuronale Netze

Die Ausgaben eines Neuronalen Netzes werden zu diskreten Zeitschritten berechnet. In einem Zeitschritt t wird für jedes Neuron zunächst die Netzeingabe berechnet. Zur Berechnung der Netzeingabe net_j eines Neurons j werden die Ausgaben der zu j verbundenen Neurone aus dem Zeitschritt $(t-1)$ herangezogen. Zu j verbundene Eingabeneurone liefern hier die aktuell angelegte Eingabe, hier erfolgt also die *Dateneingabe* in das neuronale Netz. Anschließend wird für jedes Neuron j seine Aktivierungsfunktion ausgeführt, und so die Ausgabe o_j berechnet. So berechnete Ausgaben von Ausgabeneuronen bilden die *Datenausgabe* des neuronalen Netzes.

3.1.5 Lernverfahren für neuronale Netze

Wie oben schon bemerkt, lernt ein gegebenes neuronales Netz durch Einstellung seiner Gewichtswerte. Zu diesem Zweck gibt verschiedene Lernverfahren. Ein populäres Gradientenabstiegsverfahren, welches weite Verbreitung findet, ist beispielsweise *Backpropagation of Error* [55]. Es eignet sich in seiner ursprünglichen Version jedoch nur für nicht-rekurrente neuronale Netze und bringt weiterhin die für Gradientenabstiegsverfahren typischen Probleme mit sich, wie zum Beispiel die Konvergenz nach schlechten Minima oder die Stagnation des Lernprozesses bei kleinen Gradientenbeträgen. Zudem kann man mit Backpropagation (wie mit vielen anderen Lernverfahren, insbesondere Gradientenverfahren) nur die Gewichte eines bereits existierenden neuronalen Netzes trainieren. Hier muss die Topologie also vorgegeben werden.

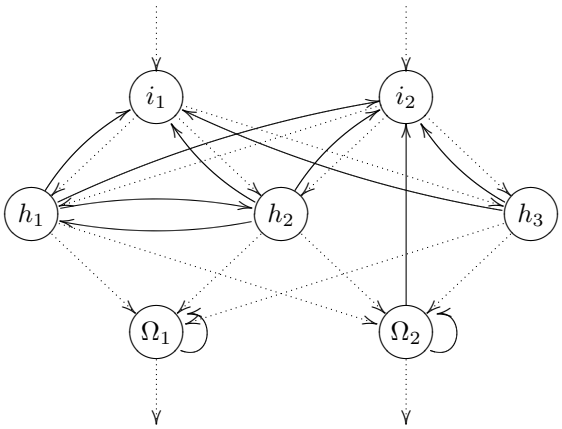


Abbildung 3.3: Ein mehrschichtiges Perzeptron mit beispielhaften, durchgezogen dargestellten Rekurrenzen. Bei beiden Ausgabeneuronen existiert jeweils eine Selbstverbindung. Von Ausgabeneuron Ω_2 zu Eingabeneuron i_2 existiert eine Rückwärtsabkürzung. Zwischen den Neuronen h_1 und h_2 existieren laterale Verbindungen. Sämtliche Vorwärtsverbindungen (nicht: Vorwärts-Abkürzungen), welche von der Eingabeschicht ausgehen, werden von Rückwärtsverbindungen erwidert.

Verfahren, welche ausschließlich die Gewichte eines neuronalen Netzes trainieren, setzen jedoch voraus, dass man die grundlegende Topologie und auch die Neuronenzahl im Vorhinein sinnvoll wählen kann. Da dies nicht immer möglich ist, existieren in der Literatur Ansätze – beispielsweise mittels evolutionärer Optimierung – nicht allein die Gewichtswerte, sondern auch die Topologie eines neuronalen Netzes zu trainieren [69,70]. Da auch im Verlauf dieser Diplomarbeit so vorgegangen werden soll, werden in Abschnitt 3.2 die grundlegenden Abläufe einer solchen evolutionären Optimierung vorgestellt. Einzelheiten der Anwendung der evolutionären Optimierungsverfahren auf neuronale Netze sowie weitere Literaturstellen in diesem Zusammenhang werden in der Dokumentation der eigentlichen Implementierung des neuronalen Netzes behandelt (Abschnitt 5.3).

3.2 Evolutionäre Optimierung

Als evolutionäre Optimierung bezeichnet man eine Metaheuristik, die von der natürlichen Evolution inspiriert ist. Sie wird heutzutage an vielen Stellen eingesetzt, um plausible Lösungen für schwere, nichtlineare Optimierungsprobleme zu finden [48]. Es existiert eine Fülle von Veröffentlichungen, die etwa bestehende evolutionäre Optimierungsverfahren abwandeln oder neue Metaheuristiken präsentieren, welche von der natürlichen Evolution inspiriert sind. Dieser Abschnitt soll sich keiner speziellen Schule und keinem partikulären Algorithmus anschließen, sondern einen kurzen, abstrakten Überblick über Merkmale liefern, welche den meisten evolutionären Optimierungsverfahren gemein sind. Zu diesem Zweck ist es sinnvoll, zunächst die biologischen Grundlagen der Evolution zu skizzieren.

3.2.1 Biologische Grundlagen

Wie heutzutage bekannt ist, trägt jedes Lebewesen einen komprimierten Bauplan seiner selbst in sich [65]: Das sogenannte *Genom*. Vereinfacht gesprochen besteht ein Genom aus einer Menge von Parametern (*Genen*), welche sehr viele Merkmale des zugehörigen Lebewesens definieren. Die Gesamtheit aller Genome einer ganzen *Population* von Lebewesen nennt man *Genpool*.

Innerhalb einer Population gibt es Lebewesen, die in der Lage sind, „erfolgreicher“ zu leben als andere. „Erfolg“ bedeutet in der Natur, die eigene Spezies möglichst effizient zu erhalten, also Nachkommen zu zeugen. Ein Lebewesen kann erfolgreicher sein als ein anderes, indem es besser an einen bestimmten Lebensraum angepasst ist: In einer Wüste ist es beispielsweise ein erfolgversprechendes Merkmal, Nahrungsmittel, insbesondere Wasser, sehr effizient zu verbrauchen. Ein Lebewesen, in dessen Genom spezielle Merkmale zur effizienten Wasserspeicherung einkodiert sind, kann sich in der Wüste durch höhere Überlebenschancen auszeichnen, also insbesondere tendenziell mehr Nachkommen haben. Durch diesen Mechanismus, die sogenannte *Selektion*, wird ein Genom, welches sich in der Umwelt bewährt hat, gegenüber schlechter geeigneten Genomen überproportional verbreitet.

Neben der Selektion gibt es einen zweiten wichtigen Aspekt, der zum Verständnis der Evolution relevant ist: Der Genpool ist nicht statisch. Zum einen enthalten, falls sich Lebewesen sexuell vermehren, die Genome der Nachkommen Merkmale aus beiden Elterngenomen (man spricht hier von der *Rekombination* von Genomen), zum anderen können auch einzelne Gene durch Faktoren wie z.B. die kosmische Hintergrundstrahlung verändert werden (hier spricht man von *Mutation*). Es kommen also laufend neue Genome zum Genpool hinzu, andere Genome verlassen den Genpool hingegen. Neue, hinzukommende Genome entstehen durch Rekombination und Mutation aus bereits vorhandenen Elterngenomen, wobei – wie oben dargestellt – erfolgreiche Genome überproportional oft als Eltern dienen. Weniger erfolgreiche Genome sterben hingegen überproportional aus.

Das Zusammenspiel von Selektion, Rekombination und Mutation bewirkt, dass die Überlebensfähigkeit einer Population von Lebewesen in derem Umwelt optimiert wird – die Lebewesen passen sich laufend graduell ihrer Umwelt an [14]. Diese Art der Optimierung versucht man durch evolutionäre Optimierungsverfahren nachzubilden und so für die Lösung verschiedenster Probleme nutzbar zu machen.

3.2.2 Technische Adaption biologischer Evolution

Wie oben erwähnt, versucht man evolutionäre Optimierungsverfahren auf *Optimierungsprobleme* anzuwenden. Optimierungsprobleme sind dadurch gekennzeichnet, dass sie parametrisierbar sind: Ein Lösungskandidat des Optimierungsproblems wird durch einen Parametersatz (sein *Genom*) definiert. Weiter kann man für einen partikulären Lösungskandidaten die Güte messen, es steht uns also eine *Evaluationsfunktion* zur Verfügung, welche Elemente des Parameterraums auf Fitness- bzw. Kostenwerte abbildet. Wie in der Natur lässt sich also messen, wie erfolgreich ein Genom ist – oder technisch ausgedrückt, wie gut ein bestimmter Parametersatz das gegebene Optimierungsproblem löst. Evolutionäre Optimierungsverfahren operieren nun – wie die Natur – auf Grundlage eines *Genpools*, der aus den Genomen einer Population von Lösungskandidaten besteht. Diese Population wird meist zufällig initialisiert.

Einmal initialisiert, durchläuft die Population von Lösungskandidaten verschiedene Phasen, die nun abstrakt beschrieben werden:

Evaluation: Für jeden Lösungskandidaten in der Population wird die Evaluationsfunktion ausgeführt.

Selektion: Anhand der Evaluationswerte werden aus der aktuellen Population Elterngenome ausgewählt. Genome mit höherer Güte werden hierbei bevorzugt ausgewählt (beispielsweise anhand einer Wahrscheinlichkeitsverteilung). Es gibt eine Fülle von Veröffentlichungen, welche verschiedene Selektionsstrategien präsentieren [48].

Erneuerung der Population: Die Population wird erneuert (man spricht hier von der nächsten *Generation*), indem die zuvor selektierten Elterngenome *rekombiniert* und *mutiert* werden. Für Rekombination und Mutation sind in der Regel problemspezifische Operatoren zu entwerfen. Weite Verwendung findet an dieser Stelle die als *Elitismus* bekannte Vorgehensweise,

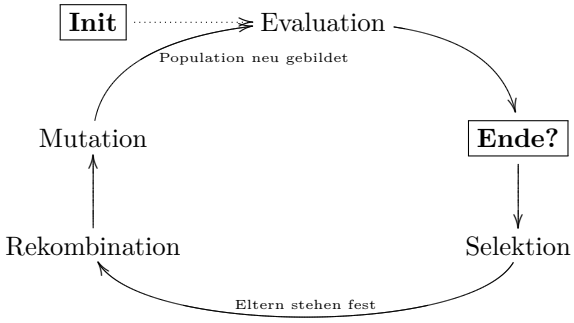
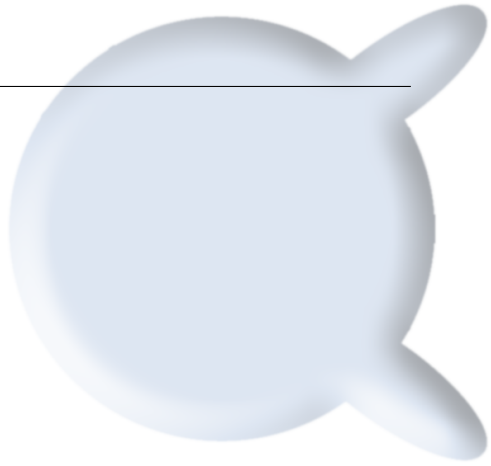


Abbildung 3.4: Illustration des im Text skizzierten Ablaufschemas evolutionärer Optimierung. Zunächst wird eine Population von Lösungskandidaten initialisiert (z.B. durch zufällig generierte Lösungskandidaten). Dies wird durch den Kasten *Init* symbolisiert und stellt den Einsprungpunkt in das Schema dar. Nach einer Evaluation aller Problemlösungskandidaten der aktuellen Generation kann der Algorithmus von außen abgebrochen werden, sofern die bis zu diesem Zeitpunkt erzielten Resultate für den Benutzer zufriedenstellend sind. Diese Abbruchmöglichkeit wird durch den Kasten *Ende?* symbolisiert. Die Ausdrücke *Rekombination* und *Mutation* repräsentieren die Erneuerung der Population.

die besten Genome aus der aktuellen Population direkt in die neue Population zu übernehmen[2]. Auch hier findet man aber eine Fülle von Veröffentlichungen, welche verschiedene Strategien zur Erzeugung der nächsten Generation präsentieren [48].

Die Gesamtheit dieser drei Phasen wird in der Regel iteriert (Abb. 3.4), bis ein bestimmtes Kriterium erfüllt ist (z.B. eine im Vorfeld definierte Problemlösungsgüte erreicht wird oder eine vorgegebene Anzahl von Generationen durchlaufen ist). Man hofft, dass sich die Population der Lösungskandidaten analog zur natürlichen Evolution einem Optimum auf der Evaluierungsfunktion nähert.

[2] Hierdurch können potentiell ewig lebende Genome auftreten, weswegen das Elitismus-Verfahren sich etwas von der natürlichen Motivation wegbewegt.



Kapitel 4

Modellierung

In diesem Kapitel werden die verschiedenen Aspekte der in dieser Diplomarbeit beschriebenen, konkreten Evolution heterogener Schwärme modelliert, insbesondere auch die konkreten Merkmale der Individuen innerhalb der Schwärme. Bei diesen handelt es sich um rein exemplarische Phantasiewesen, jedoch mit natürlich inspirierten Merkmalen. Die hier vorgestellte Modellierung ist losgelöst von der eigentlichen Implementierung und sonstigen technischen Problemstellungen zu verstehen, welche in Kapitel 5 behandelt werden.

Es ist zu beachten, dass die später beschriebenen, implementierten Softwarebestandteile nahezu beliebige Umwelten und Individuen erlauben, von denen die hier vorgestellte Modellierung lediglich eine Konkretisierung darstellt.

4.1 Schwarm und Subschwärme

Im Rahmen dieser Arbeit sollen Morphologieaspekte und Kontrollstrukturen für einen *Schwarm*, also eine Menge von Individuen (welche auch *Schwärmer* genannt werden) evolutionär optimiert werden. Der so evolvierte Schwarm soll aus drei *Subschwärmen* bestehen, welche disjunkte Untermengen des Schwarms repräsentieren. Innerhalb eines Subschwarms sind die Individuen identisch, während die Eigenschaften der Schwärmer über Subschwärme hinweg variieren können. Es könnten so beispielsweise Subschwärme mit Spezialfertigkeiten entstehen, die kooperatives Verhalten untereinander zeigen könnten, so dass die Spezialfertigkeiten dem gesamten Schwarm zugute kämen.

4.2 Der parametrisierbare Schwärmer

Für die Realisierung der Schwärmer wird ein parametrisierbarer Schwärmer vorgeschlagen. Instanzen dieses Schwärmers werden dann im Rahmen der evolutionären Optimierung erzeugt und simuliert. Die Parameter des Schwärmers lassen sich grob in vier Klassen unterteilen:

- 1 Körperparameter: n,
- 7 Aktorikparameter: $\ell_f, \ell_n, t_p, f_{\max}, f_{\text{soz}}, s, t_s$,
- 3 Sensorikparameter: α, β, γ,
- Parameter des neuronalen Netzes: Neuronenzahl, Anzahl der Verbindungen, synaptische Gewichte.

Details zu den Parametern werden in den folgenden Abschnitten beschrieben.

4.2.1 Körper

In diesem Abschnitt werden generelle Körpereigenschaften des Schwärmers beschrieben, die dessen Verhalten beeinflussen können.

Futterspeicher n: Der Körper bietet einen gewissen Raum der Größe n, in welchem Futter gespeichert werden kann. Wenn der Speicher voll ist, kann kein weiteres Futter mehr aufgenommen werden. Wird n auf große Werte gesetzt, kann der Schwärmer also mehr Futter aufnehmen und sich so z.B. zu Explorationszwecken weiter als andere Schwärmer von Futterquellen entfernen.

Nahrungsverbrauch: Die Menge des im Speicher vorhandenen Futters wird kontinuierlich verringert, da Leben Energie verbraucht. Ist die Menge verbliebenen Futters im Speicher bei 0 angelangt, stirbt der Schwärmer durch Verhungern.

Tod: Schwärmer sterben durch Futtermangel oder nach einer maximalen Lebensdauer. Diese Lebensdauer ist signifikant kürzer als die Dauer einer einzelnen Simulation, so dass die Schwärmer Nachkommen hervorbringen müssen, um den Schwarm über die Simulationszeit hinweg aufrecht zu erhalten.

4.2.2 Aktorik

Ein Schwärmer hat verschiedene Möglichkeiten, auf seine Umwelt Einfluss zu nehmen (Aktorik). Die Aktorik eines Schwärmers wird durch die folgenden Parameter definiert:

Maximale Schrittweite f_{max} und Fortbewegung: Der Schwärmer soll in der Lage sein, sich im Rahmen der obigen Körperparameter vorwärts und rückwärts mit variabler Geschwindigkeit fortzubewegen und seine Orientierung zu verändern. Ein Schwärmer besitzt eine maximale Schrittweite $f_{\max}$. Der Schwärmer kann seine Schrittweite kontinuierlich innerhalb des Intervalls $[-f_{\max}; f_{\max}]$ wählen, er kann sich also vorwärts und rückwärts entlang seiner aktuellen Orientierung bewegen. Das kontinuierliche Wählen von großen Schrittweiten nahe $f_{\max}$ lässt den Schwärmer sich schneller fortbewegen.

Maximale soziale Interaktionskraft f_{soz}: Bedingt durch die einfache Architektur der Schwärmer kann es von Nutzen sein, die Entstehung von schwarmhaften Bewegungen zu fördern. Es wird also einem Schwärmer die Möglichkeit gegeben werden, auf andere, von ihm wahrgenommene Schwärmer eine kleine, schwarmbildende Kraft auszuüben. Diese Art Kraft ist abgeleitet von „Social Interaction Forces“, welche aus Attraktions- und Repulsionskomponenten bestehen und als modellhafte Erklärung für Aggregations- und Bewegungsverhalten von Bakterienensembles und Fischschwärmen zum Einsatz kommen [39, 49]. Große Werte von f_{soz} lassen also eine stärkere Anziehung und Abstoßung der Schwärmer untereinander zu. Die durch f_{soz} induzierte Bewegung anderer Schwärmer soll tendenziell kleiner sein als die Schrittweite, die ein Schwärmer zur individuellen Fortbewegung zurücklegen kann. Der Schwärmer kann die auszuübende soziale Interaktionskraft kontinuierlich innerhalb des Intervalls $[-f_{\text{soz}}; f_{\text{soz}}]$ wählen und so auch steuern, ob er andere Schwärmer tendenziell abstößt oder anzieht.

Sprint s, t_s: Es soll eine Sprintfähigkeit geben, die es dem Schwärmer ermöglicht, sich bis zu s viele Schritte mit verdoppelter Geschwindigkeit (Sprintschritte) fortzubewegen. Nach einer Sprintetappe muss der Schwärmer die Sprintfähigkeit zunächst wieder regenerieren, was pro ausgeführtem Sprintschritt eine Zeit t_s in Anspruch nimmt. Große Werte von s und kleine t_s würden also dafür sorgen, dass ein Schwärmer viele Schritte hintereinander sprinten und sich schnell wieder regenerieren kann.

Fernkommunikation ℓ_f: Der Schwärmer kann RGB-Signale[1] versenden, die von anderen Schwärmern wahrgenommen („gesehen“) werden können, sofern sie in die Richtung

[1] Bei jeglicher, hier beschriebener Form von RGB-Kommunikation handelt es sich prinzipiell um drei voneinander unabhängige Signalkanäle, welche sich als RGB-Farbe ansprechend visualisieren und erklären lassen. Die verschiedenen Kanäle ermöglichen den Schwärmern, auf einfache Weise verschiedenartige Signale abzusetzen.

des Senders blicken (angelehnt an z.B. Lichtsignale). Mit zunehmendem Abstand d von der Signalquelle nimmt die Intensität dieses Signals wie in der Natur quadratisch ab. Ein Schwärmer besitzt jedoch eine individuelle maximale Fernkommunikations-„Leuchtstärke“ ℓ_f, mit der er seine Signale versendet. Die Signalstärke nimmt also insgesamt mit $\frac{\ell_f}{d^2}$ ab. Große Werte von ℓ_f ermöglichen es einem Schwärmer also, intensivere Fernkommunikationssignale zu versenden.

Nahbereichskommunikation ℓ_n: Es können RGB-Signale ausgesandt werden, die wie die Fernkommunikationssignale eine gewisse individuelle maximale Intensität ℓ_n besitzen und von anderen Schwärmern empfangen werden können – allerdings unabhängig von der Blickrichtung. Dies ist inspiriert durch z.B. akustische Kommunikation in Schwärmen. Analog zur Fernkommunikation nimmt die Signalstärke auch hier quadratisch mit der Distanz ab. Wird ℓ_n auf große Werte gesetzt, kann ein Schwärmer intensivere Nahbereichskommunikationssignale versenden.

Stigmergie t_p: Der Schwärmer kann RGB-Pheromone auslegen (inspiriert von z.B. Pheromonen im Ameisenstaat), welche dann für eine festgelegte Zeit in der Umwelt erhalten bleiben und von anderen Schwärmern wahrgenommen werden können. Bis ein Schwärmer ein neues Pheromon auslegen kann, vergeht eine gewisse Regenerationszeit t_p. Wird t_p auf kleine Werte gesetzt, so kann ein Schwärmer in kürzeren Abständen Pheromone auslegen.

Stoffaufnahme: Nahrung kann innerhalb eines kleinen Bereichs um die Schwärmerfront aufgenommen werden. Sie wird dann im körpereigenen Speicher gelagert und – wie bereits beschrieben – nach und nach verbraucht.

Nahrungsabgabe: Im Körper vorhandene Nahrung kann in kleinen Portionen an die Umgebung abgegeben werden, um z.B. andere Schwärmer zu füttern (inspiriert vom Sozialmagen vieler staatenbildender Insekten).

Vermehrung: Ein Schwärmer kann sich zum Zwecke der Vermehrung exakt klonen. Der Klon übernimmt die Parameter des sich vermehrenden Schwärmers, sowie eine Kopie von dessen Kontrollstruktur. Beim Klonvorgang wird das gespeicherte Futter des Schwärmers gedrittelt. Ein Drittel erhält der neue Schwärmer, ein Drittel behält der alte und das letzte Drittel wird durch die Vermehrung verbraucht.

4.2.3 Sensorik

Weiter kann ein Schwärmer auf verschiedene Weise Aspekte seiner Umwelt wahrnehmen (Sensorik). Die Sensorik eines Schwärmers wird durch folgende Parameter definiert:

Wahrnehmung der Fernkommunikation α (Sehsinn): Dieser Sinn markiert den „Sehsinn“ des Schwärmers, über welchen auch Objekte wie z.B. Futter ausgemacht werden können. Der Schwärmer soll zu extrem niedrig aufgelöstem RGB-“Sehen“ befähigt sein. Hierzu besitzt er einen kreissektorförmigen Sichtbereich mit festem Öffnungswinkel $\frac{\pi}{3}$ und einer Sichtweite α, der entlang seiner Winkelhalbierenden unterteilt wird in zwei Sichtzonen („Pixel“), welche die Sichtbereiche von einfachen Augen markieren sollen. Der Sichtbereich ist so ausgerichtet, dass der winkelhalbierende Strahl exakt der Orientierung des Schwärmers entspricht. Die Sichtzonen gehen fließend ineinander über, was der Kontrollstruktur die Informationsverarbeitung erleichtert. Damit kann ein Schwärmer die bereits beschriebenen RGB-Signale wahrnehmen, die von anderen Schwärmern abgegeben werden. Die Fernkommunikation soll auch gleichzeitig den „Sehsinn“ für die Wahrnehmung von z.B. Futter markieren. Der oben bereits erwähnte quadratische Abfall der Signalstärke mit steigender Entfernung des wahrgenommenen Objekts wird so berechnet, dass der verbleibende Anteil des Signals genau 0 beträgt, wenn der Abstand zum wahrgenommenen Objekt α beträgt. Wächst α, so hat der Schwärmer eine größere „Sichtweite“.

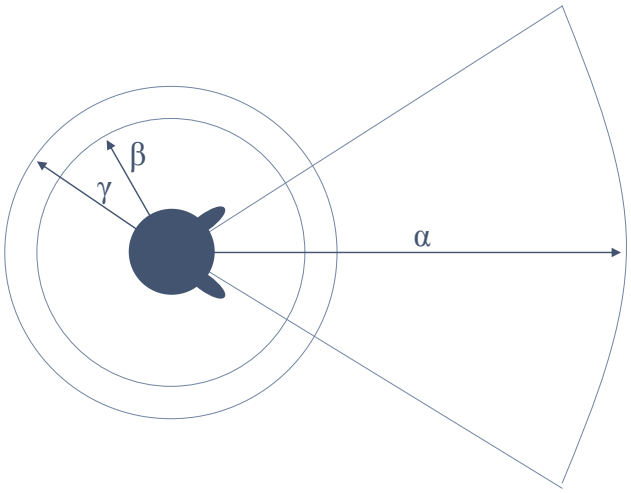


Abbildung 4.1: Schema des parametrisierbaren Schwärmers samt der Sensorikparameter α, β und γ. Die Fühler haben keine Funktion, sondern markieren nur auf intuitive Weise, in welche Richtung der Schwärmer orientiert ist. Dieses Schema ist nicht maßstabsgetreu.

Wahrnehmung der Nahbereichskommunikation β: RGB-Nahbereichskommunikation wird in einem Radius β wahrgenommen. Der entstehende kreisrunde sensorische Bereich wird in vier Wahrnehmungssektoren unterteilt, welche fließend ineinander übergehen. Der quadratische Abfall der Signalstärke wird analog zu dem der Fernkommunikation errechnet. Wächst β, so kann der Schwärmer die Nahbereichskommunikation auf größere Distanzen hin wahrnehmen.

Wahrnehmung der Anwesenheit anderer Schwärmer: Ein weiterer kreisrunder Wahrnehmungsbereich um den Schwärmer, ebenfalls mit dem Radius β, der auch in vier Sektoren mit fließendem Übergang geteilt ist, ermöglicht es Schwärmern, die Anwesenheit von anderen Schwärmern wahrzunehmen. Analog zur Wahrnehmung von anderen sensorischen Reizen soll auch hier die Intensität des Reizes mit der Entfernung des auslösenden Objektes quadratisch abnehmen.

Stigmergie-Wahrnehmung γ: Der Schwärmer soll in der Umgebung verteilte „RGB-Pheromone“ in einem Radius γ wahrnehmen können. Analog zur Wahrnehmung der Nahbereichskommunikation wird dieser sensorische Kreis um den Schwärmer in vier Sektoren aufgeteilt, um eine grob gerichtete Wahrnehmung der Pheromone zu erreichen. Diese Sektoren gehen – wie z.B. die des Sichtkegels – fließend ineinander über. Auch hier soll die Intensität der Wahrnehmung je nach Distanz des Pheromons quadratisch nachlassen, analog zu den bisher betrachteten quadratischen Intensitätsabfällen.

Interne Sensorik: Der Schwärmer soll den relativen Füllstand seines Futterspeichers wahrnehmen können.

Eine schematische Darstellung des parametrisierbaren Schwärmers samt der Skizzierung von Sensorikbereichen findet sich in Abb. 4.1.

4.2.4 Konkurrierende Fertigkeiten

Die oben skizzierten Parameter können keineswegs völlig unabhängig voneinander gesetzt werden: Es kann kein „Superschwärmer“ mit allen Parametern auf dem Bestwert parametrisiert werden. Das zugrundeliegende Prinzip ist, dass nur von endlich vielen Fertigkeitenressourcen ausgegangen wird, so wie in der Biologie kein Lebewesen in allen Disziplinen perfekt sein kann. Konkret

soll dieses Prinzip über eine feste Anzahl Fertigkeitenpunkte realisiert werden, welche auf verschiedene der oben vorgestellten Fertigkeiten verteilt werden können und so den Wert der oben vorgestellten Parameter definieren. Dies setzt voraus, dass für jeden der Parameter ein Minimal- und ein Maximalwert definiert wird, und die Anzahl der der jeweiligen Fertigkeit zugeteilten Fertigkeitenpunkte entscheidet, auf welchen Wert zwischen Minimum und Maximum der jeweilige Parameter gesetzt wird.

Die Menge der insgesamt zur Verfügung stehenden Fertigkeitenpunkte ist hierbei signifikant geringer, als nötig wäre, alle Fertigkeiten bei einem Schwärmer voll auszubauen. Wie genau die Verteilung von Fertigkeitenpunkten implementiert wird, wird im Rahmen der Implementierungsdokumentation des Schwärmers (Abschnitt 5.5) beschrieben. Die Parametrisierung der den Schwärmern eigenen Kontrollstruktur wird bewusst aus dieser Ressourcenverteilung ausgespart, da das Wachstum der Kontrollstruktur auf andere, in Abschnitt 5.7.2 beschriebene Weise beschränkt wird.

4.2.5 Kontrollstruktur

Als Kontrollstruktur dient ein rekurrentes neuronales Netz mit variabler Neuronenzahl. Die Zahl der Input- und Outputneurone ist durch die Sensorik und Aktorik vorgegeben. Die Grundlagen rekurrenter neuronaler Netze wurden in Abschnitt 3.1 erörtert. Die konkrete Implementierung und evolutionäre Entwicklung dieses Netzes wird im Abschnitt 5.3 beschrieben. Etwas später (Abschnitt 5.7.2) wird auch beschrieben, auf welche Weise die neuronalen Netze daran gehindert werden, ins Unermessliche zu wachsen.

4.3 Die Umwelt

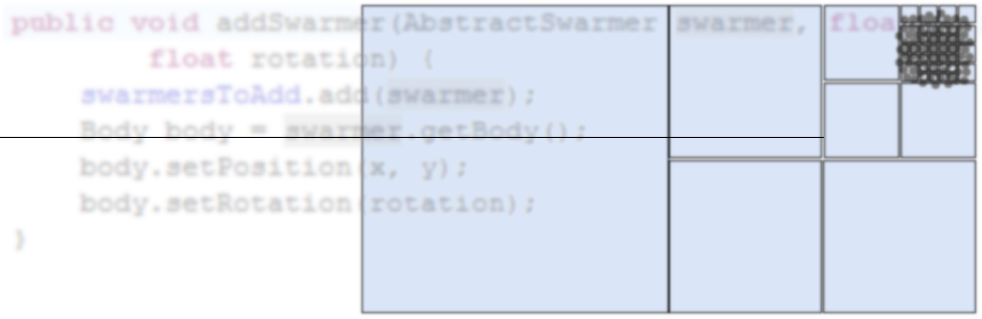
Die Umwelt der Schwärmer soll zweidimensional sein, räumlich quasi-analog und zeitdiskretisiert sein. Die Simulation soll natürlich motiviert sein und daher insbesondere auf einem Physikframework basieren. Dies ermöglicht effiziente Kollisionsdetektion von Objekten und realistische Kollisionsantworten. In der Umwelt können Hindernisse existieren. Weiterhin können Futterelemente, Pheromone und weitere Objekte in der Umwelt existieren. Sowohl Futter als auch Pheromone verschwinden nach einer gewissen Zeitspanne (motiviert durch natürliche Zersetzung). Die Umwelt soll dynamisch sein, so kann z.B. neues Futter erscheinen.

4.4 Prinzipien von Schwarmevolutionen

Sowohl die Verteilung der Fertigkeitenpunkte als auch die Kontrollstruktur des Schwärmers sollen durch einen evolutionären Algorithmus eingestellt werden. Das evolutionäre Ziel des Schwarms ist es hierbei lediglich, eine stabile Population seiner selbst zu bilden. Die Grundlagen evolutionärer Algorithmen wurden in Abschnitt 3.2 behandelt.

Wie oben beschrieben, wird ein Schwärmer durch unterschiedliche Parameter charakterisiert, welche ihrerseits von Fertigkeitenpunkten definiert werden. Diese Fertigkeitenpunkte und die Parameter seines neuronalen Netzes stellen das Genom eines einzelnen Schwärmers dar, welches als Teil eines Schwarm-Genoms evolutionär optimiert wird.

Ein Schwarm-Genom besteht aus einem Tripel von Schwärmer-Genomen. Jedes der drei Schwärmergenome repräsentiert eine Unterart Schwärmer, also einen Subschwarm. Auf das Schwarm-Genom wird der evolutionäre Algorithmus angewandt: Es werden also drei Subschwärme gleichzeitig durch den evolutionären Algorithmus optimiert. Die Evaluation selbst besteht dann aus einer Simulation, in der die Schwärmer in ihrer Umwelt leben, Futter zu sich nehmen, sich vermehren und sterben.



Kapitel 5

Implementierung der Softwareinfrastruktur

In diesem Kapitel wird die Implementierung der für die Evolution von Schwarmverhalten notwendigen Softwarebestandteile dokumentiert. Da das Volumen des speziell im Rahmen dieser Diplomarbeit geschriebenen Quellcodes umfangreich ist, wird diese Dokumentation im Wesentlichen auf semantischer Ebene stattfinden (Beschreibungen sollen also nicht bis auf die Ebene aller Implementierungsdetails und Spezialfälle führen, sondern nur die grundlegenden Mechanismen veranschaulichen), und auch nicht zwangsläufig den vollen Funktionsumfang der Softwarebestandteile umfassen.

Zunächst wird in Abschnitt 5.1 auf Grundprinzipien eingegangen, die der Diplomand während der Implementierung jeglicher Softwarebestandteile verfolgt hat. Abschnitt 5.2 beschreibt kurz die eingesetzte Evolutionssoftware sowie deren Erweiterung zu einer Client-Server-Applikation, mit dem Ziel, den entstehenden Rechenaufwand auf viele Rechner zu verteilen. In Abschnitt 5.3 wird die Implementierung eines strukturevolvierbaren neuronalen Netzes beschrieben. Abschnitt 5.4 beschreibt dann die Implementierung der Schwarmsimulationssoftware. Nachdem die bis jetzt genannten Abschnitte losgelöst voneinander betrachtet werden können, wird in Abschnitt 5.5 beschrieben, wie die konkrete Simulation, wie sie im vorherigen Kapitel modelliert wurde, auf Basis der implementierten Softwarebestandteile realisiert wird. Abschnitt 5.6 dokumentiert die Realisierung von automatisierten Analyseverfahren für bereits entstandenes Schwarmverhalten. Zuletzt widmet sich Abschnitt 5.7 dem grundlegenden technischen Aufbau und der Parametrisierung von Experimenten sowie der Hardwareinfrastruktur, auf deren Basis die Evolutionen durchgeführt werden.

5.1 Grundprinzipien während der Implementierung

Bevor auf die Implementierung der einzelnen Fragmente der Softwareinfrastruktur eingegangen wird, sollen zunächst einige Grundprinzipien und Ansprüche beschrieben werden, welche die Implementierungen geformt haben. Sämtliche Implementierungen wurden in der Programmiersprache JAVA durchgeführt. Während der Implementierungsphase wurde JAVA 1.6 genutzt, mit eventuellen Geschwindigkeitseinbußen sind die Frameworks jedoch auch auf JAVA 1.5 lauffähig. JAVA-Versionen vor 1.5 werden nicht unterstützt, da in dieser JAVA-Version eingeführte Neuerungen (z.B. Generics, [8]) genutzt werden. Als integrierte Entwicklungsumgebung wurde vorwiegend *Eclipse* [22] in der Version *Europa* benutzt.

5.1.1 Plattformunabhängigkeit

Sämtliche Softwarebestandteile sind so implementiert, dass sie auf jeder Plattform mit einer zertifizierten JAVA Virtual Machine lauffähig sind. Getestet wurden sie unter Windows 2000, Windows

XP, Windows 2003 Server (32 und 64Bit-Versionen) sowie verschiedenen Linux-Distributionen und OpenBSD.

5.1.2 Stil des Codes

Jeglicher Programmcode wurde so verfasst, dass er bereits aus sich heraus Informationen über das liefert, was er implementiert. Dies verlängert zwar Namen für beispielsweise Methoden und Objekte, beeinflusst dank eingesetzten Entwicklungswerkzeugen wie Eclipse jedoch die Geschwindigkeit des Implementierens nicht. Durch solch für sich selbst sprechenden Programmcode wird insbesondere eine Vielzahl von semantischen Fehlern von vornherein vermieden. Weiter ist der Code durchgehend mit Javadoc [46] kommentiert, so dass automatisch konsistente Code-Dokumentationen für spätere Weiterverwendung der Software generiert werden können. Der Code ist durchgehend in JAVA-Packages strukturiert, in denen funktional verwandte Programmteile übersichtlich zusammengefasst werden. Code und Kommentare sind durchgehend in englischer Sprache verfasst.

5.1.3 Universalität der Softwarefragmente

Sämtliche Softwarefragmente wurden mit der Absicht implementiert, sie nicht allein in dieser Diplomarbeit, sondern auch in anderen Projekten verwenden zu können – insbesondere in Forschung und Lehre. Die Softwarefragmente sind also universell gehalten und keineswegs auf die vorliegende Diplomarbeit spezialisiert. Hierdurch kann es vorkommen, dass Softwaremerkmale implementiert wurden, die nicht direkt im Rahmen dieser Diplomarbeit verwendet werden – in der Absicht, zum Ende der Diplomarbeit über in sich geschlossene, vielfältig und modular wiederverwendbare Software verfügen zu können.

5.1.4 Merkmale für effizientes Arbeiten

Die Softwarefragmente sind für zwei Aufgaben implementiert worden. Zum einen müssen sie ihren wissenschaftlichen Zweck erfüllen, zum anderen muss die wissenschaftliche Arbeit mit der Software ergonomisch sein, und zwar sowohl aus der Sicht eines Entwicklers, der aus den Fragmenten ein größeres Projekt zusammensetzt, als auch aus Sicht des Benutzers, der dieses Projekt dann anwendet.

Hilfestellungen für den Entwickler bestehen neben der Dokumentation und einem einheitlichen Code-Stil insbesondere aus einer Reihe von Werkzeugmethoden und -klassen. Diese stellen häufig wiederkehrende Unteraufgaben bei der Verwendung der Softwarefragmente direkt zur Verfügung. Dem Benutzer wird hingegen geholfen, indem auf Visualisierungsmerkmale und ergonomische Bedienung viel Wert gelegt wurde.

Zusätzlich wurden während der Diplomarbeit wiederkehrende Abläufe identifiziert, bei denen sich eine Automatisierung lohnt. Ein Beispiel für einen automatisierten Arbeitsschritt ist die Vorabtestsuite bereits evolvierter Schwärme, deren Implementierung in Abschnitt 5.6 dokumentiert wird.

Nachdem nun auf die grundsätzlichen Prinzipien der Implementierung eingegangen wurde, sollen im Folgenden die einzelnen implementierten Softwarebestandteile dokumentiert werden.

5.2 Erweiterung der Evolutionssoftware zur Client-Server-Applikation

5.2.1 Vorstellung der vorhandenen Software

Bei *Evolutionator* handelt es sich um eine Software, mit der man auf ergonomische Weise Lösungen zu Optimierungsproblemen mittels evolutionärer Strategien suchen kann. Die Software wurde vom Diplomanden im Rahmen eines Forschungsstipendiums sowohl konzipiert als auch implementiert.

Der Diplomand verkennt nicht, dass weitere Evolutionssoftwarepakete für JAVA existieren – genannt seien an dieser Stelle das *Java Evolutionary Computation Toolkit* [41] oder das *Java Genetic Algorithms Package* [45]. Beide besitzen einen großen Funktionsumfang und erfordern eine gewisse Einarbeitungszeit. Bedingt dadurch, dass die eingesetzte Software *Evolutionator* vom Diplomanden selbst stammt, entfällt hier die Einarbeitungsphase weitestgehend. Zudem konnte der Diplomand im Rahmen des Projektes *Beanbag Robotics* [35, 36] schon Erfahrungen im Bereich der Schwarmevolution mit der Software sammeln und so sicher sein, dass die Software bereits über die für die Diplomarbeit erforderlichen Merkmale verfügte oder diese einfach nachgerüstet werden konnten. Insofern fiel die Wahl auf *Evolutionator*.

Aus Entwicklerperspektive ist die Software *Evolutionator* insofern ergonomisch, als sie modular aufgebaut ist: Es können zum Beispiel auf einfache Weise zusätzliche Evolutionsstrategien oder Wahrscheinlichkeitsverteilungen für die Selektion hinzugefügt werden. Zudem muss ein beliebiges Optimierungsproblem, welches evolviert werden soll, nur ein einziges JAVA-Interface mit dem Namen *Evolvable* implementieren und kann danach sofort alle Merkmale der Software nutzen. Durch dieses Interface stellt das zu evolvierende Problem dem evolutionären Algorithmus unter anderem Mutations- und Kreuzungsoperatoren zur Verfügung.

Aus Benutzerperspektive ist, im Hinblick auf die Ergonomie, insbesondere die graphische Benutzeroberfläche zu nennen (Abb. 5.1 auf der folgenden Seite). Diese ermöglicht es nicht nur, die Entwicklung der Evolution zu verfolgen. Vielmehr können einzelne Lösungskandidaten oder ganze Populationen in Dateien gespeichert oder von diesen geladen, und zur Laufzeit alle relevanten Parameter der Evolution eingestellt werden. Weiter stellt die Benutzeroberfläche direkt mehrere Evolutions-Modi zur Auswahl: So kann zum Beispiel im Vorhinein eine festgelegte Anzahl von Evolutionen mit ebenfalls festgelegter Generationsanzahl gewählt werden, die dann sukzessive ausgeführt wird.

Die Berechnungen selbst werden beschleunigt, indem die Ausführungen von Mutationsoperator, Kreuzungsoperator und Evaluation auf eine ebenfalls zur Laufzeit wählbare Anzahl von Threads parallelisiert werden können. Dies ermöglicht die volle oder wohldosierte Auslastung auch von Mehr-CPU-Systemen, wie sie heute immer mehr Verbreitung finden. Da so die Parallelisierung direkt vom Evolutionsprogramm gesteuert wird, muss sie vom Benutzer bei der Implementierung seines speziellen, zu evolvierenden Problems nicht mehr berücksichtigt werden.

Im Rahmen der Implementierung der Parallelisierung auf mehrere Threads hat der Diplomand bereits an der Cornell University begonnen, eine weitergehende Parallelisierung auf mehrere Rechner über das Netzwerkprotokoll TCP zu ermöglichen. Das Evolutionsprogramm fungiert hierbei als Server. Andere Rechner können dann mit dem Server Verbindung aufnehmen, Rechenaufgaben anfordern (beispielsweise die Ausführung einer Fitnessfunktion oder Mutation), diese Rechenaufgaben lösen und das Ergebnis dem Server mitteilen. Dieses einfache Protokoll war jedoch rudimentärer Natur und besaß eine Menge Unzulänglichkeiten:

- Es konnte nur jeweils eine einzige Rechenaufgabe pro Zeit an einen Rechenclient verteilt werden.
- Eine weitere clientseitige Parallelisierungstufe, um mehrere Threads auf einem Client-Rechner auszuführen, existierte ebenfalls nicht. Sollten mehrere Threads auf einem Client-Rechner ausgeführt werden, so waren mehrere Instanzen der Client-Software zu starten.
- Die Codebestandteile des zu evolvierenden Problems mussten fest in die Clientsoftware einkompiliert werden.
- Die Clients mussten beim Start die Netzwerkadresse des Evolutionsservers vom Benutzer erhalten.
- Robustheit war nicht gegeben. Auf Fehler, wie beispielsweise ausfallende Clients, konnte nicht eingegangen werden, so dass in diesem Fall die Gesamtevolution stark beeinträchtigt wurde.

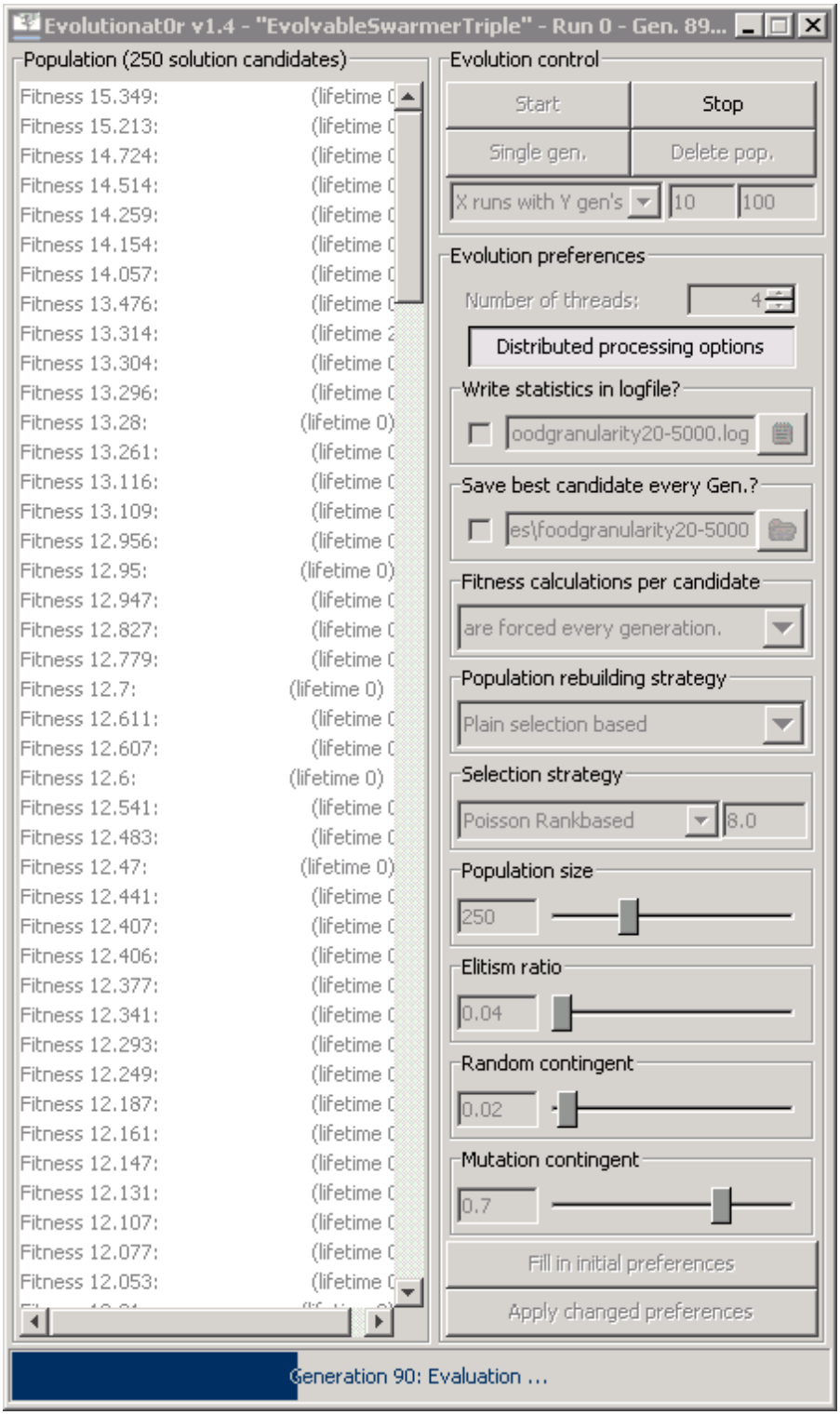


Abbildung 5.1: Abbild der graphischen Benutzeroberfläche der Evolutionssoftware. Auf der linken Seite des Fensters ist eine Liste der Lösungskandidaten der aktuellen Population zu sehen. Durch Rechtsklick lassen sich hier einer oder mehrere Kandidaten abspeichern, laden, re-evaluieren, visualisieren etc. Auf der rechten Seite finden sich die Bedienelemente zur Kontrolle der Evolution, unterteilt in einen oberen und einen unteren Block. Durch den oberen Block *Evolution control* legt man den Evolutionsmodus fest (Beispielsweise kann man vorab bestimmen, x Evolutionen hintereinander mit jeweils y Generationen durchzuführen) und startet und beendet die Evolution. Im unteren Block *Evolution preferences* können weitere Merkmale eingestellt und zur Laufzeit verändert werden: Beispielsweise, ob und wo Log-Dateien oder Problemlösungskandidaten automatisch gespeichert werden sollen, oder auch fundamentale Parameter wie Populationsgröße, Mutationsrate etc.

- Weitere Faktoren wie die Netzwerklatenz und -geschwindigkeit zu einzelnen Clients, Rechengeschwindigkeit eines jeden Clients und auch der erwartete Aufwand der Rechenaufgaben konnten den Fortgang der Evolution nachhaltig beeinflussen. Es konnte zum Beispiel passieren, dass ein besonders langsamer Client die letzte in einer Evolutions-Generation ausstehende Rechenaufgabe erhält und so schnellere Clients zum Warten zwingt.

Die folgend beschriebene Implementierung eines Kommunikationsprotokolls zur verteilten Evolution soll die oben genannten Unzulänglichkeiten vermeiden und adaptiv, effizient und robust Rechenaufgaben verteilen. Die Beschreibung des Protokolls ist in mehrere Schritte untergliedert, um für den Leser greifbarer zu machen, welches Protokollfragment welchem Zweck dient. Zunächst soll das grundlegende Prinzip des Datenaustauschs zwischen Client und Server beschrieben werden (Abschnitt 5.2.2). In Abschnitt 5.2.3 wird beschrieben, wie der Client flexibel auf verschiedene zu evolvierende Probleme angewandt werden kann ohne rekompiliert werden zu müssen, wie ihm neue Merkmale hinzugefügt werden können und wie steuerbar ist, wohin die Clients sich verbinden. Abschnitt 5.2.4 stellt dar, wie auf Fehler während der Evolution eingegangen wird. Schließlich folgt eine Beschreibung des Algorithmus, welcher bestimmt, wieviele Rechenaufgaben welchem Client auf Anfrage zugewiesen werden (Abschnitt 5.2.5).

5.2.2 Prinzip der Kommunikation zwischen Client und Server

Wichtigstes Grundprinzip ist, dass Rechenaufgaben nicht mehr einzeln versendet werden, sondern in *Paketen* variabler Größe. Die Grundstruktur der Kommunikation zwischen einem Rechenclient und dem Evolutionsserver läuft wie folgt ab:

Der Client meldet sich zunächst beim Server an (Details zur Anmeldeprozedur folgen im Abschnitt 5.2.3). Bis der Client beendet wird (z.B. durch einen Benutzer des PCs, auf dem der Client läuft, oder indem der Client sich selbst beendet), durchläuft er nun eine Schleife mit immergleichen Instruktionen.

Am Anfang jedes Schleifendurchlaufs bittet der Client den Server um ein Paket Rechenaufgaben. Erhält der Client nun Rechenaufgaben vom Server, so werden diese abgearbeitet und anschließend die Ergebnisse – wieder in einem Paket – an den Server zurückgeschickt. Aus verschiedenen Gründen kann der Client aber auch statt einem Paket Rechenaufgaben ein Wartesignal erhalten. In diesem Fall wartet der Client eine kurze Zeit ab und bittet dann erneut um ein Paket Rechenaufgaben.

Diese Wartezeit vergrößert sich um jeweils 500ms, wenn mehrere Wartesignale hintereinander auftreten, um einen Anfrageansturm auf den Server zu verhindern. Die maximale Wartezeit liegt bei 10 Sekunden, die initiale Wartezeit bei 500ms. Empfängt der Client nach einer Wartephase ein Paket Rechenaufgaben, so wird die Wartezeit wieder auf die initiale Wartezeit gesetzt.

Der Server hat die Aufgabe, eine Liste von vorhandenen Rechenaufgaben sowie verbundene Clients zu verwalten und auf Nachfrage Pakete von Rechenaufgaben – oder aber Wartesignale – an Clients zu verteilen. Auf welche Weise der Server die Anzahl der Rechenaufgaben in einem Paket bestimmt und welche Faktoren hierbei eine Rolle spielen, wird in Abschnitt 5.2.5 beschrieben.

5.2.3 Loginprozedur, Clientuniversalität und Protokoll-Updatefähigkeit

Im Rahmen der obigen Beschreibung der Unzulänglichkeiten der rudimentären Implementierung des Client-Server-Protokolls wurde bereits genannt, dass der Client insofern unflexibel war, als dass die Programmbestandteile des spezifischen zu evolvierenden Problems in den Client einkompiliert werden mussten. Zusätzlich musste die Adresse des Evolutionsservers beim Clientstart eingegeben werden, was bei vielen Clients nicht praktikabel ist – insbesondere, wenn diese nicht vor Ort sind, sondern sich automatisch über das Internet verbinden sollen. Weiterhin mussten die

Clients bei Änderung des Kommunikationsprotokolls komplett neu kompiliert und händisch neu gestartet werden.

Wie diese Unzulänglichkeiten behoben wurden, soll nun beschrieben werden – in Form der Prozedur, die ein Client zwischen seinem Programmstart und dem Zeitpunkt, an dem er zu einer Evolution verbunden ist, ausführt. Diese Prozedur unterteilt sich in mehrere Schritte:

Start des Clients. Das Clientprogramm wird gestartet. Dabei können via Kommandozeilenargumenten verschiedene Betriebsmodi des Clients gewählt werden, welche in Abschnitt 5.2.8 näher betrachtet werden.

Prüfung auf Clientaktualisierungen. Der Client lädt von einem festgelegten HTTP-Server[1] eine Textdatei herunter, welche die Versionsnummer des zur Zeit aktuellen Clients enthält. Unterscheidet sich diese Versionsnummer von derjenigen des lokal vorhandenen Clients, so wird (ebenfalls vom genannten Internetserver) eine JAR-Datei mit den aktuellen Clientprogrammbestandteilen nachgeladen und die vorhandenen Programmbestandteile mit dieser aktualisiert. Die neuen Programmbestandteile können beispielsweise eine erweiterte Version des Kommunikationsprotokolls beinhalten. Diese Vorgehensweise ermöglicht ein effizientes Update, da der Entwickler nichts weiter tun muss, als die Versionstextdatei und die neuen Clientprogrammbestandteile auf dem HTTP-Server zu deponieren. Sämtliche Clients bringen sich dann bei einem Neustart selbst auf den neuesten Stand, eine Rekompilierung jedes einzelnen Clients ist nicht erforderlich. Wann genau Clients neu gestartet werden, wird in Abschnitt 5.2.4 weiter betrachtet.

Suche des Evolutionsservers. Bis zum jetzigen Zeitpunkt ist dem Client die Adresse des Evolutionsservers unbekannt und muss in Erfahrung gebracht werden. Dies geschieht durch das Herunterladen und Auslesen einer weiteren Textdatei vom HTTP-Server, in der die Netzwerkadresse und der Verbindungsport enthalten sind. Diese Vorgehensweise ermöglicht ein effizientes „Umlenken“ der Clients durch simples Ändern der Textdatei auf einem HTTP-Server, falls sich die Adresse des Evolutionsservers ändert.

Verbindung zum Evolutionsserver. Der Client versucht, sich mit Hilfe der nun bekannten Netzwerkadresse des Evolutionsservers zu diesem zu verbinden. Dem Client sind hierbei im Vorhinein keine Informationen über das konkrete zu evolvierende Problem bekannt.

Herunterladen der problemspezifischen Programmbestandteile. Sobald der Client zum Evolutionsserver verbunden ist, sendet ihm der Evolutionsserver zunächst eine weitere JAR-Datei, welche die Programmbestandteile des zu evolvierenden Problems enthält und vom Client geladen wird. So wird der Client universell gehalten, seine Anpassung an das spezielle Problem erfolgt vollautomatisch. Rekompilierungen sind nun unnötig.

Teilnahme an der verteilten Evolution. Der Client ist nun bereit für die ersten Rechenaufgaben und fordert diese an.

Ein Client beendet sich und startet sich selbst automatisch neu, wenn die Verbindung zum Evolutionsserver beendet wird. Dies kann z.B. zum Ende einer Evolution, aber auch aus Verbindungsfehlern heraus passieren. Weiter kann der Server die Verbindung aktiv beenden, wenn sich Kommunikationsprotokoll von dem des Clients unterscheidet, der Client sich also unerwartet verhält. Der Server provoziert auf diese Weise einen Clientneustart und damit ein Update desselben.

Kann der Client sich nicht zu einer Evolution verbinden, beispielsweise weil die Textdatei mit der Adresse des Evolutionsservers nicht vorhanden oder die Evolution selbst nicht erreichbar ist, so versucht es der Client nach einer Wartezeit erneut ab dem Schritt „Suche des Evolutionsservers“. Diese Wartezeit ist initial 30 Sekunden lang und wird mit jedem weiteren fehlgeschlagenen Versuch in Folge um 30 Sekunden inkrementiert. Sie kann maximal 10 Minuten betragen. Um im Falle des gleichzeitigen Verbindungsabbruchs vieler Clients die Verbindungsanfragen der Clients über die Zeit zu streuen und so Lastspitzen am Server zu vermeiden, wird die Wartezeit jeweils mit einem zufällig gleichverteilt gewählten Faktor im Intervall $[0,9; 1,1]$ multipliziert.

[1] Hier: `http://evolution.dkriesel.com`

5.2.4 Robustheit der verteilten Berechnung

Weiter wurde angesprochen, dass das rudimentäre Client-Server-Protokoll nicht robust gegenüber Ausfällen von Clients war. Zu Ausfällen werden auch Verbindungsabbrüche gezählt. Bei einem Ausfall gingen die an einen Client vergebenen Rechenpakete verloren. Da aber beispielsweise am Ende einer Evolutionsgeneration alle vorhandenen Problemlösungskandidaten evaluiert sein müssen, hielt so die ganze Evolution an: Die Generation konnte nicht zuende geführt werden, also konnten keine neuen Rechenaufgaben generiert werden, so dass alle Clients nur noch Wartesignale erhielten. Bei stabilen Rechnerclustern in einem lokalen Netz kann ein solch rudimentäres Protokoll funktionieren – bei einem Zusammenschluss von Rechnern über instabile Medien wie das Internet ist es hingegen praktisch nicht einsetzbar. Zwei wesentliche Regeln im Client-Server-Protokoll machen die verteilte Berechnung robust:

Regelmäßige Verbindungsüberprüfung: Der Client schickt dem Server alle 10 Sekunden ein Kontrollsignal. Erhält der Server für 20 Sekunden kein solches Signal, so bricht er die Verbindung ab. Der Server stellt also den Ausfall eines Clients spätestens nach 20 Sekunden fest.

Rückgabe nicht abgearbeiteter Rechenaufgabenpakete: Der Server verwaltet für jeden Client eine Liste der Rechenaufgaben, die ein Client gerade bearbeitet. Bricht die Verbindung zu einem Client aus irgendeinem Grund ab, so werden dessen Rechenaufgaben der Liste der unbearbeiteten Rechenaufgaben wieder hinzugefügt.

Die Befolgung dieser beiden Regeln sorgt dafür, dass Ausfälle festgestellt werden können, und dass Rechenaufgaben, welche ein ausgefallener Client abarbeiten sollte, automatisch wieder an andere Clients verteilt werden.

Zur Robustheit der verteilten Berechnung gehört jedoch auch das Eingehen auf möglicherweise vorhandene, relativ langsame Rechenclients. Diese könnten andere, schnellere Clients z.B. am Ende einer Generation zum Warten zwingen. Dies hätte zur Folge, dass man viel Rechenzeit auf schnellen Rechnern verliert, weil wenige langsame Rechner ihre Rechenaufgaben noch nicht abgearbeitet haben. Wie auf solche Effizienzproblematiken eingegangen wird, ist Bestandteil des folgenden Abschnitts.

5.2.5 Verfahren der Zuweisung von Rechenpaketen zu Clients

Die Effizienz der Rechenaufgabenverteilung steht und fällt mit dem Verfahren, das die Rechenaufgaben zu Paketen geeigneter Größe zusammenfasst und an die Clients austeilt. Von welchen Faktoren die Paketgrößen abhängen, soll nun zunächst anhand eines umgangssprachlichen Regelwerks beschrieben werden. Anschließend erfolgt dann eine detailliertere Beschreibung der Prozedur zwischen dem Zeitpunkt, zu dem ein Client Aufgaben anfordert, und jenem, zu dem entweder ein Rechenaufgabenpaket oder ein Wartesignal zurückgeschickt wird.

Die groben, umgangssprachlichen Richtlinien für die Größenbestimmung lauten:

1. Je schneller ein Clientrechner, desto mehr Rechenaufgaben soll er abarbeiten.
2. Je mehr Clients, desto weniger Rechenaufgaben für jeden.
3. Sind noch viele Rechenaufgaben in der Warteschlange, so können tendenziell mehr Rechenaufgaben auf einmal verteilt werden, da nicht zu befürchten ist, dass hierdurch andere Clients keine Aufgaben mehr erhalten und warten müssen, was zu Verlust von Rechenzeit führen würde.
4. Sind hingegen nur wenige Aufgaben in der Warteschlange übrig, so müssen entsprechend weniger Rechenaufgaben auf einmal verteilt werden, um die Parallelisierung möglichst lange aufrecht zu erhalten.
5. Es darf nicht passieren, dass die Evolution auf einzelne Clients wartet.

6. Sämtliche Threads, die vom Client zur Verfügung gestellt werden, müssen möglichst gut ausgelastet werden.

Um möglichst allen diesen Kriterien gerecht zu werden, wurde ein Algorithmus implementiert, dessen einzelne Schritte im Folgenden beschrieben werden. Die Eingabe in den Algorithmus stellt die Anfrage eines Rechenclients nach einem Paket Rechenaufgaben dar, seine Ausgabe ist ein Paket Rechenaufgaben oder aber ein Wartesignal. Der Algorithmus unterteilt sich grob in drei Phasen. Zunächst ermittelt der Server, wie groß (gemessen in Clientrechenzeit) das zu erstellende Aufgabenpaket sein soll. Danach wird ein Rechenaufgabenpaket erstellt, welches ungefähr der ermittelten Größe entspricht. Schließlich wird das Paket noch unter Berücksichtigung einiger Spezialfälle und Feinheiten optimiert.

In jedem Schritt des nachfolgenden Algorithmus werden relevante Steuergrößen, von denen der Algorithmus abhängt, *hervorgehoben* und im Anschluß erklärt.

Ermittlung der gewünschten Clientrechenzeit. Es wird anhand der *Clientgeschwindigkeit* errechnet, welchen Anteil der Gesamtrechenleistung der anfragende Client ausmacht. Aus dieser Größe und dem *Gesamtrechenaufwand* der Aufgabenwarteschlange wird der Anteil des Clients am Gesamtrechenaufwand ermittelt. Es ist wichtig, dass Clients nicht sofort ein Paket Rechenaufgaben erhalten, dessen geschätzter Aufwand den Anteil des Clients am Gesamtrechenaufwand abdeckt. Dies garantiert, dass durch in der Warteschlange verbleibende Rechenaufgaben in gewissem Maße auf beispielsweise neu verbindende Clients eingegangen werden kann. Weiter kann flexibler auf Fehleinschätzungen von z.B. der Clientgeschwindigkeit reagiert werden. In der Praxis hat sich bewährt, die gewünschte Clientrechenzeit mit derjenigen Zeit zu initialisieren, die der Client schätzungsweise benötigt, um $\frac{2}{3}$ seines Anteils am Gesamtrechenaufwand abzuarbeiten. Diese kann abgeschätzt werden, da Schätzungen von Clientgeschwindigkeit, *Servergeschwindigkeit* und *geschätzter Zeitaufwand einer jeden Rechenaufgabe in der Warteschlange* bekannt sind. Die so initialisierte gewünschte Clientrechenzeit wird dann auf mehrere Arten modifiziert:

- Ist die gewünschte Clientrechenzeit größer als der im Algorithmus festgesetzte Maximalwert von 10 Minuten, so wird die gewünschte Clientrechenzeit auf 10 Minuten gesetzt. Dies soll verlorene Rechenzeit im Falle eines Clientausfalls im Rahmen halten.
- Ist der Client weniger als 10 Minuten verbunden, so wird seine gewünschte Rechenzeit halbiert. Dies soll insbesondere während der Anfangsphase der verteilten Evolution, in der sich viele Clients gleichzeitig einloggen, verhindern, dass für manche Clients keine Aufgaben mehr übrig sind.
- Ist die gewünschte Clientrechenzeit kleiner als der im Algorithmus festgesetzte Wert von einer Sekunde, so wird die gewünschte Clientrechenzeit auf eine Sekunde gesetzt. Dies soll verhindern, dass die Verteilungsgranularität im Falle sehr vieler kleiner Rechenaufgaben beliebig fein wird, was ein hohes Kommunikationsaufkommen erzeugen würde.

Fertigstellung des vorläufigen Aufgabenpaketes. Solange in der Warteschlange Rechenaufgaben vorhanden sind und die geschätzte Paketrechendauer auf dem Client nicht die gewünschte Clientrechenzeit überschreitet, werden dem vorläufigen Aufgabenpaket Rechenaufgaben aus der Warteschlange hinzugefügt.

Optimierung des vorläufigen Aufgabenpaketes. Es können nun einige Spezialfälle auftreten, die in der Optimierungsphase behandelt werden:

- Die Anzahl der Aufgaben im Paket kann geringer sein als die Anzahl der auf dem Client laufenden Rechenthreads. In diesem Fall wird das Paket solange weiter aufgefüllt, wie die Warteschlange nicht leer ist, die Anzahl der Rechenthreads nicht erreicht ist und der Client nicht seinen Anteil an der Gesamtrechenzeit überschreitet.
- Die Anzahl der Aufgaben im Paket kann nach wie vor geringer sein als die *Anzahl der auf dem Client laufenden Rechenthreads*, beispielsweise wenn keine Aufgaben mehr in

der Warteschlange waren oder der Anteil der Gesamtrechenzeit des Clients überschritten wurde. In diesem Fall erfolgt eine Besonderheit: Die restlichen Threads des Clients werden genutzt, indem das Paket mit zufällig ausgewählten Aufgaben aufgefüllt wird, welche bereits versendet wurden, aber deren Ergebnis noch nicht vorliegt. Als Ergebnis einer auf diese Weise mehrfach zugewiesenen Rechenaufgabe wird dann das zuerst eintreffende verwendet; alle eventuellen weiteren werden verworfen. So kann sich die Evolution vor einzelnen, sehr langsamen Clients schützen, da deren Aufgaben, falls sie zu viel Zeit benötigen, automatisch an andere Clients wiederverteilt werden. Ein Client kann sich also nur selbst blockieren, nicht aber die gesamte Evolution.

Rückmeldung an den Client: Sind keine Aufgaben in der Warteschlange und auch keine mehr ausstehend, so ist das Paket leer und dem Client wird ein Wartesignal gesendet. Anderenfalls wird dem Client das im Laufe des beschriebenen Verfahrens erstellte Aufgabenpaket gesendet.

Während der Beschreibung des Verteilungsverfahrens wurden verschiedene Steuergrößen im Text *hervorgehoben*, deren Herkunft kurz dargelegt werden soll.

Clientgeschwindigkeit und *Servergeschwindigkeit* können abgeschätzt werden, da ein fest definierter Benchmark regelmäßig auf dem Client sowie einmalig auf dem Server ausgeführt wird. Benchmarkergebnisse von Clients werden dann an den Server geschickt. Die erste Übertragung des Benchmarkwertes findet bereits beim Einloggen eines Clients statt.

Der *Gesamtrechenaufwand* der Warteschlange kann abgeschätzt werden, da die Serverrechenzeit einer jeden Rechenaufgabe abgeschätzt werden kann.

Der *Aufwand einer einzelnen Aufgabe* kann abgeschätzt werden, indem beim Übertragen von Rechenergebnissen vom Client an den Server die Rechenzeiten, welche für jede Aufgabe benötigt wurden, mit übertragen werden. Dadurch, dass Benchmarkwerte sowohl von Server als auch Client vorliegen, können diese übertragenen Rechenzeiten allesamt auf Serverrechenzeiten umgerechnet werden, was über viele Clients hinweg einen verlässlichen Schätzwert für die Serverrechenzeit einer einzelnen Aufgabe ergibt. Dieser Schätzwert wird für jede Art von Aufgabe separat gebildet: Fitnessauswertung, Mutation und Kreuzung. Da sich diese Werte über die Evolution hinweg stark verändern können, werden sie in jeden Generation neu errechnet.

Die *Anzahl der auf dem Client laufenden Rechenthreads* wird ebenfalls vom Client an den Server übertragen: Einmal beim einloggen, und anschließend jedes mal, wenn sich die Anzahl der Threads z.B. auf Benutzerwunsch ändert.

Aus der Dynamik der Steuergrößen des Algorithmus resultiert dann eine effiziente, adaptive Verteilung von Rechenpaketen auch an viele, heterogene Clients.

5.2.6 Kodierung der Rechenaufgaben und Komprimierung

Im Rahmen der Beschreibung der Software *Evolutionator* wurde dargestellt, dass zur Anwendung der Software auf ein evolvierbares Problem ein Interface *Evolvable* implementiert werden muss. Über dieses Interface erhält die Evolutionssoftware Repräsentationen einzelner Lösungskandidaten für das zu optimierende Problem als Zeichenkette (String). Solche Zeichenketten können nicht nur erzeugt, sondern auch wieder ausgelesen werden, was beispielsweise für das Abspeichern von Lösungskandidaten in Dateien genutzt wird, aber auch für das Senden von Lösungskandidaten über das Netzwerk.

Für Rechenaufgaben, welche Evaluationen repräsentieren, wird dann die Zeichenkettenrepräsentation eines Individuums zum Client gesendet, dort geparst, die Evaluation ausgeführt und das Ergebnis zurückgesendet. Im Falle einer Rechenaufgabe, welche eine Kreuzung repräsentiert, werden zwei Repräsentationen an den Client übertragen, beide geparst, auf beide Problemlösungskandidaten der Kreuzungsoperator ausgeführt und die Zeichenkettenrepräsentation des resultierenden Individuums zurückgesendet. Mutations-Rechenaufgaben werden analog realisiert.

Da Zeichenkettenrepräsentationen meist schematisch aufgebaut und daher oft hochrepetitiv sind, bietet sich an, Komprimierungsverfahren zu nutzen. Das Client-Server-Protokoll komprimiert also Aufgabenpakete sowie Ergebnispakete mittels des Verfahrens *GZip* [15], bevor sie gesendet werden. Der jeweilige Empfänger dekomprimiert sie dann. Da dieser Vorgang direkt im Protokoll stattfindet, ist er für die Evolution selbst völlig transparent. Je nach Art der Zeichenkettenrepräsentationen lassen sich so große Mengen an Datenverkehr einsparen, was insbesondere bei der Evolution über das Internet die Übertragungszeiten reduziert.

5.2.7 Start des Servers über eine graphische Benutzeroberfläche

Um den Evolutionsserver zu starten, steht eine graphische Benutzeroberfläche zur Verfügung (Abb. 5.2 auf der rechten Seite). Mittels dieser kann frei gewählt werden, welche Art von Rechenaufgaben (Mutation, Kreuzung, Fitnessauswertung) überhaupt über verschiedene Rechner parallelisiert werden soll. Liegt der Hauptrechenaufwand beispielsweise allein in den Fitnessauswertungen, so kann es effizient sein, nur diese auf verschiedene Rechner zu verteilen, Mutations- und Kreuzungsoperatoren aber direkt auf dem Server-Computer auszuführen.

5.2.8 Verschiedene Betriebsmodi des Clients

Zuletzt sollen die verschiedenen Betriebsmodi des Clientprogramms kurz beschrieben werden. Je nach persönlicher Vorliebe kann der Benutzer das Clientprogramm mit oder ohne graphischer Benutzeroberfläche (Abb. 5.3 auf Seite 44) starten. Die graphische Benutzeroberfläche hat den Vorteil, dass die Anzahl der Rechenthreads zur Laufzeit geändert werden kann, wohingegen sie beim Start im reinen Konsolenmodus initial fest eingegeben werden muss.

Als weiteres wichtiges Merkmal existiert ein Hintergrunddienstmodus, der sich insbesondere für Büro-PCs eignet: In diesem Modus läuft der Client völlig unsichtbar und verwendet tagsüber (von 07:00h bis 21:00h) nur $(n-1)$ Rechenthreads, wobei n die Anzahl der auf dem Clientcomputer vorhandenen CPUs ist. In der restlichen Zeit wird dann auf n Rechenthreads hochgeschaltet[2]. In diesem Modus startet der Client automatisch und auch ohne dass ein Benutzer am System angemeldet ist. So kann Rechenzeit genutzt werden, die normalerweise brach liegt.

5.3 Implementierung des strukturevolvierbaren neuronalen Netzes

Wie bereits beschrieben, wird als interne Kontrollstruktur der Schwärmer ein neuronales Netz eingesetzt. Dieses soll im Laufe einer Evolution nicht nur fähig sein, seine synaptischen Gewichte zu verändern, vielmehr soll es auch seine Struktur verändern können. Damit die Freiheit der Evolution nicht eingeschränkt wird, ist der Anspruch an die hier beschriebene Implementierung des neuronalen Netzes, beliebige Netztopologien repräsentieren zu können. Dies schließt insbesondere rekurrente Topologien mit ein.

Es war nicht das Ziel der Implementierung des strukturevolvierbaren neuronalen Netzes, ein vollständig objektorientiertes, komplett graphisch bedienbares Framework zu schaffen, wie z.B. JOONE [44] eines ist. Ziel war vielmehr eine overheadfreie, schlanke, abstrakte, für Entwickler gedachte Implementierung eines neuronalen Netzes beliebiger Topologie, das aus Geschwindigkeitsgründen nicht von jedem durch die Objektorientierung möglichen Mechanismus Gebrauch macht. Im Folgenden soll beschrieben werden, welche Möglichkeiten die im Rahmen dieser Arbeit angefertigte Implementierung eines neuronalen Netzes bietet und wie diese technisch realisiert sind.

[2] Der Unterschied zwischen n und $(n-1)$ mag sich zunächst klein anhören. Da allerdings ein beträchtlicher Anteil der verfügbaren Rechner genau 2 CPUs besitzt, lag der Zuwachs an Rechengeschwindigkeit durch die volle Ausnutzung aller CPUs oft bei ca. 70%.

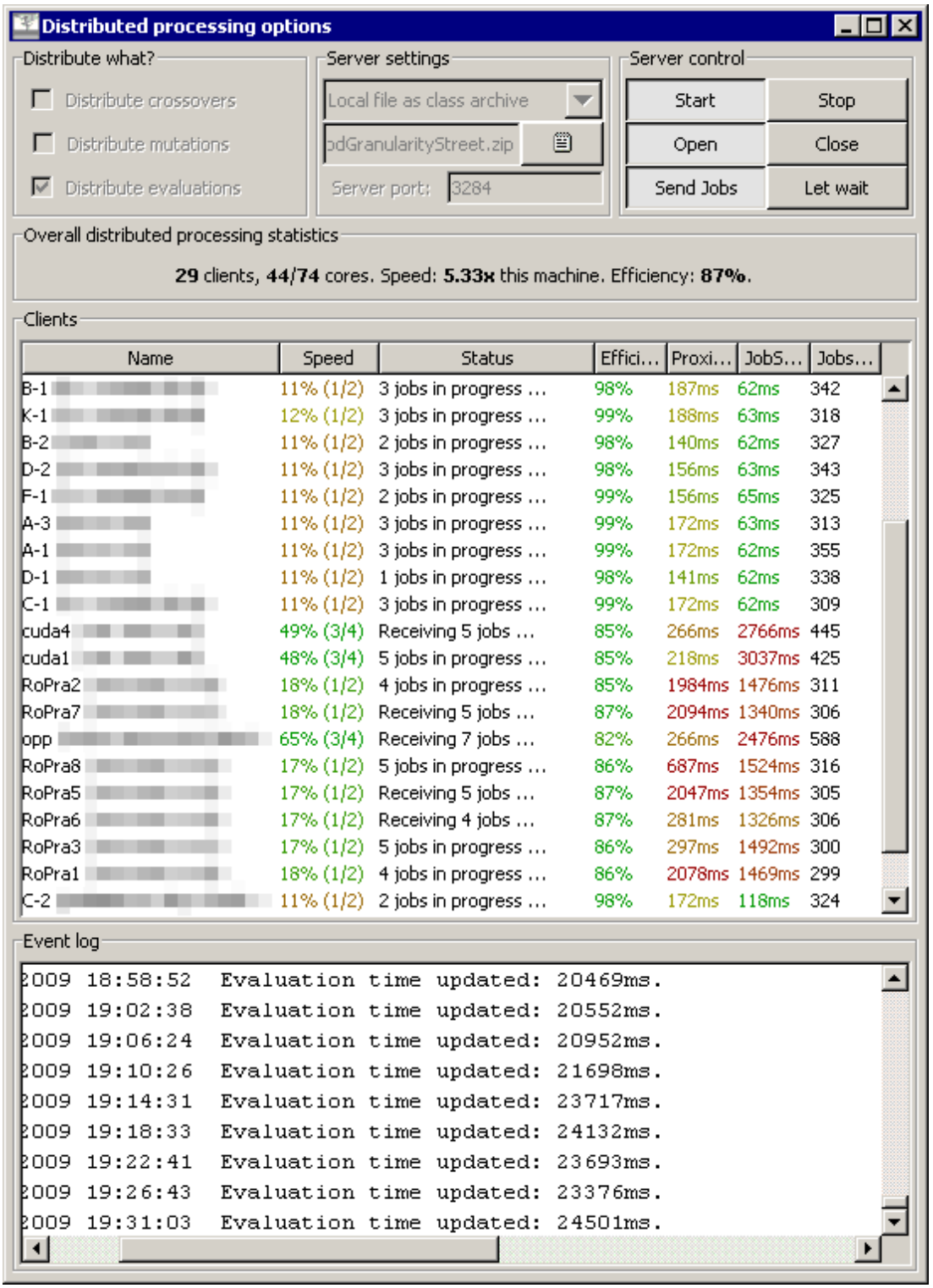


Abbildung 5.2: Abbild der graphischen Benutzeroberfläche des Evolutionsservers. Der Bereich *Clients* zeigt verschiedene Daten zu den aktuell verbundenen Clientcomputern an. Verbindungsdetails wurden unkenntlich gemacht. Über dem Bereich *Clients* sind verschiedene Statistiken und Kontrollelemente zu sehen, mit welchen sich der Server steuern lässt. Under dem *Clients*-Bereich findet sich ein Protokoll, welches verschiedene Informationen in chronologischer Reihenfolge präsentiert, wie z.B. das An- und Abmelden verschiedener Clients. Die gemessene Evolutionsgeschwindigkeit und Effizienz berücksichtigt Wartezeiten, Datenübertragungszeiten und Geschwindigkeiten einzelner Rechner. Die aktuelle Geschwindigkeit wird relativ zum Server („this machine“) dargestellt, in diesem Fall ein Dell Poweredge 1900 Unternehmensserver mit 8 Intel Xeon-CPUs.

Abbildung 5.3: Abbild der graphischen Benutzeroberfläche des Evolutionsclients. Im linken Bereich kann der Benutzer zur Laufzeit die Anzahl der CPUs wählen, die er der Evolution zum Rechnen zur Verfügung stellen will. Im rechten Bereich wird dargestellt, wie der Fortschritt des aktuell zugewiesenen Paketes Rechenaufgaben vorangeht.

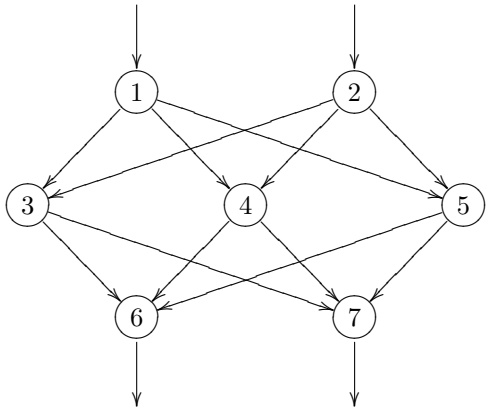


Abbildung 5.4: Partitionierung der Neuronenmenge eines beispielhaften Netzes in Schichten. Die Neuronen sind mit ihren Indizes beschriftet. Eingabeneurone sind 1 und 2, Ausgabeneurone 6 und 7. Die Neurone $3, 4$ und 5 sind der versteckten Schicht zugeordnet. Der Index 0 ist reserviert für das hier nicht abgebildete Biasneuron.

5.3.1 Interne Datenstrukturen

Um den prinzipiellen Aufbau des neuronalen Netzes verständlich zu machen, zunächst eine formale Betrachtung: Sei – zunächst losgelöst von den synaptischen Verbindungen – ein neuronales Netz mit N vielen Neuronen in M vielen Neuronenschichten betrachtet. Die Menge der Neuronen-Indizes ist dann $\{0, 1, \ldots, N\}$, wobei der Index 0 für das Bias-Neuron steht, und die Indizes $1, \ldots, N$ für die N vielen Neurone. Die Menge der Schicht-Indizes ist analog definiert zu $\{0, 1, \ldots, M-1\}$. Weder für M noch für N existieren in der Implementierung des neuronalen Netzes Beschränkungen. Grenzen sind einzig und allein durch den bereitgestellten Hauptspeicher gegeben. Sei $s(n) : \{0, 1, \ldots, N\} \mapsto \{0, 1, \ldots, M-1\}$ diejenige Abbildung, welche einen Neuronenindex $n \in \{0, 1, \ldots, N\}$ in den Index derjenigen Schicht überführt, zu der das indizierte Neuron gehört. Seien weiterhin $n_1, n_2, n_3 \in \{0, 1, \ldots, N\}$. Dann gilt:

1. $s(n_1) > s(n_2) \;\Rightarrow\; n_1 > n_2$,
2. $s(n_1) < s(n_2) \;\Rightarrow\; n_1 < n_2$,
3. $s(n_1) = s(n_3) \wedge n_1 < n_2 < n_3 \;\Rightarrow\; s(n_1) = s(n_2) = s(n_3)$.

Die Menge der Neuronenindizes wird also durch die Bildung von Schichten partitioniert (Abb. 5.4). Hierbei stellt die vorderste Schicht die Eingabeschicht dar, und die hinterste Schicht die Ausgabeschicht. Dazwischen liegen in aufsteigender Reihenfolge die inneren Schichten.

Nach der Betrachtung der Organisation der Neurone sei nun die Organisation der synaptischen Verbindungen betrachtet. Da beliebige Netztopologien ermöglicht werden sollen, können prinzipiell von jedem Neuron zu jedem anderen Neuron Verbindungen vorkommen (mit Ausnahme von Verbindungen in Richtung Eingabeneurone und in Richtung Bias-Neuron). Die Zahl der Verbindungen wächst also potentiell quadratisch mit der Anzahl der Neurone. So sind die synaptischen Verbindungen durch zwei quadratische Matrizen kodiert:

1. Eine Matrix beinhaltet die Gewichtswerte in doppelter Gleitkomma-Genauigkeit,
2. die andere Matrix beinhaltet Boolsche Variablen, welche definieren, ob die betrachtete Synapse *angeschaltet* ist oder nicht (eine nähere Betrachtung der Existenz von Synapsen findet sich in Abschnitt 5.3.2).

Hierbei wird bewusst in Kauf genommen, dass diese Kodierung für sehr große und gleichzeitig dünn verbundene Netze speicherineffizient ist. Datenstrukturen, welche – unter Aufrechterhaltung der Möglichkeit beliebig gearteter Verbindungen – speichereffizienter sind (wie beispielsweise Adjazenzlisten), sind aber in der Regel weniger effizient beim wahlfreien Zugriff auf einzelne Gewichte, was die Berechnungsgeschwindigkeit des neuronalen Netzes herabsetzen würde. Eine Implementierung, welche bis auf Neuronenebene objektorientiert modelliert ist, erschien dem Diplomanden ebenfalls aus Geschwindigkeitsgründen nicht sinnvoll.

5.3.2 Klassen und Existenz synaptischer Verbindungen

Wie schon in Abschnitt 3.1 beschrieben, kann man synaptische Verbindungen von einem Neuron n_1 in Richtung des Neurons n_2 anhand ihrer Start- und Zielschicht in verschiedene Klassen unterteilen:

Vorwärtsverbindungen: Man spricht von einer Vorwärtsverbindung, wenn n_2 eine Schicht weiter in Richtung Ausgabeschicht liegt als n_1, also gilt

$$s(n_2) = s(n_1) + 1.$$

Vorwärtsabkürzungen: Dies sind Verbindungen, welche noch einen größeren Schritt in Richtung Ausgabeschicht tun, also Neuronenschichten überspringen: Es gilt

$$s(n_2) > s(n_1) + 1.$$

Rückwärtsverbindungen: Führen nicht in Richtung der Ausgabeschicht, sondern in Richtung Eingabeschicht, wobei keine Schichten übersprungen werden. Es gilt

$$s(n_2) = s(n_1) - 1.$$

Rückwärtsabkürzungen: Analog zu Vorwärtsabkürzungen, nur in rückwärtiger Richtung. Führen in Richtung Eingabeschicht, wobei Neuronenschichten übersprungen werden. Es gilt

$$s(n_2) < s(n_1) - 1.$$

Laterale Verbindungen sind schichtinterne Verbindungen. Sie führen von einem Neuron zu einem anderen, welches in derselben Neuronenschicht liegt: Es gilt

$$s(n_2) = s(n_1).$$

Selbstverbindungen: Führen von einem Neuron zu wieder demselben Neuron, sind also ein Spezialfall der lateralen Verbindungen.

Existieren Rückwärtsverbindungen, -abkürzungen, laterale oder Selbstverbindungen, so ist das neuronale Netz *rekurrent*. Jede dieser Klassen von Synapsen kann einzeln als erlaubt oder unerlaubt definiert werden. Jede einzelne Synapse ist also nicht nur wie bereits oben beschrieben *angeschaltet* oder *abgeschaltet*, sie ist auch, je nachdem zu welcher Synapsenklasse sie gehört, entweder *erlaubt* oder *unerlaubt*.

Synaptische Verbindungen werden in der Verwendung des neuronalen Netzes nur dann berücksichtigt, wenn sie *existieren*. Eine Synapse existiert genau dann, wenn folgende zwei Kriterien zutreffen:

1. Die Synapse ist angeschaltet.
2. Die Synapse ist erlaubt.

Verbindungen, welche ein Neuron der Eingabeschicht oder das Biasneuron zum Ziel haben, sind niemals erlaubt und damit auch niemals existent.

Nachdem nun die Verwaltung von Neuronen und Verbindungen, also der gesamten Netzstruktur betrachtet wurde, sollen Operatoren auf der Netzstruktur definiert werden.

5.3.3 Strukturoperatoren

Die Struktur eines neuronalen Netzes muss – um dynamisch verändert werden zu können – über definierte Operatoren verfügen.

Hinzufügen von synaptischen Verbindungen: Schaltet eine synaptische Verbindung ein, so dass sie fortan in die Berechnung mit einbezogen wird, falls sie erlaubt ist. Das Gewicht einer neuen synaptischen Verbindung wird mit einem gleichverteilt zufällig gewählten Wert $x \in [-0.5; 0.5]$ belegt.

Entfernen von synaptischen Verbindungen: Schaltet eine synaptische Verbindung ab, so dass sie fortan von der Berechnung ausgenommen ist.

Hinzufügen eines Neurons in eine Neuronenschicht: Fügt ein Neuron in eine bestimmte Neuronenschicht m hinzu. Dies hat zur Folge, dass die Neuronenindizes aller Neurone, welche in Schichten $m+1$ und größer liegen, um eins inkrementiert werden. Den entstehenden freien Index erhält das neue Neuron. Die Matrizen mit Gewichtswerten und Schaltzuständen der Synapsen werden entsprechend neu geschrieben und dabei um eine Zeile und eine Spalte erweitert. Neu entstandene mögliche Verbindungen sind zunächst deaktiviert und können später mittels des Operators „Hinzufügen von synaptischen Verbindungen“ hinzugefügt werden.

Entfernen eines Neurons: Entfernt ein Neuron, sofern es nicht das letzte verbleibende Neuron seiner Schicht ist. Zu diesem Zweck werden wieder die Matrizen mit Gewichtswerten und Schaltzuständen der Synapsen erneuert, enthalten allerdings nicht mehr die zu dem Neuron gehörende Spalte bzw. Zeile. Da die Anzahl der Schichten initial fest vorgegeben wird, kann ein Neuron nicht entfernt werden, wenn es das einzige seiner Schicht ist.

5.3.4 Weitere Operatoren

Es sind weitere Operatoren in der Implementierung des neuronalen Netzes enthalten, welche nicht direkt die Struktur betreffen:

Festlegen der Aktivierungsfunktion pro Schicht: Für jede Schicht Neuronen kann die neuronale Aktivierungsfunktion separat festgelegt werden. Implementiert sind bis jetzt die Aktivierungsfunktionen Tangens hyperbolicus, Fermi-Funktion und Identität. Um Modularität und Erweiterbarkeit zu gewährleisten, müssen Aktivierungsfunktionen nur ein JAVA-Interface implementieren und können dann direkt im Netz verwendet werden. So kann man sehr schnell weitere Aktivierungsfunktionen definieren und verwenden, ohne mit dem Quellcode des neuronalen Netzes an sich in Berührung zu kommen.

Training des Netzes mit Backpropagation: Unter Angabe von Trainingsbeispielen, Lernrate und der Anzahl der durchzuführenden Lernschritte wird ein Online-Training mit Backpropagation durchgeführt. In jedem einzelnen Lernschritt wird ein Trainingsbeispiel zufällig ausgewählt und mit Backpropagation trainiert. Dies funktioniert selbstverständlich nur bei nicht-rekurrenten Netzen.

5.3.5 Export und Import

In der Implementierung des neuronalen Netzes sind Import- und Exportfunktionen inbegriffen. Mittels dieser lässt sich ein neuronales Netz in eine Zeichenkette exportieren, die beispielsweise in eine Datei gespeichert werden kann. Umgekehrt lässt sich eine solche Zeichenkette auch wieder in ein neuronales Netz importieren. Da hier nur die existenten synaptischen Verbindungen als Adjazenzliste exportiert werden, steigt der Speicherbedarf nicht quadratisch mit der Anzahl der Neuronen, sondern linear mit der Anzahl der existenten Synapsen. Zusätzlich enthält die exportierte Zeichenkette noch einen konstant großen Satz an Informationen, welcher die Topologie und die Aktivierungsfunktionen definiert.

5.3.6 Geschwindigkeitsoptimierung durch Konnektivitäts-Zwischenspeicher

Propagiert man Daten durch ein existierendes Netz, so muss beim Berechnen der Netzeingabe eines Neurons n aufgrund der beliebigen Netztopologien damit gerechnet werden, dass von jedem anderen Neuron eine synaptische Verbindung zu n existiert. Der Rechenaufwand würde bei dieser naiven Propagierung quadratisch mit der Anzahl der Neurone steigen. Darum hält die hier vorgestellte Implementierung eines neuronalen Netzes Konnektivitäts-Zwischenspeicher vor. Für jeden Neuronenindex n wird hierbei gespeichert:

- Ein Array mit den Indizes der Neurone, von denen eine synaptische Verbindung mit dem Ziel n ausgeht, sowie
- ein Array mit den Indizes der Neurone, zu denen von n aus eine Verbindung ausgeht.

Dieser Zwischenspeicher wird automatisch bei der ersten Propagierung (z.B. durch Eingabe von Werten in das Netz oder einen Backpropagation-Schritt) durch das Netz aufgebaut und dann so lange vorgehalten, wie sich nichts an der Konnektivität des Netzes ändert. Bei einer Änderung verliert der Zwischenspeicher seine Gültigkeit und wird dann bei der nächsten Propagierung durch das Netz wieder neu aufgebaut.

So muss nicht bei jeder Propagierung für jedes Neuron jedes andere Neuron auf existierende Verbindungen geprüft werden, sondern es kann der Zwischenspeicher verwendet werden, um direkt die Aktivierungen der relevanten Neurone abzurufen. Der Propagierungsaufwand steigt also nicht mehr quadratisch mit der Anzahl der Neurone, sondern linear mit der Anzahl der synaptischen Verbindungen.

5.3.7 Evolutionsoperatoren

Damit das neuronale Netz evolutionär optimiert werden kann, ist es nötig, Operatoren für die Mutation eines neuronalen Netzes und Kreuzung zweier neuronaler Netze zu definieren. Zur Evolution neuronaler Netze gibt es mittlerweile eine Vielzahl veröffentlichter Operatorenpakete, Auswertungen und theoretischer Arbeiten, z.B. [7, 25, 67, 69–71], aus denen die hier implementierten Operatoren entnommen sind. Keiner der im folgenden beschriebenen Operatoren verändert die Anzahl der Eingabe- und Ausgabeneurone. Die Anzahl der Neuronenschichten wird ebenfalls nicht verändert[3]. Die Anzahl von Eingabeneuronen, Ausgabeneuronen und Neuronenschichten wird bei Anwendung der Operatoren über eine Population neuronaler Netze als gleich vorausgesetzt.

[3] Da im Folgenden neuronale Netze mit beliebig gearteten Verbindungen verwendet werden, lassen sich mit einer einzigen verdeckten Neuronenschicht ohnehin beliebig viele emulieren.

Mutation

Mutationsoperatoren haben die Aufgabe, vorhandene Lösungskandidaten zufällig geringfügig zu verändern. An dieser Stelle sollen verschiedene implementierte Operatoren vorgestellt und anschließend dokumentiert werden, wie und in welchem Umfang sie im Rahmen dieser Arbeit genutzt werden. Es sei zunächst betrachtet, wie die synaptischen Gewichte mutiert werden. Hierfür sind zwei Operatoren implementiert:

Exponentialmutation: Ein Ansatz, der sich im Rahmen des Projektes *Beanbag Robotics* [35,36] und anderen Projekten bewährt hat, ist der folgende: Man wähle zufällig gleichverteilt ein $x \in [-4; 1]$. Dann subtrahiere oder addiere (zufällig gleichverteilt gewählt) man den Wert $d = \mathrm{e}^x$ von bzw. zu einem beliebigen existenten Gewicht. Offensichtlich bewegt man sich hier – wenn die Gewichte den Parameterraum repräsentieren – ausschließlich entlang der Achsen dessen Koordinatensystems, und zwar mit der Schrittweite d.

Gauß-Mutation: Ein verbreiteter Ansatz, welcher sich nicht nur entlang der Achsen des Koordinatensystems des Parameterraums bewegt, ist beispielsweise zu finden in [19, 70]. Auch hier wird analog zu der Exponentialmutation eine Schrittweite d ermittelt. Hierbei werden sämtliche Gewichte als Vektor angesehen. Auf jede Komponente dieses Vektors wird dann eine gaußverteilt (um 0, mit Standardabweichung d) ermittelte Zufallsvariable addiert.

Als Mutationsoperatoren, welche die Struktur des neuronalen Netzes verändern, stehen zur Verfügung:

Synapsen hinzufügen oder entfernen: Offensichtlich liegt als einfacher Strukturmutationsoperator nahe, einzelne synaptische Verbindungen an- oder auszuschalten.

Schwach verbundenes Neuron entfernen: Ein weiterer offensichtlicher Operator ist das Entfernen eines beliebigen, zufällig gewählten Neurons. Dieser Operator wird in der vorgestellten Implementierung leicht abgewandelt verwendet: Es wird das am schwächsten verbundene Neuron entfernt. Das am schwächsten verbundene Neuron ist dasjenige mit der niedrigsten Summe der Gewichtsbeträge aller seiner ein- und ausgehenden Verbindungen. Die Begründung für dieses Vorgehen liegt darin, dass das Entfernen eines Neurons ein recht großer Eingriff in das neuronale Netz ist, der das Verhalten des Netzes im Extremfall sehr stark verändern kann. Der Sinn von Mutationsoperatoren im Rahmen der evolutionären Optimierung ist jedoch, nur eine kleine Bewegung über den Parameterraum durchzuführen. Indem also das am schwächsten verbundene Neuron eliminiert wird, versucht man, den Sprung im Parameterraum möglichst klein zu halten, wobei dies gerade bei rekurrenten neuronalen Netzen natürlich nie garantiert werden kann.

Schwach verbundenes Neuron hinzufügen: Dieser Operator fügt einer zufällig gewählten beliebigen inneren Schicht ein Neuron hinzu. Zusätzlich werden wenige synaptische Verbindungen von und zu dem neuen Neuron ebenfalls zufällig erstellt und mit niedrigen Gewichtswerten aus dem Intervall $[-0,5; 0,5]$ zufällig initialisiert. Das Neuron wird zunächst schwach verbunden, um den Sprung im Parameterraum möglichst klein zu halten, so dass Lösungskandidaten, auf die dieser Operator angewendet wurde, nicht gleich wieder absterben.

Weitere Operatoren sind dieser Grundausstattung bei späteren Verwendungen des Netzes auf einfache Weise hinzufügbar, z.B. wird in [70, 71] eine Auswahl an weiteren Operatoren vorgestellt, insbesondere auch problemspezifische. Bei jeder Mutation wird mit 80% Wahrscheinlichkeit eine synapsenbasierte Mutation durchgeführt (Gauß-Mutation, Exponentialmutation, Synapsen hinzufügen oder entfernen). Wird eine synapsenbasierte Mutation durchgeführt, so wird zufällig gleichverteilt einer der genannten Operatoren ausgewählt und ausgeführt. Mit 20% Wahrscheinlichkeit wird eine neuronenbasierte Mutation durchgeführt (schwach verbundenes Neuron entfernen/hinzufügen). Wird eine neuronenbasierte Mutation durchgeführt, so werden zufällig gleichverteilt Neuronen hinzugefügt (schwach verbundenes Neuron hinzufügen) oder entfernt (schwach verbundenes Neuron entfernen).

Kreuzung

Kreuzungsoperatoren haben die Aufgabe, zwei Genome zu verschmelzen. Der oder die durch den Kreuzungsoperator entstehenden Genome enthalten dann jeweils Merkmale von beiden eingegebenen Genomen. Wie aus verschiedenen Arbeiten hervorgeht [25, 70] ist es aufgrund der Art der verteilten Informationsverarbeitung und -repräsentation bei neuronalen Netzen im Allgemeinen nicht sinnvoll, Kreuzungsoperatoren zu definieren. Im Rahmen von Projekten der evolutionären Robotik und Schwarmrobotik hat der Diplomand jedoch die Erfahrung gemacht, dass die durch einen Kreuzungsoperator erzeugten, häufig sehr großen, wenn auch ungezielten Sprünge im Parameterraum gerade in den ersten Generationen einer Evolution hilfreich sein können. Insofern wurde im Rahmen der Implementierung des strukturevolvierbaren neuronalen Netzes ein Kreuzungsoperator entwickelt. Die Kreuzung ist in folgende Phasen unterteilt (vergleiche Abb. 5.5 auf der folgenden Seite):

Netze verschmelzen: Es wird zunächst ein neues Netz erzeugt, welches dieselbe Anzahl an Eingabeneuronen, Ausgabeneuronen und Neuronenschichten beinhaltet wie die eingegebenen Netze. Danach werden sämtliche Verbindungen und inneren Neuronenschichten beider eingegebenen Netze in das neue Netz so einkopiert, dass die Strukturen beider Netze zunächst erhalten bleiben. Schicht i des neuen Netzes enthält hiernach also die Neurone aus Schicht i beider eingegebener Netze und zusätzlich sind die Verbindungen zwischen den Neuronen topologisch identisch mit denjenigen aus den eingegebenen Netzen. Nach Abschluss dieses Schrittes existieren die inneren Teile beider Netze nebeneinander im neuen Netz. Falls Verbindungen, welche die Ausgabeschicht als Ziel haben, kollidieren, werden sie zufällig gleichverteilt entweder vom ersten oder zweiten eingegebenen Netz übernommen.

Neues Netz beschneiden: Da in dem so gebildeten Netz nun verhältnismäßig viele Neurone vorhanden sind, wird jedes innere Neuron mit einer Wahrscheinlichkeit von 50% gelöscht. Seien n_1 und n_2 die Zahlen der versteckten Neuronen der eingegebenen Netze. Dann enthält das neue Netz nun durchschnittlich $\frac{n_1+n_2}{2}$ innere Neurone. So wird sichergestellt, dass durch den Kreuzungsoperator kein Wachsen der Netze ins Unermessliche provoziert wird.

Schwache Verbindungen hinzufügen: Durch die Kombination von zwei getrennten, in das neue Netz kopierten Topologien und das anschließende Beschneiden existieren im neuen Netz relativ wenige Verbindungen. Um dem entgegenzuwirken, werden zufällig gewählte Verbindungen zwischen Neuronen eingeschaltet, bis der durchschnittliche Anteil existierender Verbindungen an der Gesamtzahl möglicher Verbindungen über beide eingegebene Netze erreicht oder überschritten ist. So erstellte synaptische Verbindungen werden, wie schon beschrieben, mit niedrigen Gewichtswerten aus dem Intervall $[-0,5; 0,5]$ zufällig initialisiert.

Das so gebildete neue Netz wird dann als Ergebnis der Kreuzung zurückgegeben.

5.3.8 Hybridtraining

Durch Backpropagation als weiteres Implementierungsmerkmal gibt es die Möglichkeit, das Netz einem Hybridtraining aus evolutionärer Optimierung und Backpropagation zu unterziehen. Hierbei wird gewissermaßen der Startpunkt für Backpropagation evolviert. In der Literatur wurden hiermit im Rahmen bestimmter Problemstellungen gute Erfahrungen gemacht [5, 70]. Da diese Art Training in der Diplomarbeit nicht eingesetzt wird, sei sie hier nur kurz erwähnt.

5.3.9 Visualisierungsmerkmale

Ebenso kurz beschrieben sei die Implementierung von Visualisierungsmerkmalen: Es gibt die Möglichkeit, ein neuronales Netz in eine Datei zu exportieren, die von der Graphenvisualisierungssoftware *GraphViz* [20] gelesen und dargestellt werden kann. Verschiedene Einstellungsmöglichkeiten erlauben hier verschiedene Arten der Visualisierung: Beispielsweise können die Gewichtswerte von sehr schwachen Verbindungen aus Gründen der Übersichtlichkeit ausgeblendet oder schwächer dargestellt werden.

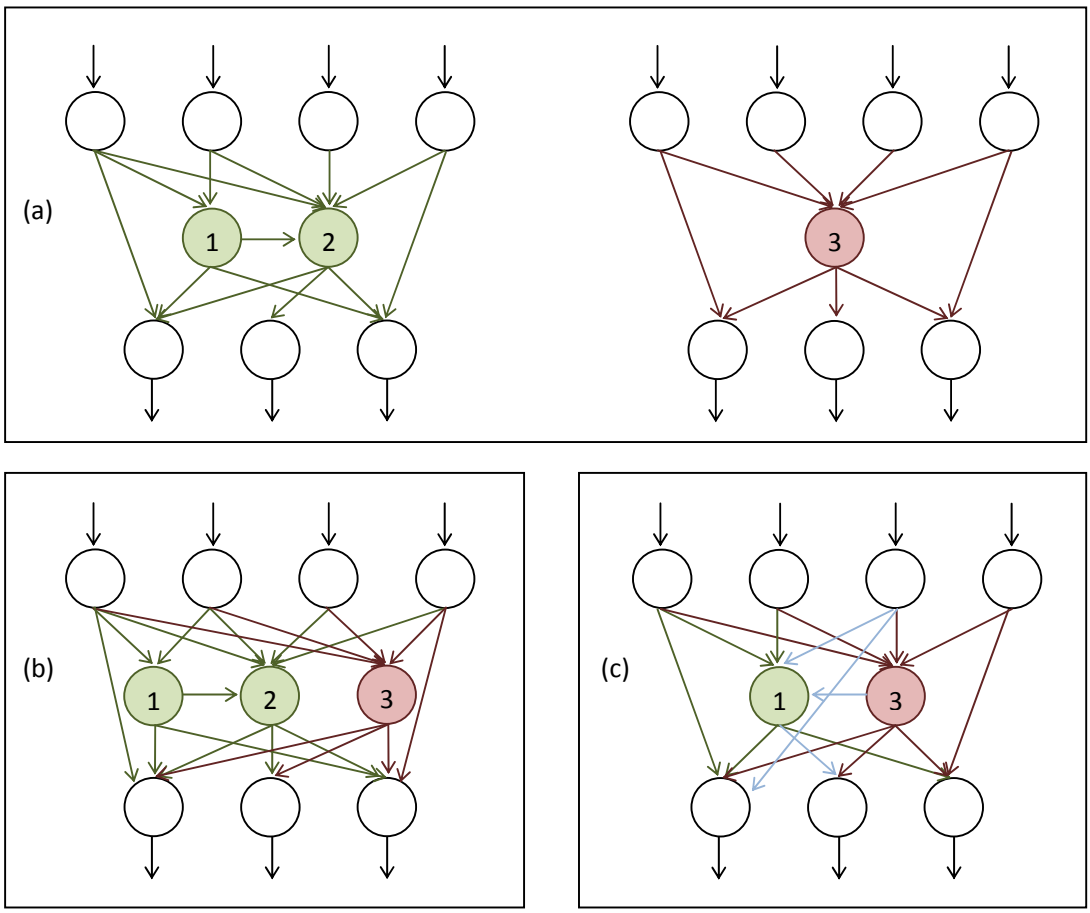


Abbildung 5.5: Illustration der Kreuzung neuronaler Netze anhand eines einfachen Beispiels mit nur einer verdeckten Schicht.
Abbildungsteil (a) zeigt beide Elternnetze, auf die der Kreuzungsoperator angewendet wird. Sie sollen im Folgenden als „grünes“ und „rotes“ Netz bezeichnet werden. Die inneren Neurone sind zur Orientierung nummeriert. Sowohl das grüne, als auch das rote Netz besitzen zwei Vorwärtsabkürzungen zwischen den jeweils gleichen Ein- und Ausgabeneuronen, sowie einige Vorwärtsverbindungen. Das grüne Netz enthält zusätzlich noch eine laterale Verbindung von Neuron 1 zu Neuron 2.
Teil (b) zeigt das Netz, welches nach der Verschmelzung der beiden Eltern entstanden ist. Es gab zwei Synapsenkollisionen, nämlich zwischen den Vorwärtsabkürzungen, welche die Ausgabeschicht zum Ziel haben. Hier wurde zufällig gleichverteilt eine Verbindung vom roten und eine vom grünen Netz übernommen. Beide inneren Teile, sowohl des roten, als auch des grünen Netzes existieren im verschmolzenen Netz nun – modulo der Kollisionen – nebeneinander.
Teil (c) zeigt das Zielnetz nach Beschneidung und dem Hinzufügen schwacher Verbindungen. In diesem Fall wurde Neuron 2 aus dem Netz entfernt, und mit ihm alle Verbindungen, an denen es beteiligt war. Um den Verlust an Konnektivität auszugleichen, wurden dem Netz zufällig einige schwache Verbindungen hinzugefügt (hier hellblau dargestellt).

5.4 Implementierung der Schwarmsimulationssoftware SwarmWorld

Nun soll auf die in der Diplomarbeit implementierte Schwarmsimulationssoftware *SwarmWorld* eingegangen werden. Zunächst soll im folgenden Abschnitt beschrieben werden, welche Ansprüche an die Schwarmsimulationssoftware gestellt werden. In den nachfolgenden Abschnitten wird dann ein Überblick über die technische Realisierung der Software gegeben. Die technische Beschreibung erfolgt dann bedingt durch die Integration einer Abstraktionsebene in die Software auf zweierlei Weise: Zum einen werden die internen technischen Mechanismen zur Realisierung der Simulation losgelöst von der konkret zu simulierenden Welt beschrieben (Entwicklerperspektive). Dem gegenübergestellt wird jeweils die Benutzerperspektive: Hier wird dargestellt, wie man die Software einsetzt, um auf einfache Weise eine konkrete simulierte Welt zu erschaffen. Aufgrund der Code-Menge bewegen sich Beschreibungen der Realisierung zum Großteil auf semantischer Ebene: Es wird mehr Wert darauf gelegt, die Fähigkeiten der Software zu beschreiben, als auf die Beschreibung der exakten Implementierung mit all ihren Spezialfällen und technischen Details, welche während der Entwicklung Beachtung gefunden haben.

5.4.1 Anspruch und vorhandene Frameworks

Im Bereich der Multiagentensimulation existieren bereits verschiedene Softwareframeworks, welche sich in Umfang, Abstraktionsgrad, technischen Aspekten, Anspruch an den Benutzer und Simulationszielsetzung unterscheiden. Zwei populäre Vertreter unter diesen Frameworks sind *RePast* [59] und *MASON* [42]. Ein Vergleich von Multiagenten-Simulationssystemen findet sich in [3]. Beide Frameworks sind in JAVA geschrieben, beide dienen der agentenbasierten Simulation sozialer Abläufe und der Abstraktionsgrad beider Frameworks ist sehr hoch. Beide Frameworks lassen eine nahezu beliebige Definition des Simulationsmodells und folglich auch der beteiligten Agenten zu. Insbesondere sind sie nicht spezialisiert auf Simulationen in zweidimensionalen Umgebungen unter Berücksichtigung physikalischer Gesetze. Dies führt insgesamt zu einer gewissen Einarbeitungszeit und Implementierungsphase im Vorfeld konkreter Experimente. Allerdings bieten beide Frameworks Möglichkeiten, die Multiagentensimulation mit einem evolutionären Algorithmus zu kombinieren (*RePast* durch direkt eingebaute Merkmale, *MASON* durch das Schwesterprojekt *Java Evolutionary Computation Toolkit* [41]). Bedingt durch den Abstraktionsgrad der Evolutionsframeworks ist hier allerdings ebenfalls erheblicher Konkretisierungsaufwand zu erwarten. Für *MASON* existiert sogar ein Zusatzmodul, welches ein 2D-Weltmodell samt der Dynamik starrer Körper implementieren soll, jedoch auf den Diplomanden im Vergleich zu spezialisierten 2D-Physikframeworks wie *Phys2D* oder *JBox2D* einen unausgereiften und physikalisch nicht stabilen Eindruck gemacht hat.

In dieser Diplomarbeit ist es das Ziel des Diplomanden, Multiagentensimulationen einzig in simulierten 2D-Welten durchzuführen, welche physikalischen Gesetzen starrer Körper gehorchen. Da der Diplomand bereits im Rahmen verschiedener Projekte Erfahrungen in der Benutzung und Anpassung von Physikframeworks, sowie der Konzeption und Implementierung von Multiagentensimulationen sammeln konnte, ist der Einarbeitungsaufwand sowohl bei *MASON* als auch *RePast* mindestens so hoch, wie derjenige zum Aufsetzen eines eigenen, auf die eigenen Bedürfnisse zugeschnittenen Simulationsprogramms. Zeitgleich wäre zu befürchten, dass unter dem gewaltigen Überhang an Funktionalität, den beide Frameworks bieten, die rechnerische Effizienz leidet – welche für die Durchführbarkeit von freien Evolutionen in der Art, wie sie in Abschnitt 4.4 beschrieben wird, unabdingbar ist.

Die Ansprüche an das zu entwickelnde Schwarmsimulationsprogramm *SwarmWorld* sollen in den folgenden Abschnitten definiert werden.

Schwarmwelten, Schwärmer und passive Objekte

Es soll eine zweidimensionale, räumlich kontinuierliche, jedoch zeitdiskretisierte Welt simuliert werden, welche den Gesetzen der Dynamik starrer Körper gehorcht, also effiziente Mechanismen

zur Kollisionsdetektion und -antwort bereitstellt. In der simulierten Welt soll unterschieden werden zwischen Schwärmern (aktiven Agenten, welche Sensorik, Aktorik und eine Kontrollstruktur besitzen) sowie passiven Objekten (welche keine Sensorik, Aktorik und Kontrollstruktur besitzen).

Abstraktionsgrad

Der Abstraktionsgrad der Simulationssoftware soll hoch genug für eine Vielzahl verschiedenster Experimente sein. Konkretisierung dieser abstrakten Welt soll hauptsächlich mittels Implementierung von Schwärmern (bzw. deren Sensorik, Aktorik und Kontrollstruktur) und passiven Objekten erfolgen und (JAVA-Kenntnisse vorausgesetzt) auf einfache und effiziente Weise möglich sein. Idealerweise so einfach, dass das Framework z.B. im Rahmen von Lehrveranstaltungen weiterverwendet werden kann.

Sensoren, Emitter und Sensorikdomänen

Schwärmer sollen Informationen über ihre Umwelt durch ihre Sensorik aufnehmen. Jeder Sensor ist hierbei einer Sensorikdomäne zugeordnet (z.B. „Sehsinn“ oder „Geruchssinn“) und kann Emitter wahrnehmen, welche in derselben Sensorikdomäne liegen. So kann ein Sichtsensor sichtbare Emitter wahrnehmen, während das Wahrnehmen von Geruchs-Emittern Sensoren der Domäne „Geruchssinn“ vorbehalten ist. Emitter können Schwärmer oder passive Objekte sein. Die Art und Anzahl der Sensorikdomänen soll vom Benutzer frei definierbar und nicht begrenzt sein. Die Definition solcher Sensorikdomänen erlaubt die Auftrennung der rechnerischen Abwicklung in mehrere Teile, was die Effizienz erhöhen kann.

Subjektive Weltsicht von Schwärmern

Jeder Schwärmer soll durch seine Sensorik eine subjektive Weltsicht besitzen, welche von der Simulationssoftware automatisch zur Verfügung gestellt wird. Hierdurch wird die Effizienz der Implementierung konkreter Schwärmer stark erhöht, da die Kontrollstruktur eines Schwärmers direkt auf die subjektive Weltsicht zurückgreifen kann.

Physikalische Domäne

Analog zu den Sensorikdomänen gibt es auch eine physikalische Domäne, die alle Objekte beinhaltet, welche den Gesetzen der Dynamik starrer Körper gehorchen, also z.B. von einer Kollisionsdetektion verwaltet werden sollen. Hierzu gehören die physikalischen Körper der Schwärmer, und für passive Objekte ist individuell wählbar, ob sie in die physikalische Domäne gehören oder nicht. Durch diese Wahlfreiheit wird berücksichtigt, dass passive Objekte in der simulierten Welt völlig verschiedene Rollen übernehmen können: So ist es für ein Fußballobjekt sinnvoll, in der physikalischen Domäne zu liegen, um mittels Kollisionsdetektion und Übertragung physikalischer Kräfte mit den Schwärmern interagieren zu können. Dies erscheint für ein Lockstoff-Objekt, das einzig und allein als durch Sensoren wahrnehmbarer Emitter konzipiert ist und die Schwärmer ansonsten nicht beeinflusst, weniger sinnvoll. Ein solches Objekt braucht dann in der Kollisionsdetektion nicht betrachtet zu werden und würde demzufolge auch nicht zur physikalischen Domäne gehören, was Rechenzeit spart[4].

Effizienz

Die Erwähnung der Rechenzeit führt uns zur Effizienz: Die Simulationssoftware muss ausreichend schnell arbeiten, um in einem evolutionären Algorithmus als Evaluationsfunktion dienen zu können. Dies muss auch der Fall sein, wenn z.B. viele hundert passive Objekte in der Welt

[4] Wie solche Objekte verwaltet werden, wird in Abschnitt 5.4.6 noch beschrieben.

existieren. In Abschnitt 5.2 wurde bereits beschrieben, auf welche verschiedenen Weisen die verwendete Evolutionssoftware die Evaluation parallelisieren kann. Insofern wäre eine weitere Ebene der Parallelisierung direkt in der Schwarmsimulation kontraproduktiv. Sämtliche für die Simulation notwendigen Berechnungen sollen daher sequentiell, also in nur einem Thread, ablaufen.

Graphische Darstellung der Simulation

Die graphische Darstellung soll losgelöst von der eigentlichen Simulation sein: Die Simulation soll also ungebremst und ohne Visualisierung ablaufen können, oder mit regelbarer Geschwindigkeit bei zugeschalteter Visualisierung. Hierbei sollen zwei Visualisierungsmodi existieren. Der erste Visualisierungsmodus soll ohne das Zutun des Benutzers von der Simulationssoftware selbst realisiert werden und z.B. zum Zweck der Fehlerdiagnose die realen physikalischen Gegebenheiten darstellen. Er soll auf jede konkrete Simulation direkt anwendbar sein. Ein zweiter Visualisierungsmodus soll dann vom Benutzer völlig frei definiert werden können.

Nachdem nun die Ansprüche an die Simulationssoftware definiert sind, soll auf ihre tatsächliche technische Realisierung eingegangen werden. Die Beschreibung der *SwarmWorld* kann hier nicht erschöpfend sein, was ihre Fähigkeiten oder gar Implementierungsdetails betrifft.

5.4.2 Entwickler- und Benutzerperspektive

Die Klassen der Simulationssoftware, mit denen der Benutzer später in Berührung kommt, sind weitestgehend abstrakt gehalten. Abstrakte Klassen in JAVA sind Klassen, von denen Bestandteile zur weiteren Verwendung vom Benutzer zunächst im Rahmen einer Klassenvererbung konkretisiert werden müssen, um einsatzfähig zu sein. Die Simulationssoftware *SwarmWorld* wird also im folgenden aus zwei Perspektiven beschrieben. Die Entwicklerperspektive betrachtet zu den verschiedenen Aspekten der Schwarmwelt Details, welche als Kern der Simulation fest implementiert wurden. Die Benutzerperspektive betrachtet hingegen, welche Bestandteile der verschiedenen Aspekte abstrakt gehalten sind, also vom Benutzer, der spezielle Simulationsexperimente durchführen will, im Rahmen einer Vererbung der abstrakten Klassen konkretisiert werden müssen. Es soll nun darauf eingegangen werden, auf welche Weise Schwarmwelt, Schwärmer und passive Objekte realisiert sind und wie die zugehörigen Programmbestandteile grob aufgebaut sind. Im Verlaufe dessen soll beiden genannten Perspektiven Rechnung getragen werden. Es wird mit dem phyikalischen Unterbau der Simulationssoftware begonnen.

5.4.3 Verwendung des Physikframeworks Phys2D

Dieser Abschnitt beschreibt den physikalischen Kern der Schwarmsimulationssoftware und wird daher ausschließlich aus Entwicklerperspektive beschrieben. Für JAVA existieren verschiedene Frameworks, welche in der Lage sind, die Dynamik starrer Körper zu simulieren. Zwei populäre unter diesen sind *JBox2D* [53], der JAVA-Port des Physikframeworks *Box2D* für C++ [11], und das verwandte *Phys2D* [23]. Nach ausgiebigem Test beider Frameworks hinsichtlich Bedienbarkeit der API und Performanz stellte der Diplomand fest, dass beide Frameworks seine Kriterien erfüllen und entschied sich für die Verwendung von *Phys2D* aufgrund leichter Vorteile der API. Jedoch wurden in dem Bewusstsein, dass auch *JBox2D* für weitere Verwendung sehr wohl in Frage käme und sehr verwandt mit *Phys2D* ist, die Berührungspunkte zwischen physikalischem Unterbau und Schwarmsimulationssoftware so gestaltet, dass ein späterer Umstieg auf *JBox2D* mit wenig Aufwand möglich ist. So ist die Schwarmsimulationssoftware mit wenig Aufwand auch dann weiterverwendbar, wenn die Entwicklung eines der beiden Physikframeworks nicht mehr weitergeführt wird.

Wie alle 2D-Physikframeworks stellt *Phys2D* eine Repräsentation starrer 2D-Körper verschiedener Geometrien (z.B. Kreise, Rechtecke, Polygone) zur Verfügung, sowie eine Repräsentation eines 2D-Raumes, in dem diese Körper enthalten sind. In diesem Raum können physikalische Kräfte auf die Körper einwirken, so wie die Körper solche Kräfte z.B. durch Kollision auf andere Körper auftragen können. Diese Repräsentationen von Raum und Körpern wurden im Rahmen der Implementierung durch Vererbung genutzt und wesentlich erweitert.

5.4.4 Realisierung der Schwarmwelt und Optimierung von Phys2D

Entwicklerperspektive: Die Schwarmwelt wird realisiert, indem die zweidimensionale Welt, welche von *Phys2D* simuliert wird, im Sinne der objektorientierten Programmierung vererbt wird. Die Schwarmwelt bringt durch die Vererbung der *Phys2D*-Welt also bereits die notwendigen Fähigkeiten zur Kollisionsdetektion und Kollisionsantwort mit. Die *Phys2D*-Welt wurde erweitert um dynamische Datenstrukturen für die Verwaltung von Schwärmern, passiven Objekten und Sensorikdomänen – Elemente, auf die gleich näher eingegangen wird. Weiter wurden Operatoren auf diesen Datenstrukturen implementiert, beispielsweise zum Hinzufügen und Entfernen von Schwärmern und Objekten.

Zur Spezialisierung auf Schwarmsimulationen wurden die Mechanismen zur Kollisionsdetektion der Broad Phase[5] gegenüber den originalen Mechanismen aus *Phys2D* optimiert. Um diese Optimierung zu beschreiben, wird zunächst kurz auf die Broad Phase in *Phys2D* eingegangen. In *Phys2D* wird ein QuadTree [56] eingesetzt, welcher den Raum rekursiv und jeweils gleichmäßig in achsenparallele Rechtecke viertelt und bei jeder dieser Vierteilungen für die entstehenden Zellen überprüft, ob eine von zwei Bedingungen erfüllt ist:

1. Für die Zahl n der verbleibenden Objekte in der betrachteten Zelle gilt $n \leq n_{\max}$, wobei $n_{\max}$ ein frei wählbarer Parameter ist.
2. Für die Tiefe der Rekursion r der Betrachteten Zelle gilt $r \geq r_{\max}$, wobei $r_{\max}$ ein frei wählbarer Parameter ist.

Ist eine der beiden Bedingungen erfüllt, so stoppt die Rekursion und die in der betrachteten Zelle enthaltenen Körper werden mit Algorithmen der Narrow Phase untereinander auf Kollision geprüft. Mittels dieser Parameter kann man also sehr stark beeinflussen, welchen Geschwindigkeitsvorteil der QuadTree tatsächlich bietet: Ist $n_{\max}$ zu klein und $r_{\max}$ zu groß gewählt, so sinkt zwar die Anzahl der durchzuführenden Tests in der Narrow Phase, der Aufwand den QuadTree zu erstellen kann diesen Vorteil jedoch wieder ausgleichen – immerhin muss mittels vereinfachter Tests beim Teilen einer Vaterzelle für jeden Körper in der Vaterzelle festgestellt werden, in welcher Teilzelle er liegt. Ist umgekehrt $n_{\max}$ zu groß oder $r_{\max}$ zu klein gewählt, so ist der QuadTree sehr schnell erstellt, da nicht viele Zellteilungen vorgenommen werden müssen. Die Anzahl der durchzuführenden Tests in der Narrow Phase ist jedoch sehr groß. Die konkrete Wahl dieser beiden Parameter hängt im Allgemeinen von der konkreten Simulation ab und soll hier nicht weiter betrachtet werden. Vielmehr soll nun, nachdem das grundlegende Prinzip der Broad Phase von *Phys2D* bekannt ist, eine Optimierung durchgeführt werden.

Es ist offensichtlich, dass der oben beschriebene QuadTree den Aufwand der Kollisionsdetektion drastisch reduzieren kann, wenn die Körper gleichmäßig im Raum verteilt sind. In diesem Fall liegt in jeder neuen Zelle ungefähr ein Viertel der in der Vaterzelle vorhandenen Körper. Da die Simulationssoftware jedoch Schwärme simulieren soll, ist eine solch gleichmäßige Verteilung der physikalisch zu berücksichtigenden Schwarmkörper im Raum äußerst unwahrscheinlich. Wahrscheinlicher ist eine Gruppenbildung, was die Performanz des QuadTree-Algorithmus einbrechen lässt, da unter Umständen gewisse Zellen immer und immer wieder geteilt werden müssen, ohne dass sich die Zahl der in ihr enthaltenen Körper (und damit die der Zugehörigkeitstests beim Teilen) signifikant verringert. Im schlimmsten Fall ist man schließlich bei der Rekursionstiefe $r_{\max}$ angelangt, ohne die Anzahl der auszuführenden Narrow-Phase-Tests signifikant reduziert zu haben. In diesem Fall verursachen

[5] Bei Kollisionsdetektionen unterscheidet man zwei Phasen: Die *Broad Phase* und die *Narrow Phase*. Während der Broad Phase wird anhand sehr einfacher, schneller, jedoch ungenauer Kollisionsausschlusstests festgestellt, bei welchen in einem Raum enthaltenen Tupeln von Körpern überhaupt Kollisionen möglich sind. Hierdurch erhält man eine Liste von Tupeln von Körpern, welche möglicherweise miteinander kollidieren.

Anschließend werden in der Narrow Phase die in der Broad Phase gebildeten Tupel mittels spezialisierter Algorithmen auf wirkliche Kollisionen überprüft: So benötigt man für einen Kollisionstest zwischen Kreis und Polygon einen anderen spezialisierten Algorithmus als zwischen zwei Polygonen. Diese spezialisierten Algorithmen sind i.A. deutlich rechenaufwändiger als die einfachen, welche in der Broad Phase verwendet werden.

Ohne eine Broad Phase müsste man die spezialisierten Algorithmen für jedes mögliche Körpertupel der vorhandenen Körper aufrufen, was ein inakzeptabler Rechenaufwand wäre.

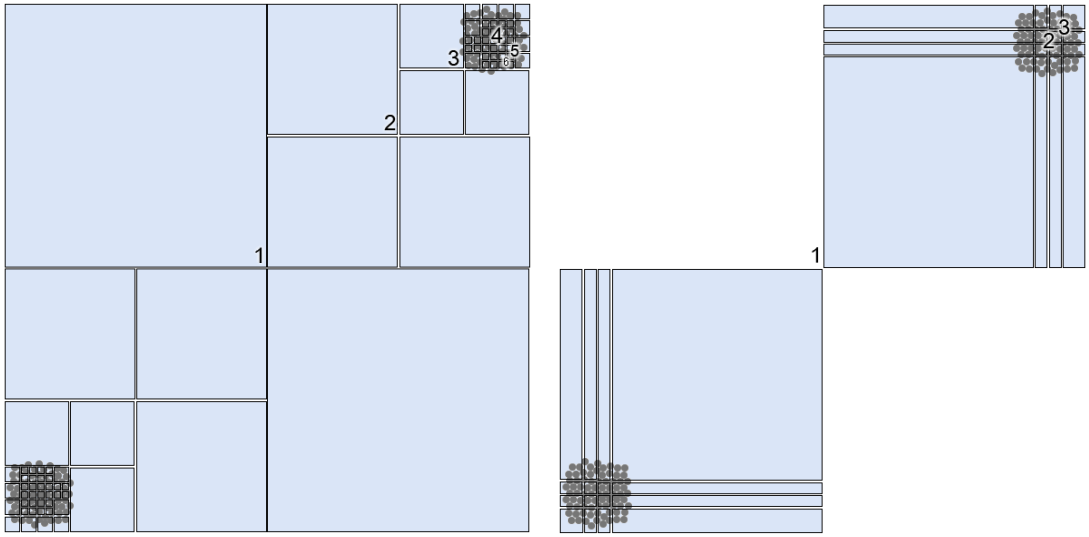


Abbildung 5.6: Illustration von originalem QuadTree (links) und optimiertem QuadTree (rechts) mit gleichen Parametern $n_{\max} = 10$ und $r_{\max} = 6$ auf zwei Objektgruppen mit jeweils 100 Objekten. Bei beiden Bildern sind einige Schnittpunkte von Teilungsgeraden mit der jeweiligen Rekursionstiefe der Teilzellen beschriftet. Die Ur-Zelle, welche noch alle Objekte enthält, ist mit Tiefe 0 bezeichnet. Beim rechten QuadTree wurden die obere linke und untere rechte Zelle in Tiefe 1 nicht eingefärbt. Dies resultiert aus einer weiteren kleinen Optimierung (Zellen mit 0 und 1 Elementen werden im optimierten QuadTree direkt bei der Teilung komplett verworfen) und ist hier nicht weiter von Bedeutung.
Offensichtlich führt der Original-QuadTree sehr viele Zellteilungen durch, die sogar bis in die 6. Rekursionstiefe gehen – einzelne Objekte werden also 6mal auf Mitgliedschaft in Teilzellen überprüft. Gleichzeitig sind die ersten Zellteilungen relativ aufwändig, da erst bei Rekursionstiefe 4 eine effektive Viertelung der Objektmenge erfolgt. Bis dahin wird also ein Großteil der Objekte immer wieder auf Teilzellenmitgliedschaft geprüft. Beim optimierten QuadTree findet eine solch effektive Viertelung bereits in Rekursionstiefe 2 statt, so dass in Rekursionstiefe 3 das Ziel bereits erreicht ist und weitere Rekursionen unnötig sind.

sowohl das Erstellen des QuadTrees als auch die anschließenden Narrow-Phase-Tests sehr viel Rechenaufwand.

Es liegt also im Falle der Schwarmsimulation nahe, ein wenig mehr Rechenaufwand in gezielteres Teilen der Zellen beim Erstellen des QuadTrees zu investieren. Man könnte durch diese Investition gleich mehrere Vorteile ernten:

1. Eine Verringerung des Rechenaufwandes durch Spezialfälle wie den oben geschilderten oder abgeschwächte Formen davon
2. Eine Verringerung des Rechenaufwandes durch weniger Zellteilungen im Verhältnis zur Anzahl der Körper
3. Eine Verringerung des Rechenaufwandes durch weniger Narrow-Phase-Tests: Durch die optimalere Zellteilung erhält man die Möglichkeit, $n_{\max}$ kleiner zu wählen, was tendenziell die Anzahl der Narrow-Phase-Tests reduziert.

Im Falle dieser Schwarmsimulationssoftware ist für eine gezieltere Zellteilung die Methode gewählt worden, die betrachtete Vaterzelle so zu teilen, dass der Schnittpunkt der beiden zu x- und y-Achse parallelen Teilungsgeradensegmente genau auf dem Schwerpunkt aller Körper in der Vaterzelle liegt. Durch dieses Verfahren wurde in verschiedenen Tests mit ungleichmäßig platzierten Körpern ein Performanzgewinn von $\approx 30\%$ erzielt. Eine Vergleichsabbildung beider Verfahren, angewendet mit denselben Parametern $n_{\max}$ und $r_{\max}$ auf eine gleichartig zufällig verteilte Objektkonfiguration, findet sich in Abbildung 5.6.

Benutzerperspektive: Die abstrakte Schwarmwelt wird im Rahmen einer Vererbung vom Benutzer konkretisiert. Im Rahmen einer Vererbung kann die erbende, also konkretisierte Klasse gegenüber der vererbenden beliebige Eigenschaften hinzugewinnen. Die Betrachtung die-

ser und der folgenden abstrakten Klassen ist aus Benutzerperspektive also auf Codebestandteile beschränkt, deren Konkretisierung durch explizit in JAVA als abstrakt gekennzeichnete Methoden erzwungen wird. Im Falle der abstrakten Schwarmwelt kann der Benutzer beispielsweise durch die Implementierung abstrakter Methoden die Schwarmwelt initialisieren. Er kann weiter festlegen, zu welchen Zeitpunkten in der Simulation bestimmte Ereignisse passieren, z.B. Objekte erscheinen, oder schlicht Statistiken über die Welt sammeln. Diese Beschreibung bewegt sich absichtlich auf niedriger Detailstufe, da außer der Realisierung der Schwarmwelt bis jetzt noch keine weiteren Realisierungen betrachtet wurden. Im Rahmen der Beschreibung des Ablaufs eines Simulationsschrittes (Abschnitt 5.4.8) wird auf die vom Benutzer konkretisierbaren Elemente der abstrakten Schwarmwelt noch einmal genauer eingegangen.

5.4.5 Realisierung von Schwärmern und passiven Objekten

Nachdem nun die Schwarmwelt betrachtet wurde, die Schwärmer und passive Objekte verwaltet, liegt es nahe, deren technische Realisierung zu betrachten.

Entwicklerperspektive: Wie schon angedeutet, liefert *Phys2D* eine Repräsentation verschiedener Formen starrer Körper mit. Diese Repräsentation wird sowohl für Schwärmer, als auch für passive Objekte verwendet: Sowohl ein Schwärmer als auch ein passives Objekt besitzen einen starren Körper als Form. Für jedes Objekt ist zusätzlich wählbar, ob es Bestandteil der physikalischen Domäne ist, also von der Kollisionsdetektion und -antwort beeinflusst werden soll. Ein Schwärmer besitzt zusätzlich noch eine Liste von Sensoren, um Informationen aus der Umwelt aufzunehmen und eine Liste von Emittern, um von anderen Sensoren wahrgenommen zu werden. Ein weiterer essentieller Bestandteil, den ein Schwärmer im Gegensatz zu einem passiven Objekt besitzt, ist eine eigene Kontrollstruktur. Wird diese ausgeführt, so kann sie auf von Sensoren gelieferte Eingaben zurückgreifen. Aufbauend auf den Eingaben und auf z.B. Schwärmerinternen Zuständen können dann Aktionen ausführt werden. Aktionen können z.B. Bewegungen sein, oder das Erzeugen von Objekten in der Welt. Der abstrakte Schwärmer stellt für solche Aktionen vorgefertigte Operatoren zur Verfügung, die direkt in der Kontrollstruktur ausgeführt werden können.

Benutzerperspektive: Analog zur abstrakten Schwarmwelt sind Schwärmer und passive Objekte ebenfalls abstrakt implementiert. Konkreten Schwärmern und passiven Objekten können durch Vererbung beliebige Merkmale verliehen werden. Erzwungen wird bei der Vererbung von Schwärmern beispielsweise die Konkretisierung der Kontrollstruktur und seines physikalischen Körpers. Der abstrakte Schwärmer stellt weiterhin Operatoren zur Verfügung, über die ihm Sensoren und Emitter hinzugefügt werden können. Abstrakte passive Objekte erzwingen die Konkretisierung von ihrem physikalischen Körper, sowie eine Typzuordnung und die Angabe, ob das bezeichnete Objekt sich in der physikalischen Domäne befindet oder nicht.

5.4.6 Realisierung von Sensoren, Emittern und Sensorikdomänen

Analog zu den bisherigen Realisierungen einzelner Simulationsbestandteile sind auch Sensoren und Emitter zum Teil abstrakt gehalten.

Entwicklerperspektive: Sensoren werden durch ihre Wahrnehmungsbereiche aus subjektiver Sicht des zugehörigen Schwärmers repräsentiert. Ein kreisförmiger Sichtbereich würde durch seinen Mittelpunkt subjektiv vom Schwärmer aus (meist der Schwärmer selbst, also $(0, 0)$) und seinen Radius bestimmt, ein polygonaler Sichtbereich durch die Koordinaten seiner Eckpunkte aus subjektiver Sicht des Schwärmers. Demzufolge werden Sensoren implementiert durch eine Form. Zur Implementierung dieser Form werden die Körperrepräsentationen für z.B. Polygone oder Kreise genutzt, welche *Phys2D* mitbringt.

Emitter, also Objekte, welche von Sensoren wahrgenommen werden können, sind entweder passive Objekte, oder aber Schwärmern zugehörig. Das Konstrukt solcher Schwärmer-Emitter ist entwickelt worden, damit auch Schwärmer von Sensoren wahrgenommen werden können. Intern werden auch Emitter durch einen Körper repräsentiert: Emitter in Form von passiven Objekten besitzen die Form des zugehörigen Objektes, Schwärmer-Emitter diejenige des zugehörigen Schwärmerkörpers. Der abstrakte Schwärmer verfügt, wie schon beschrieben, über Operatoren, mit denen man ihm auf einfache Weise Sensoren bzw. Emitter hinzufügen kann.

Durch die Repräsentation von Sensoren und Emittern als Körper ist es möglich, das Problem der Sensorik (welcher Sensor sieht welchen Emitter?) durch eine angepasste Form der Kollisionsdetektion lösen zu lassen (welcher Sensorsichtbereichs-Körper kollidiert mit welchem Emitter-Körper?).

Dies durchzuführen ist die Aufgabe der Sensorikdomänen: Eine Sensorikdomäne verwaltet alle Sensoren und alle Emitter eines spezifischen Typs. Es existieren also in einer Simulation genau so viele Sensorikdomänen, wie Typen vorkommen, Sensoren und Emitter verschiedener Typen werden also völlig separat behandelt. Stellt eine Sensorikdomäne fest, dass ein Sensor einen bestimmten Emitter wahrnimmt, so wird der Emitter direkt an den Sensor weitergegeben. So wird eine subjektive Weltsicht für jeden Schwärmer erzeugt. Hierbei ist sicherzustellen, dass Emitter eines bestimmten Schwärmers nicht an Sensoren desselben Schwärmers weitergegeben werden.

Benutzerperspektive: Der Sichtbereich eines Sensors sowie seine Bestandteile, welche einen wahrgenommenen Emitter verarbeiten, sind abstrakt und werden demzufolge vom Benutzer definiert – ebenso, welcher Sensorikdomäne der Sensor zugehörig ist. Emitter werden dann – wie oben beschrieben – durch die Konkretisierung von passiven Objekten definiert, oder aber als Schwärmer-Emitter. In beiden Fällen ist die Form vorgegeben, es muss aber die Sensorikdomäne für jeden Emitter spezifiziert werden. Dadurch, dass im Rahmen einer Konkretisierung durch Objektvererbung beliebige Klassenmerkmale hinzugefügt werden können, ist der Benutzer bei Gestaltung der Sensoren, Emitter und deren Datenverarbeitung sehr frei. Sensorikdomänen brauchen aus Benutzerperspektive nicht betrachtet werden, sie sind nicht abstrakt und ihre Verwaltung erfolgt automatisch.

5.4.7 Operatoren auf Datenstrukturen der Schwarmwelt

Wie bereits angedeutet, kann die Zahl der passiven Objekte und Schwärmer und folglich die Anzahl der Emitter- und Sensortypen über die Simulation hinweg variieren. Es müssen also Operatoren auf den Datenstrukturen der Schwarmwelt definiert sein, welche aus Benutzerperspektive angewandt werden können. Einige wichtige dieser Operatoren seien hier kurz aus Entwicklerperspektive beschrieben:

Hinzufügen und Entfernen von passiven Objekten: Ein passives Objekt wird der Objektverwaltungsliste der Schwarmwelt hinzugefügt bzw. aus ihr entfernt. Ist es Bestandteil der physikalischen Domäne, so wird dessen Körper in die Körperverwaltungsliste der zugrundeliegenden *Phys2D*-Simulationswelt hinzugefügt bzw. aus dieser entfernt. Ist es ein Emitter, so wird es in die jeweilige Sensorikdomäne seines Typs hinzugefügt bzw. aus dieser entfernt.

Hinzufügen und Entfernen von Schwärmern: Ein Schwärmer wird der Schwarmsimulationswelt hinzugefügt, indem er der Schwärmerverwaltungsliste hinzugefügt und sein Körper zu der Körperverwaltungsliste der zugrundeliegenden *Phys2D*-Simulationswelt hinzugefügt wird. Zusätzlich wird jeder Emitter und jeder Sensor des Schwärmers den Sensorikdomänen ihres Typs hinzugefügt. Das Entfernen der Schwärmer aus der Schwarmsimulationswelt erfolgt analog.

Hinzufügen und Entfernen von Sensorikdomänen: Wann immer ein Emitter oder ein Sensor eines Typs hinzugefügt werden soll, zu dem noch keine Sensorikdomäne existiert, wird diese Sensorikdomäne automatisch erstellt und der Liste der Sensorikdomänen hinzugefügt.

Analog wird eine Sensorikdomäne, welche nach dem Entfernen eines Emitters bzw. Sensors leer ist, aus der Liste der Sensorikdomänen gelöscht. Diese Vorgänge sind völlig transparent für den Benutzer.

5.4.8 Ausführung eines Simulationsschrittes

Nachdem nun definiert ist, wie die einzelnen Elemente einer Schwarmsimulation technisch aus Benutzer- und Entwicklerperspektive realisiert sind, ist noch die Information wichtig, wie mit ihnen während der Simulation verfahren wird. Wie eingangs bereits erwähnt, besteht eine Simulation aus einer Menge diskreter Simulationsschritte. Ein solcher SwarmWorld-Simulationsschritt ist hierbei aus Entwicklerperspektive unterteilt in folgende Phasen:

1) Ereignisse generieren. In dieser abstrakten und damit vom Benutzer definierten Phase können Ereignisse der Welt modelliert werden. Beispielsweise könnten Objekte und Schwärmer hinzugefügt oder entfernt werden, prinzipiell sind der Phantasie des Benutzers hier keine Grenzen gesetzt.

2) Schwärmer- und Objektlisten aktualisieren. Möchte man der Welt Schwärmer oder Objekte hinzufügen oder solche aus ihr entfernen, so kann man die zugehörigen, oben skizzierten Operatoren zu beliebiger Zeit aufrufen. Die hinzugefügten und entfernten Elemente werden dann aber zunächst in temporären Listen zwischengespeichert und in dieser Phase des Simulationsschrittes erst tatsächlich hinzugefügt bzw. entfernt. Dieser definierte Zeitpunkt ist nötig, um die Datensätze und die Verarbeitung konsistent zu halten – würden diese Operationen beispielsweise mitten in einem Simulationsschritt ausgeführt, so könnten sich Objekt oder Schwärmerlisten während der Verarbeitung ändern, was zu unvorhersehbaren Ergebnissen führen würde.

3) Sensorwerte der Schwärmer zurücksetzen. Eine häufige Fehlerquelle im Simulationsbereich – gerade wenn man Simulationssoftware in der Lehre anwendet – besteht darin, die Sensoren nach Verwendung ihrer Eingabewerte nicht zu reinitialisieren, also auf neue Datenaufnahme vorzubereiten. Insofern wurde dieser Sensor-Reset als eigene Phase in den Simulationsschritt integriert. Für jeden Schwärmer wird hier eine abstrakte Methode (wieder vom Benutzer zu implementieren) ausgeführt, welche die Sensoreingaben reinitialisiert. So kann dieser wichtige Schritt nicht vergessen werden und ist im Code des Benutzers an einem definierten Ort.

4) Sensorische Eingaben der Schwärmer generieren. In dieser Phase wird für alle Sensorikdomänen separat der Kollisionstest zwischen Sensorikbereichen und Emittern ausgeführt (dieser Vorgang ist völlig losgelöst von Kollisionstest und -antwort in der physikalischen Domäne zu betrachten). Hierzu werden zunächst die subjektiv definierten Sensorik- und Emitterkörper anhand der Position und Orientierung der Schwärmer in objektive transformiert. Wie bereits beschrieben, werden dann Emitter, welche mit den Sensorikbereichen von Sensoren kollidieren, direkt zur weiteren Verarbeitung an die betroffenen Sensoren weitergegeben. Am Ende dieser Phase sind also die Sensorik-Eingabewerte aller Schwärmer gesetzt.

5) Kontrollstruktur der Schwärmer ausführen. Für jeden Schwärmer wird die Kontrollstruktur aufgerufen. Diese ist wieder abstrakt, also durch den Benutzer zu realisieren. Die Kontrollstruktur kann auf die Sensorik-Eingaben zugreifen, und diese nach Belieben verwenden und schwärmerinterne Vorgänge ablaufen lassen. Doch sowohl die Verwendung der Sensorikdaten als auch die inneren Logiken sind optional. Das Einzige, was zwangsweise am Ende des Kontrollstruktur-Aufrufs geschehen muss, ist, dass die Kontrollstruktur eine gewünschte Bewegung aus der subjektiven Sicht des Schwärmers mitteilt. Diese Bewegung – bestehend aus einem Positionsänderungsvektor und einer Orientierungsänderung (können beide auch 0 sein) – ist es, die der Schwärmer in diesem Simulationsschritt vollziehen soll.

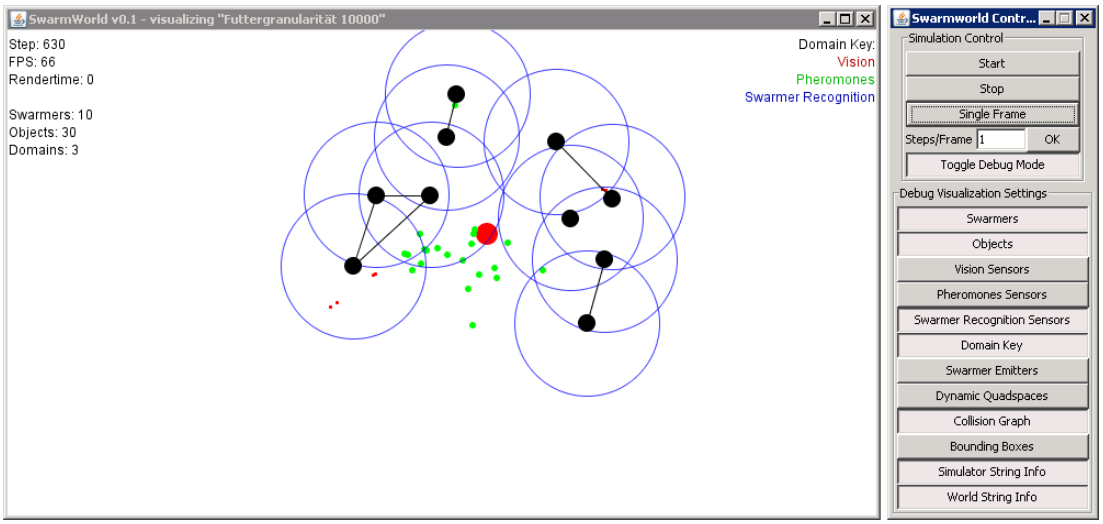


Abbildung 5.7: Graphische Oberfläche der SwarmWorld. Links die Visualisierung der Simulation, rechts die Bedienungsoberfläche, über welche die Simulation kontrolliert werden kann. Zusätzlich können simulierte Elemente (Schwärmer und Objekte) mit der Maus angeklickt werden, um spezifische Informationen über sie anzeigen zu lassen. Dargestellt ist die Bedienungsoberfläche während des physikalischen Visualisierungsmodus. Wird in den benutzerdefinierten Modus gewechselt, so zeigt die Bedienungsoberfläche automatisch andere Elemente an.

6) **Physikalische Simulation.** In diesem Schritt wird für jeden Schwärmer dessen gewünschte subjektive Bewegung zunächst in eine objektive transformiert und diese dann als Kraft auf den physikalischen Schwärmerkörper aufgetragen. Anschließend erfolgt der physikalische Simulationsschritt, in welchem Kollisionen zwischen in der physikalischen Domäne vorhandenen Objekten festgestellt und durch das Physikframework aufgelöst werden.

7) **Statistiken generieren.** In diesem letzten, wieder abstrakten Schritt kann der Benutzer beliebige Daten aus der Schwarmwelt abgreifen und so Statistiken nachhalten. Diese Phase wird nur alle x Simulationsschritte ausgeführt, wobei x ein vom Benutzer eingestellter Parameter ist. Diese Konfigurationsmöglichkeit ist insbesondere nützlich, falls der Benutzer sehr rechenaufwändige Routinen in dieser abstrakten Phase unterbringen möchte.

5.4.9 Visualisierungsmechanismen

Es sei zuletzt die graphische Oberfläche (GUI) der *SwarmWorld* betrachtet, welche aus einer Bedienungsoberfläche für die Simulation und aus der Visualisierung der Simulation selbst besteht (Abb. 5.7). Zunächst ist es wichtig zu erwähnen, dass die Visualisierung losgelöst von der eigentlichen Simulation implementiert wurde. Simulationen können also unvisualisiert und ungebremst ausgeführt werden (zum Beispiel als Evaluationsfunktion im Rahmen eines evolutionären Algorithmus), oder visualisiert mit einstellbarer Ausführungsgeschwindigkeit. Um die Simulationsgeschwindigkeit während visualisierter Simulation zu regeln, kann der Benutzer via GUI die Anzahl der tatsächlichen Simulationsschritte wählen, die pro Visualisierung ausgeführt werden. Die Visualisierungsmechanismen der *SwarmWorld* sind so implementiert, dass maximal 60 Visualisierungen pro Sekunde durchgeführt werden. Die simulierte Welt kann auf zwei verschiedene Arten visualisiert werden.

Physikalischer Visualisierungsmodus: Visualisiert werden die physikalischen Körper der Simulation – nämlich diejenigen der Schwärmer, der passiven Objekte, der Sensoren und Emitter – farblich differenziert anhand ihrer verschiedenen Sensorikdomänen (rechter Teil der Abb. 5.8 auf der folgenden Seite). Sämtliche Visualisierungsmerkmale wie z.B. einzelne Sensorikdomänen sind separat zu- und abschaltbar. Zusätzlich einblendbar sind verschiedene Diagnosedaten, wie z.B. die einzelnen Zellen des QuadTrees der Kollisionsdetektion

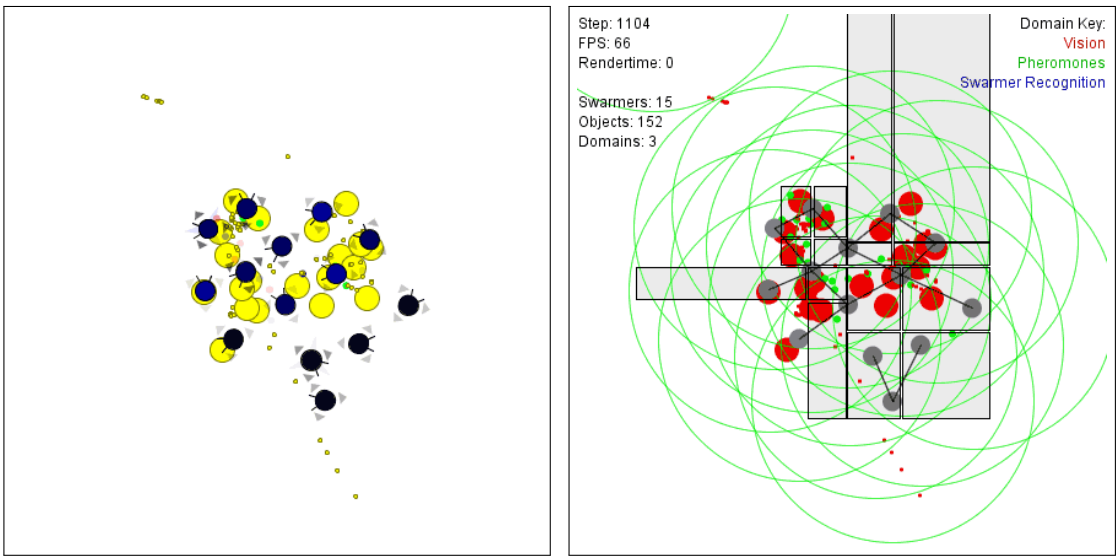


Abbildung 5.8: Benutzerdefinierter Visualisierungsmodus (links) und physikalischer Visualisierungsmodus (rechts), jeweils angewendet auf die gleiche Situation mit Schwärmern und Futterobjekten (im benutzerdefinierten Modus schwarz bzw. gelb). Der benutzerdefinierte Modus blendet hier verschiedene Dinge ein, die im Rahmen der simulierten Physik keine Rolle spielen (z.B. Fühler und sonstige Visualisierungen um die Schwärmer). Der physikalische Visualisierungsmodus „durchleuchtet“ die Welt anhand der physikalischen Gegebenheiten und macht es möglich, technische Zusatzinformationen einzublenden. Durch die eingeblendete Legende oben rechts und die automatisch kontrastreich gewählten Farben ist intuitiv erkennbar, welche Objekte welcher Sensorikdomäne zugehörig sind. Eingeblendete Zusatzinformationen sind in diesem Fall technische Informationen (oben links), Sensorikbereiche der Domäne „Pheromones“ (grüne Kreislinien), Zellen des Kollisions-QuadTrees sowie der Kollisionsgraph.

oder ein Kollisionsgraph, der physikalische Körper mit Liniensegmenten verbindet, falls sie miteinander in der Narrow Phase auf Kollision geprüft werden. Der physikalische Visualisierungsmodus ist fest in die Software einprogrammiert. Der Benutzer kann ihn also direkt nutzen, ohne sich um Visualisierungen Gedanken machen zu müssen, was die Effizienz im Experimentdesign erhöht.

Benutzerdefinierter Visualisierungsmodus: Der Benutzer hat die Möglichkeit, mittels Operatoren auf Schwärmern und passiven Objekten subjektive geometrische Formen und deren Umrandungs- und Füllfarben zu definieren, welche gezeichnet werden sollen. Diese Formen werden dann – unabhängig davon, ob sie der wirklich simulierten Physik entsprechen – anhand der Position und Orientierung des Schwärmers bzw. passiven Objekts zu objektiven Entsprechungen transformiert und gezeichnet (rechter Teil der Abb. 5.8). Die Formen und Farben können sich auch während der Simulation ändern. So ist es möglich, das Aussehen der Visualisierung völlig frei zu gestalten. Zusätzlich kann sowohl für Schwärmer, als auch für passive Objekte mittels entsprechender Operatoren eine zweite Klasse solcher geometrischen Formen definiert werden, welche sich bei Bedarf der Visualisierung zuschalten lässt und für z.B. Hilfslinien verschiedenster Art genutzt werden kann. Es sind weiterhin Schnittstellen definiert, über welche jedes passive Objekt und jeder Schwärmer textuelle, spezifische Informationen zu z.B. internen Zuständen zur Verfügung stellen kann. Diese werden dargestellt, wenn man das jeweilige Objekt oder den Schwärmer mit Mausklick auswählt.

Ein weiteres Merkmal der GUI ist, dass Operatoren zur Verfügung gestellt werden, mit denen der Benutzer der Bedienungsoberfläche zur Simulationskontrolle auf einfache Weise weitere Elemente hinzufügen kann, welche sich auf die konkrete Simulation beziehen.

5.5 Implementierung des parametrisierbaren Schwärmers und seiner Umwelt

Der parametrisierte Schwärmer und seine Umwelt werden implementiert, indem die im letzten Abschnitt beschriebenen abstrakten Bestandteile der Schwarmsimulationssoftware konkretisiert werden. Bei dieser Konkretisierung wurde die in Kapitel 4 beschriebene Modellierung verfolgt. Sämtliche angegebenen absoluten Werte in diesem Abschnitt beziehen sich auf beliebig definierbare und skalierbare Längeneinheiten.

5.5.1 Implementierung des Schwärmers

Körper

Der physikalische Körper ist kreisförmig mit einem Radius von 8. Dieser Radius hat sich in Simulation und Visualisierung bewährt.

Sensorik und interne Datenstrukturen

Wie in der Modellierung schon vorgestellt, besitzt der parametrisierbare Schwärmer verschiedene Sensoren, welche über die Sensorikparameter α, β, γ definiert werden. Diese Sensoren konkretisieren jeweils den durch die *SwarmWorld* vorgegebenen abstrakten Sensor und werden einem Schwärmer über die durch den abstrakten Schwärmer vorgegebenen Operatoren hinzugefügt. Sie sind jeweils einer eigenen Sensorikdomäne zugeordnet: „Vision“ im Falle des Sehsinns, mit dem z.B. Futter oder die Fernkommunikation anderer Schwärmer wahrgenommen wird. „Swarmer Recogition“ im Falle des Sensors für Schwärmerwahrnehmung und Nahbereichskommunikation, und „Pheromones“ im Falle des Sensors zur Pheromonwahrnehmung. Die Arbeitsweise aller drei Sensoren beinhaltet folgende Aspekte:

Drei Farbkanäle: Jeder Sensor hat drei Ausgabekanäle, die voneinander unabhängig sind, die aber als Farbkanäle interpretiert werden (Rot, Grün und Blau), was sich auch aussagekräftig visualisieren lässt. Ausnahme: Der Sensor der „Swarmer Recognition“ generiert zusätzlich zu den drei Farbkanälen einen analog gebildeten, vierten Wert zur reinen Schwärmerwahrnehmung, ohne Rücksichtnahme auf eventuelle Nahbereichskommunikation. Dieser Wert besteht allein aus dem nachfolgend noch definierten Faktor $q(x, d)$. Alle Sensoren haben also eine numerische, vektorielle Ausgabe und erwarten ebensolche Eingaben von den Emittern. Was für Eingaben Emitter genau liefern, wird im Abschnitt 5.5.1 behandelt.

Quadratischer Abfall mit wachsender Distanz: Je weiter ein Emitter (bezogen auf seine von *Phys2D* gemessene Distanz[6]) von der Position des Schwärmers entfernt ist, desto schwächer wird der resultierende Sensorwert. Der Sensorwert sinkt quadratisch mit dem Wachsen der Distanz: Sei x die Sichtweite des Sensors und d die Distanz zwischen Schwärmer und Emitter. Dann wird der resultierende Sensorwert mit dem Faktor

$$q(x,d) = \begin{cases} \left(\frac{x-d}{x}\right)^2 & \text{falls } d \leq x \\ 0 & \text{falls } d > x \end{cases}$$

multipliziert. Offensichtlich gilt $q(x, d) = 0$ für $x = d$, $q(x, d) = 1$ für $x = 0$, und für $0 < d < x$ fällt $q(x, d)$ in Richtung x auf einem Parabelbogen ab. Jeglicher Sensorwert wird also mit zunehmender Distanz zwischen Schwärmer und Emitter quadratisch schwächer, bis er an der Grenze des Sichtbereiches des Sensors ganz verschwunden ist. Der Fall $d > x$ wird in $q(x, d)$ betrachtet, um Messungenauigkeiten zu verhindern: Da die Distanz von *Phys2D*

[6] Diese Distanz gibt nicht notwendigerweise die Distanz vom Schwärmer zum nächsten Punkt auf dem Emitter wieder, da als Positionen der Objekte deren Schwerpunkte herangezogen werden. Sie ist aber durch *Phys2D* direkt gegeben und damit weniger berechnungsintensiv. Die Emitter sind in den durchgeführten Simulationen jedoch hinreichend klein, so dass der Unterschied zwischen diesen beiden Distanzen gering ist.

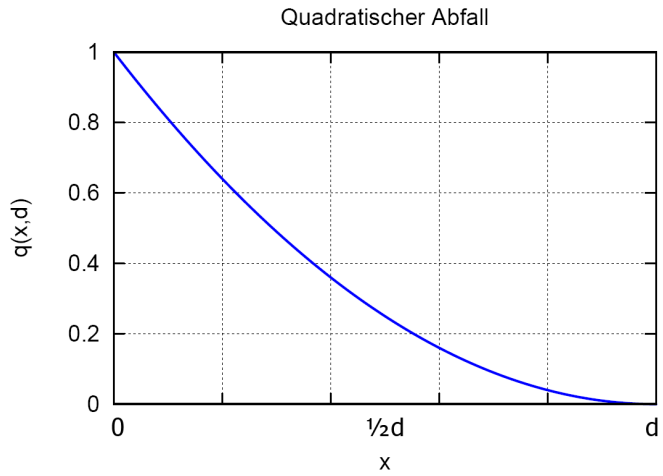


Abbildung 5.9: Plot des Faktors $q(x, d)$ für $0 \leq x \leq d$.

berechnet wird, kann es sein, dass ein Emitter den Sichtbereich eines Sensors berührt, also ein zugehöriger Sensorwert berechnet wird, die gemessene Distanz zum Sensor jedoch größer ist als x. In dem Fall würde der Sensorwert mit zunehmender Distanz zum Emitter kurz stärker werden, bis der Emitter nicht mehr wahrgenommen wird. Dies wird durch die explizite Betrachtung des Falls $d > x$ verhindert. Ein Plot des Faktors $q(x, d)$ befindet sich in Abb. 5.9.

Unterteilung und fließender Übergang: Sensoren sind in Zonen unterteilt, welche dem Schwärmer separate sensorische Eingaben liefern. Der „Vision“-Sensor teilt sich entlang seiner Winkelhalbierenden in zwei Zonen, als vereinfachte Sichtbereiche zweier Augen. Die beiden anderen Sensoren teilen sich zur Differenzierung der Reizrichtung gleichmäßig in vier Zonen mit Winkelhalbierenden bei $\phi, \phi + \frac{\pi}{2}, \phi + \pi, \phi + \frac{3\pi}{2}$, wobei ϕ die Orientierung des Schwärmers ist. Zwischen den vier Zonen besteht jeweils ein fließender Übergang: Befindet sich ein Emitter genau auf der Winkelhalbierenden einer Zone, so erhält diese Zone den vollen Anteil des ausgelösten Sensorwertes. Zwischen den Winkelhalbierenden einer und der nächsten Zone wird der Signalanteil, den jede der beiden beteiligten Zonen erhält, anhand des Winkels des Emitters zum Schwärmer linear interpoliert. Liegt der wahrgenommene Emitter relativ zum Schwärmer also beispielsweise exakt zwischen den Winkelhalbierenden zweier Zonen, so erhält jede Zone die Hälfte des resultierenden Sensorwertes. Zwischen den zwei Winkelhalbierenden der Zonen des „Vision“-Sensors wird auf dieselbe Weise linear interpoliert. Befindet sich ein vom „Vision“-Sensor wahrgenommenes Objekt außerhalb dieser beiden Winkelhalbierenden, so nimmt die Sensorzone, welche es berührt, den vollen Signalanteil wahr; die jeweils andere Zone nimmt nichts wahr.

Addition der Sensorwerte: Nimmt ein Sensor mehrere Emitter wahr, so werden die resultierenden Sensorwerte (je Zone) separat berechnet und dann addiert. Insofern sei bei den restlichen Aspekten jeweils die Wahrnehmung einzelner Emitter betrachtet.

Verrauschte Sensorik: Auf jeden resultierenden Sensorwert wird später (beschrieben im Rahmen der Funktionsweise der Kontrollstruktur) ein zufällig gleichverteiltes Rauschen addiert. Dies sei nur der Vollständigkeit halber an dieser Stelle angemerkt.

Zusammengefasst wird ein durch einen Emitter geliefertes *Roh-Signal* mit dem Faktor $q(x, d)$ multipliziert und anschließend auf Sensorzonen verteilt. So entstandene Sensorwerte werden für alle wahrgenommenen Objekte pro Sensorzone aufsummiert und die summierten Werte verrauscht an den Schwärmer weitergegeben.

Die interne Sensorik wird im Rahmen der Beschreibung der Arbeitsweise der Schwärmerkontrollstruktur behandelt.

Emitter und Aktorik

Die technische Realisierung des parametrisierbaren Schwärmers beinhaltet einen Schwärmer-Emitter in der Sensorikdomäne „Vision“ (emittiert die Fernkommunikation) und einen in der Sensorikdomäne „Swarmer Recognition“ (emittiert die Nahbereichskommunikation). Analog zur dreikanaligen Wahrnehmung der in diesen Sensorikdomänen arbeitenden Sensoren ist die Ausgabe dieser Emitter ebenfalls dreikanalig und als RGB-Farbe zu interpretieren.

Wie schon im Rahmen der Modellierung vorgestellt, wird die Aktorik von den sieben Aktorikparametern $\ell_f, \ell_n, t_p, f_{\max}, f_{\text{soz}}, s, t_s$ definiert. Die beiden genannten Emitter werden nur von den Parametern ℓ_f (Intensität der Fernkommunikation) sowie ℓ_n (Intensität der Nahbereichskommunikation) direkt beeinflusst.

Es kann nun die Art der Ausgabe der Emitter (das oben schon erwähnte *Roh-Signal*) genau definiert werden: Sei x die Ausgabe des „Vision“-Emitters und y die Ausgabe des „Swarmer Recognition“-Emitters. Dann gilt $x \in \ell_f \cdot [0;1]^3$ und $y \in \ell_n \cdot [0;1]^3$. Mit $[0;1]^3$ wird ein dreidimensionaler Vektor mit Komponenten im Intervall $[0;1]$ bezeichnet, der die RGB-Farbe definiert. Die konkreten Ausgabewerte eines jeden Emitters werden in jedem Simulationsschritt durch die Kontrollstruktur des jeweiligen Schwärmers neu festgelegt.

Bleibt noch die Realisierung der von den Aktorikparametern $t_p, f_{\max}, f_{\text{soz}}, s$ sowie t_s beeinflussten Aktorik zu behandeln. $f_{\max}$ definiert den maximalen Betrag des Vektors der von *Phys2D* auf den Schwärmerkörper aufzutragenden Fortbewegungskraft, die am Ende einer Kontrollstrukturausführung vom Schwärmer an die Simulationssoftware übergeben wird. Der Einfluss von t_p, f_{soz}, s und t_s und die restliche Aktorik wird während der Behandlung der Kontrollstruktur näher beschrieben.

Arbeitsweise der Kontrollstruktur eines Schwärmers

Wie im Rahmen der allgemeinen Beschreibung der Schwarmsimulationssoftware dargestellt, sind die sensorischen Eingaben bereits generiert, wenn die Kontrollstruktur eines Schwärmers ausgeführt wird. Die Konkretisierung der abstrakt gehaltenen Kontrollstruktur beinhaltet die Verwaltung einiger interner Zustände des Schwärmers sowie die Ausführung eines neuronalen Netzes zur Überführung der sensorischen Eingaben in Ausgaben zur Schwärmerkontrolle. Folgende Aspekte finden bei der Ausführung der Kontrollstruktur eines Schwärmers Beachtung:

Alter des Schwärmers: Mit jeder Ausführung der Schwärmerkontrollstruktur wird ein Wert inkrementiert, der das Alter des Schwärmers nachhält. Der Schwärmer stirbt nach 10.000 Ausführungen seiner Kontrollstruktur und wird dann aus der Schwarmwelt entfernt.

Nahrungsverbrauch: Der Inhalt des Nahrungsspeichers (dessen Speichergröße n ebenfalls ein Parameter ist, der bereits im Rahmen der Modellierung eingeführt wurde) wird mit jeder Ausführung der Kontrollstruktur dekrementiert. Liegt der Wert vor der Dekrementierung bei 0, so verhungert der Schwärmer und wird aus der Schwarmwelt entfernt. In diesem Schritt wird auch der interne Sensorikwert, welcher den Füllstand des internen Futterspeichers repräsentiert, generiert. Sei n_{current} der aktuelle Futtervorrat und n wie oben angegeben. Dann ist der interne Sensorikwert gegeben durch $\frac{n_{\text{current}}}{n}$. Der Schwärmer kann also gewissermaßen Hunger verspüren.

Sprintkondition regenerieren: Es gibt einen Sprintkonditionswert *currentSprintSteps*, welcher mit 0 initialisiert wird. Auf diesen nehmen die beiden Parameter s und t_s Einfluss. $\lfloor$*currentSprintSteps*$\rfloor$ repräsentiert die Anzahl der Schritte, die der Schwärmer aktuell „sprinten“ kann. Sprintet der Schwärmer, so wird der nach Kontrollstrukturausführung zurückgegebene Vektor der gewünschten Bewegung mit 2 multipliziert. Der Parameter t_s repräsentiert die Anzahl der Zeitschritte, die benötigt werden, um einen Sprintschritt zu regenerie-

Neuron(e)	Eingabe
0 – 5	Eingabe aus Sensor der Domäne „Vision“ (2 Zonen × 3 Farbkanäle)
6 – 17	Eingabe (Nahbereichskommunikation) aus Sensor der Domäne „Swarmer Recognition“ (4 Zonen × 3 Farbkanäle)
18 – 21	Eingabe (Schwärmerwahrnehmung) aus Sensor der Domäne „Swarmer Recognition“ (4 Zonen)
22 – 33	Eingabe aus Sensor der Domäne „Pheromones“ (4 Zonen × 3 Farbkanäle)
34	Eingabe aus der internen Sensorik (Füllstand Futtervorrat)

Tabelle 5.1: Tabelle der Eingabeneurone.

ren, also wird in dieser Regenerationsphase $\frac{1}{t_s}$ zu *currentSprintSteps* dazuaddiert. *currentSprintSteps* kann aber nicht größer werden als der Parameter s, der die maximale Anzahl Sprintschritte verwaltet.

Pheromonausschüttung regenerieren: Der Parameter t_p definiert die Anzahl Zeitschritte, die nach dem Ablegen eines Pheromons vergangen sein müssen, ehe das nächste Pheromon abgelegt werden darf. Es ist ein Pheromonregenerationswert *currentPheromoneWaitTime* implementiert, der mit 0 initialisiert und nach jedem Ablegen eines Pheromons auf t_p gesetzt wird. Er wird mit jeder Ausführung der Kontrollstruktur um 1 dekrementiert, kann aber 0 nicht unterschreiten. Gilt $currentPheromoneWaitTime = 0$, so kann wieder ein Pheromon gelegt werden.

Hinzufügen von Rauschen: Auf jede Eingabe in das neuronale Netz wird, wie oben schon angedeutet, ein jeweils zufällig gleichverteilt generiertes Rauschen $r \in [-\frac{1}{100}; \frac{1}{100}]$ addiert. Dies ist naturinspiriert und fördert die Generalisierungsfähigkeit entstehender Kontrollstrukturen.

Ausführung des neuronalen Netzes: Da nun die internen Zustände und Sensorikeingaben verwaltet sind, kann das kontrollierende neuronale Netz ausgeführt werden. Es hat 35 Eingabeneurone (Tabelle 5.1) und 17 Ausgabeneurone (Tabelle 5.2 auf der rechten Seite). Die Aktivierungsfunktion bedingt, dass jede Komponente des Ausgabevektors im offenen Intervall $]-1;1[$ liegt. Durch die 17 Ausgaben wird in den nun folgenden Schritten die Aktorik gesteuert.

Orientierungsänderung und Bewegung: Die Orientierungsänderungsausgabe (1) multipliziert mit $\frac{\pi}{4}$ repräsentiert die tatsächliche Orientierungsänderung des Schwärmers. Die Bewegungsausgabe (0) multipliziert mit $f_{\max}$ repräsentiert die Strecke, die der Schwärmer zurücklegen möchte. Beide Ausgaben können sowohl positiv als auch negativ sein, werden am Ende der Kontrollstrukturausführung an die Simulationssoftware übergeben und – umgewandelt in entsprechende physikalische Kräfte – auf den physikalischen Schwärmerkörper aufgetragen. Positive Werte von (0) repräsentieren eine Vorwärtsbewegung, negative eine Rückwärtsbewegung. Positive Werte von (1) repräsentieren eine Drehung entgegen des Uhrzeigersinns, negative eine Drehung im Uhrzeigersinn[7]. Auf Schwärmerkörper aufgetragene Kräfte werden dann, wie schon dargestellt, im physikalischen Simulationsschritt angewandt. Hierbei wird zunächst die Orientierung geändert und anschließend die Bewegung ausgeführt. Orientierungsänderungen stellen Rotationen um den Schwerpunkt des Schwärmerkörpers dar. Da der Körper des beschriebenen Schwärmers kreisförmig ist, handelt es sich hier also um eine Drehung um den Mittelpunkt des Körpers.

Soziale Interaktionskraft: Sei s_1 der hier betrachtete Schwärmer, dessen Kontrollstruktur ausgeführt wird. Für jeden von s_1 wahrgenommenen Schwärmer s_2, mit jeweiligem Abstand

[7] Dies wird durch das Physikframework fest vorgegeben.

Neuron(e)	Ausgabe
0	Steuerung der Bewegung. Positive Werte repräsentieren eine Vorwärtsbewegung, negative eine Rückwärtsbewegung.
1	Steuerung der Orientierungsänderung. Positive Werte repräsentieren eine Drehung entgegen des Uhrzeigersinns, negative eine Drehung im Uhrzeigersinn.
2	Steuerung der sozialen Interaktionskraft
3	Schalter, der angibt, ob gesprintet werden soll
4 – 6	Steuerung der Fernkommunikation (3 Farbkanäle)
7 – 9	Steuerung der Nahbereichskommunikation (3 Farbkanäle)
10 – 12	Steuerung der Pheromonfarbe, falls Pheromon abgelegt wird (3 Farbkanäle)
13	Schalter, der angibt, ob Pheromon abgelegt werden soll
14	Schalter, der angibt, ob Nahrung aufgenommen werden soll
15	Schalter, der angibt, ob Nahrung abgegeben werden soll
16	Schalter, der angibt, ob Reproduktion erfolgen soll

Tabelle 5.2: Tabelle der Ausgabeneurone. Ausgaben, welche als Schalter bezeichnet werden, sind aktiviert, wenn der Ausgabewert positiv ist; ansonsten nicht.

d zu s_1, wird die Generatorfunktion soz(d) für die soziale Interaktionskraft (in Anhangsabschnitt A.1 erläutert) mit der diesbezüglichen Ausgabe (2) und dem Parameter f_{soz} multipliziert. Dies ergibt die Strecke, die s_2 sich – nach dem Willen von s_1 – auf s_1 tendenziell zubewegen soll (negative Werte des Produktes von soz(d) mit der Ausgabe (2)), oder sich tendenziell von ihm wegbewegen soll (positive Werte des Produktes). Bedingt durch den Ausgabebereich $]-1;1[$ können andere Schwärmer abgestoßen oder angezogen werden. Die Richtung des resultierenden Kraftvektors ist durch die Position von s_1 zu s_2 zueinander gegeben und dadurch, ob Anziehung oder Abstoßung vorliegt. Auf s_2 wird die entsprechende Kraft aufgetragen und im physikalischen Simulationsschritt zusammen mit anderen aufgetragenen Kräften (wie z.B. s_2-eigenen Bewegungskräften) weiter behandelt. Da soz(d) hier einen Wertebereich von ca. $[-1;0,4]$ besitzt, liegt der Betrag der hier induzierten Bewegung von s_2 im Intervall $[-f_{\text{soz}};0,4 \cdot f_{\text{soz}}]$.

Sprint: Ist die Sprintausgabe (3) > 0, so gilt der aktuelle Schritt als Sprintschritt und die Kraft, welche vom Physikframework auf den Schwärmerkörper aufgetragen wird, verdoppelt sich – ansonsten nicht. Dies ist nur dann möglich, wenn gilt *currentSprintSteps* > 1. In diesem Fall wird *currentSprintSteps* um 1 dekrementiert.

Fernkommunikation: Der Fernkommunikationsemitter (Sensorikdomäne „Vision“) wird auf den Farbwert gesetzt, welcher aus den diesbezüglichen Ausgaben entnommen wird (4,5,6). Negative Ausgabewerte werden durch 0 ersetzt. Der resultierende, als RGB-Farbe interpretierte Vektor wird mit der Fernkommunikationsintensität ℓ_f multipliziert.

Nahbereichskommunikation: Der „Swarmer Recognition“-Emitter wird auf den Farbwert gesetzt, welcher aus den diesbezüglichen Ausgaben entnommen wird (7,8,9). Negative Ausgabewerte werden durch 0 ersetzt. Der resultierende, als RGB-Farbe interpretierte Vektor wird mit der Nahbereichskommunikationsintensität ℓ_n multipliziert.

Pheromon ablegen: Ist die diesbezügliche Ausgabe (13) > 0, so wird, falls gilt *currentPheromoneWaitTime* $= 0$, ein Pheromon abgelegt. Die drei Farbkanäle des Pheromons werden mit den Ausgaben (10,11,12) initialisiert, wobei analog zu vorherigen Farbausgaben negative Werte durch null ersetzt werden. Nach Ablage eines Pheromons wird *currentPheromoneWaitTime* mit t_p überschrieben.

Nahrungsaufnahme: Ist Ausgabe (14) > 0, so wird – so viel wie in den Futterspeicher passt – von einem Futterobjekt aufgenommen, sofern sich eines in unmittelbarer Nähe zur

Schwärmerfront befindet. Der Futterwert des Futterobjektes wird um die aufgenommene Menge reduziert und das Futterobjekt – wenn verbraucht – aus der Welt entfernt.

Nahrungsabgabe: Ist Ausgabe (15) > 0, so wird der Welt ein kleines Futterobjekt im Wert von 100 Futtereinheiten an der Position des Schwärmers hinzugefügt und der interne Futterspeicher um die entsprechende Menge dekrementiert. Dies funktioniert nur, wenn der Schwärmer noch soviel Futter besitzt.

Vermehrung: Ist Ausgabe (16) > 0, so versucht der Schwärmer sich zu vermehren. Dies funktioniert nur, wenn der Futterspeicher mindestens halbvoll ist. Im Vermehrungsfall wird das Futter im Futterspeicher gedrittelt: Ein Drittel verbleibt beim aktuellen Schwärmer, eines erhält der neue Schwärmer und das letzte Drittel wird im Rahmen des Vermehrungsprozesses verbraucht. Der neue Schwärmer, ein exakter Klon des Vaterschwärmers, wird der Schwarmwelt in unmittelbarer Nähe des Vaterschwärmers hinzugefügt. Sein (ebenfalls geklontes) neuronales Netz wird mit zufälligen Aktivierungen im Intervall $[-0,5; 0,5]$ initialisiert.

Definition von Parametern durch Metaparameter

Während der bisherigen Beschreibung des parametrisierbaren Schwärmers wurde auf die 11 Parameter $n, \ell_f, \ell_n, t_p, f_{\max}, f_{\text{soz}}, s, t_s, \alpha, \beta, \gamma$ zurückgegriffen. Wie in der Modellierung beschrieben, können diese nicht völlig unabhängig voneinander gesetzt werden, damit kein „Superschwärmer“ mit allen Parametern auf dem Bestwert entstehen kann. Es wird nur von endlich vielen Fertigkeitenressourcen ausgegangen. Im Rahmen der Modellierung wurde beschrieben, dass dieses Prinzip über eine feste Anzahl Fertigkeitenpunkte (ähnlich wie in einem Rollenspiel) realisiert werden soll, welche auf die einzelnen Fertigkeiten verteilt werden. Hierbei soll die Anzahl der zur Verfügung stehenden Fertigkeitenpunkte signifikant geringer sein, als es nötig wäre, um alle Fertigkeiten voll auszubauen.

In den später beschriebenen Experimenten liegt die Anzahl der zur Verfügung stehenden Fertigkeitenpunkte, soweit nicht anders angegeben, bei 300. In eine jede Fertigkeit können 0 bis 100 Punkte investiert werden. Die 11 Parameter werden über 10 sog. Metaparameter definiert, wobei die Metaparameter die Anzahl der Fertigkeitenpunkte beinhalten. Die Zahl 10 kommt zustande, da die beiden Parameter t_s und s beide die Sprintfertigkeit betreffen und daher von einem einzigen Sprint-Metaparameter definiert werden.

Die konkrete Definition der Parameter funktioniert über Optima und Pessima für die Parameter: Liegt ein zu einem Parameter gehöriger Metaparameter bei 100, so wird für den Parameter sein Optimum eingesetzt. Liegt der Metaparameter bei 0, so wird das Pessimum eingesetzt. Liegt der Metaparameter zwischen 0 und 100, so wird linear zwischen Pessimum und Optimum interpoliert. In Tabelle 5.3 auf der rechten Seite finden sich die Optima und Pessima für alle Schwärmerparameter.

Evolutionsoperatoren auf Schwärmerebene

Der Satz von Metaparametern sowie das neuronale Netz machen das Genom eines Schwärmers aus, auf welches der evolutionäre Algorithmus Einfluss nimmt. Es sind also Mutations- und Kreuzungsoperatoren auf diesem Genom zu definieren.

Mutation: Ein Schwärmer wird mutiert, indem entweder sein neuronales Netz mutiert wird, oder indem sein Satz Metaparameter mutiert wird. Die Art der Mutation wird zufällig, aber nicht gleichverteilt ermittelt: Aufgrund der i.d.R. wesentlich höheren Parameteranzahl des neuronalen Netzes gegenüber dem Metaparametersatz wird mit 75% Wahrscheinlichkeit das neuronale Netz und mit 25% Wahrscheinlichkeit der Metaparametersatz mutiert. Das neuronale Netz wird unter Benutzung der in Abschnitt 5.3.7 beschriebenen Mutationsoperatoren mutiert. Der Metaparametersatz wird mutiert, indem zwei unterschiedliche Metaparameter zufällig gleichverteilt gewählt werden und ein ebenfalls zufällig gleichverteilt

Parameter	Pessimum	Optimum
Futterspeicher n	2000	6000
Fernkommunikations-Intensität ℓ_f	1	10
Nahbereichskommunikations-Intensität ℓ_n	1	10
Pheromon-Regenerationszeit t_p	1000	200
Maximale Schrittweite $f_{\max}$	1	6
Maximale soziale Interaktionskraft f_{soz}	0.5	4
Anzahl der Sprintschritte s	1	10
Regenerationszeit pro Sprintschritt t_s	2000	200
Wahrnehmungsradius „Vision“ α	100	400
Wahrnehmungsradius „Swarmer Recognition“ β	50	150
Wahrnehmungsradius „Pheromones“ γ	75	250

Tabelle 5.3: Tabelle der Pessima und Optima der durch Metaparameter gesteuerten konkreten Parameter.

gewählter Wert $1 \leq x \leq 3$ von einem Metaparameter subtrahiert und auf den anderen addiert wird. Die Mutation ist so implementiert, dass kein Metaparameter kleiner als 0 oder größer als 100 werden kann.

Kreuzung: Der Kreuzungsoperator kreuzt zwei parametrisierbare Schwärmer s_1 und s_2 zu einem neuen, hier s_3 genannt. Dies kann auf folgende vier Arten geschehen, von denen eine zufällig gleichverteilt gewählt wird:

1. s_3 erhält das neuronale Netz von s_2 und die Metaparameter von s_1.
2. s_3 erhält das neuronale Netz von s_1 und die Metaparameter von s_2.
3. s_3 erhält ein neuronales Netz, was aus der Kreuzung der Netze s_1 und s_2 ensteht, sowie zufällig gleichverteilt einen der Metaparametersätze von s_1 und s_2. Dies ist die einzige Stelle, wo tatsächlich neuronale Netze gekreuzt werden. Die Kreuzung von neuronalen Netzen wird also aus Gründen, die in Abschnitt 5.3.7 schon skizziert wurden, verhältnismäßig selten angewandt.
4. s_3 erhält zufällig gleichverteilt eines der neuronalen Netze von s_1 oder s_2. Gleichzeitig wird für s_3 ein neuer Metaparametersatz aus denjenigen von s_1 und s_2 gebildet, indem jeder Metaparameter zufällig gleichverteilt von s_1 oder s_2 übernommen wird. Für jeden Metaparameter des neuen Satzes gilt dann $0 \leq x \leq 100$, die Summe der Metaparameter muss aber nicht gleich der verfügbaren Fertigkeitenpunkte sein. Also werden in einem zweiten Schritt der Kreuzung einzelne Fertigkeitenpunkte zufällig gleichverteilt von jedem neuen Metaparameter abgezogen bzw. zu ihm hinzuaddiert, bis die Summe der Metaparameter gleich der verfügbaren Fertigkeitenpunkte ist. Dieser Schritt ist so implementiert, dass kein Metaparameter kleiner als 0 oder größer als 100 werden kann.

Evolutionsoperatoren auf Schwarmebene

Wie im Rahmen der Modellierung (Kapitel 4) schon beschrieben, wird Heterogenität von Schwärmen ermöglicht, indem nicht ein Schwärmer evolviert wird, sondern ein Schwärmertripel. Da sich jeder Schwärmer exakt klonen und so vermehren kann, können so drei unterschiedliche, aber in sich homogene Subschwärme entstehen. Das Genom eines Schwärmertripels besteht aus drei Schwärmergenomen. Auf einem Schwärmertripel sind ebenfalls Evolutionsoperatoren zu definieren. Da auf der Ebene der Schwärmertripel die Evolution stattfindet, ist hier ebenfalls eine Fitnessfunktion zu definieren.

Mutation: Für die Mutation eines Schwärmertripels werden zufällig gleichverteilt ein oder zwei Schwärmer mittels der oben beschriebenen Mutation auf Schwärmerebene mutiert.

Kreuzung: Für die Kreuzung eines Schwärmertripels aus zwei gegebenen wird zufällig gleichverteilt eine der beiden folgenden Kreuzungsarten gewählt und ausgeführt:

1. Aus den sechs Schwärmern der zwei gegebenen Schwärmertripel wird zufällig gleichverteilt und wiederholungsfrei ein neues Schwärmertripel gebildet.
2. Zufällig gleichverteilt wird eines der beiden gegebenen Schwärmertripel t gewählt. Das gekreuzte Schwärmertripel ensteht, indem ein zufällig gleichverteilt gewählter Schwärmer s_t aus t mit einem ebenso gewählten Schwärmer des anderen Tripels gekreuzt und das Resultat der Kreuzung für s_t eingesetzt wird.

5.5.2 Passive Objekte in der Umwelt

Es sind zwei Arten von Objekten in der Umwelt vorgesehen: Futter und Pheromone.

Futterobjekte befinden sich als Emitter in der Sensorikdomäne „Vision“, werden also vom Sehsinn wahrgenommen. Sie besitzen einen internen Futterwert, welcher die Menge an noch nicht vertilgtem Futter repräsentiert. Mit jedem Schwärmer, der Nahrung von dem Futterobjekt zu sich nimmt, nimmt der interne Futterwert um die gleiche Nahrungsmenge ab. Ist der interne Futterwert bei 0 angelangt, so ist das Futterobjekt aufgebraucht und verschwindet aus der Schwarmwelt. Futterobjekte als Emitter liefern ein *gelbes* RGB-Rohsignal $s = i \cdot (1, 1, 0)$ (Rot- und Grünkanal sind aktiviert). Hierbei ist die Intensität i ebenfalls abhängig vom initialen Futterwert des Futterobjektes[8]. Futterobjekte verschwinden (wenn nicht schon vorher aufgebraucht) nach 10.000 Zeitschritten aus der Welt. Dies ist motiviert durch natürliche Zersetzung.

Pheromonobjekte sind ebenfalls kreisrund und liefern als Emitter ein RGB-Rohsignal $s \in [0; 1]^3$ (repräsentiert durch den Farbwert, der dem jeweiligen Pheromonobjekt beim Ablegen zugewiesen wurde). Pheromone verschwinden nach 2.000 Zeitschritten aus der Welt, ebenfalls durch natürliche Zersetzung motiviert.

5.5.3 Visualisierung der Schwärmer, ihrer Merkmale und ihrer Umwelt

Bleibt noch darzustellen – insbesondere im Hinblick auf spätere Abbildungen – auf welche Weise ein parametrisierbarer Schwärmer visualisiert wird. Es wurde schon beschrieben, dass die Visualisierung eines Schwärmers aus zwei Klassen geometrischer Formen besteht: Aus solchen, die standardmäßig dargestellt werden und anderen, welche separat zuschaltbar sind. Die standardmäßig dargestellten stellen den Schwärmerkörper dar, sowie mehrere Fühler. Über die Anzahl der Fühler (1, 2 oder 3) wird die Zugehörigkeit eines Schwärmers zu Subschwarm 1, 2 oder 3 visualisiert, sowie die Orientierung des Schwärmers. Als zuschaltbare Formen werden die sensorischen Bereiche des Schwärmers dargestellt.

Fernkommunikation wird durch Strahlen um den Schwärmerkörper dargestellt (Teil (a) der Abb. 5.10 auf der rechten Seite). Die Nahbereichskommunikation wird, wie schon beschrieben, als RGB-Farbe interpretiert, mit welcher der Schwärmerkörper eingefärbt wird (Abb. 5.10, Teil (b)). Von Schwärmern gelegte Pheromone werden durch kleine farbige Kreise dargestellt, entsprechend ihres RGB-Wertes (Abb. 5.10, Teil (c)). Soziale Interaktionskräfte werden, falls sie anziehend wirken, durch Pfeile in Richtung des ausübenden Schwärmers dargestellt. Falls sie abstoßend wirken, zeigen die Pfeile vom Schwärmer weg. Je größer der Betrag der Aktivierung des zugehörigen Ausgabeneurons ist, desto intensiver sind die Pfeile dargestellt (Abb. 5.10, Teil (d)). Futterobjekte werden – wie schon in Abbildung 5.8 auf Seite 60 dargestellt – gelb, kreisrund und dunkelgelb umrandet dargestellt, mit variablen Größen entsprechend ihres initialen Futtergehaltes.

[8] Details der Errechnung der Signalintensität und der physikalischen Größe des Futters würden hier zu weit in die Details führen.

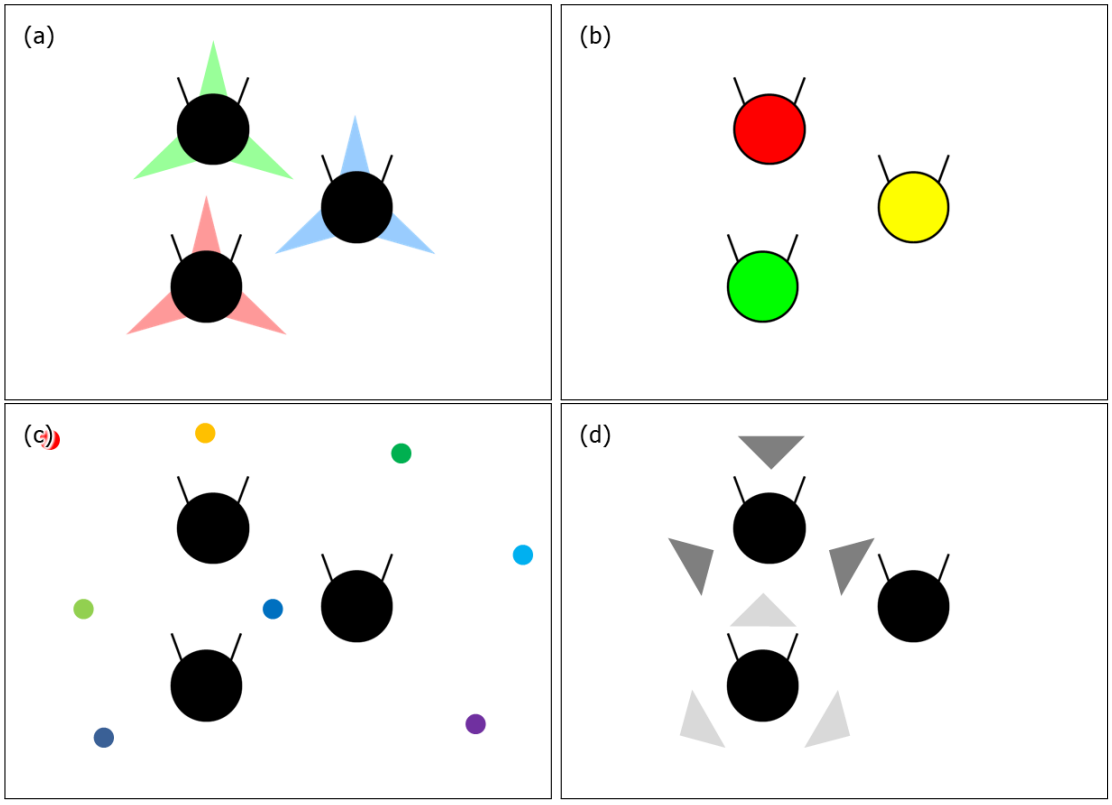


Abbildung 5.10: Visualisierung verschiedener Merkmale eines Schwärmers: (a) Visualisierung der Fernkommunikation. (b) Visualisierung der Nahbereichskommunikation. (c) Visualisierung von Pheromonen, wie sie von Schwärmern gelegt werden können. (d) Visualisierung von sozialen Interaktionskräften verschiedener Intensität, sowohl anziehend als auch abstoßend, wie sie von Schwärmern ausgeübt werden können.

5.6 Implementierung der Testsuite für fertig evolvierte Schwärme

Um bereits fertig evolvierte Schwärme in gegebenen Umwelten teilweise automatisiert zu analysieren und insbesondere besonders interessante Schwärme zu identifizieren, ohne jeden einzelnen in der Simulation beobachten zu müssen, wurde eine automatisierte Testsuite entwickelt. Diese soll sowohl Anhaltspunkte für die Wichtigkeit einzelner Subschwärme, als auch für die Wichtigkeit einzelner Fertigkeiten jedes Subschwarms geben. Hierzu muss zunächst definiert werden, was ein Test und was Wichtigkeit ist.

5.6.1 Tests, Referenzperformanz und Wichtigkeit

Als „Test“ wird im Folgenden eine Menge von 20 Evaluationen eines gegebenen Schwarms und einer gegebenen Umwelt unter bestimmten Bedingungen bezeichnet. Über die resultierenden Fitnesswerte dieser Evaluationen wird zu Vergleichszwecken das arithmetische Mittel gebildet. Dies ist nötig um Schwankungen auszugleichen, die solch natürlich motivierten Experimenten zueigen sind. Die Testsuite führt zunächst einen Referenztest aus, in dem der gegebene Schwarm ohne Veränderungen getestet wird. Dadurch wird ein Referenzperformanzwert gebildet.

Eine Eigenschaft wird als wichtig bezeichnet, wenn die Performanz eines Schwarms bei Fehlen dieser Eigenschaft signifikant einbricht. Formaler: Seien t, r Tests, wobei r der oben erwähnte Referenztest ist. Sei weiter $p(t)$ die Performanz eines Tests t, $p(r)$ also die Referenzperformanz, $p(t), p(r) > 0$. $p(t)$ repräsentiert das arithmetische Mittel der Fitnesswerte aller in t enthaltenen Einzelevaluationen. Sei e eine (wie auch immer geartete) Eigenschaft, und t_e der Test, in welchem diese Eigenschaft zwecks Wichtigkeitsmessung deaktiviert wurde. Die Wichtigkeit W_e einer Eigenschaft e bezüglich eines Experimentes ist dann definiert zu

$$W_e = 1 - \frac{p(e)}{p(r)}.$$

Unter der Annahme, dass $0 \leq p(e) \leq p(r)$ gilt, ist offensichtlich $0 \leq W_e \leq 1$. Ist W_e nahe 1, so war e wichtig für die Performanz, beim Fehlen von e bricht die Performanz ein. Ist W_e nahe 0, so gab es keine signifikanten Veränderungen in der Performanz durch das Deaktivieren von e.

5.6.2 Auf Wichtigkeit getestete Eigenschaften

Getestete Eigenschaften sind zum einen die drei Subschwärme eines Schwarms. Jeder Subschwarm wird also einmal deaktiviert und dann die Performanz gemessen. Weiter werden verschiedene Fertigkeiten separat in jedem Subschwarm deaktiviert und so auf Wichtigkeit getestet:

Soziale Interaktionskraft: Die soziale Interaktionskraft des betreffenden Subschwarms wird deaktiviert.

Nahbereichskommunikation: Emitter und Sensoren für die Nahbereichskommunikation des betreffenden Subschwarms werden deaktiviert.

Fernkommunikation: Emitter für die Fernkommunikation des betreffenden Subschwarms werden deaktiviert, jedoch nicht die entsprechenden Sensoren (diese sind zwangsweise erforderlich, um Futter wahrzunehmen).

Stigmergie: Die Fähigkeit, Pheromone abzulegen und wahrzunehmen, wird für den betreffenden Subschwarm deaktiviert.

Schwärmerwahrnehmung: Die Fähigkeit, die Anwesenheit anderer Schwärmer wahrzunehmen (nicht zu verwechseln mit der Wahrnehmung von Nahbereichskommunikation), wird für den betreffenden Subschwarm deaktiviert.

Futter abgeben: Schwärmer des betreffenden Subschwarms können kein Futter mehr abgeben.

Sprintfähigkeit: Schwärmer des betreffenden Subschwarms können nicht sprinten.

Es ergeben sich so 25 Tests pro evolviertem Schwärmertripel, also 500 Re-Evaluationen unter verschiedenen Bedingungen. Anhand der Wichtigkeit von Subschwärmen und Eigenschaften pro Subschwarm kann ermessen werden, welcher Subschwarm und welche Eigenschaft überhaupt in die weitere Analyse und Analyse-Illustration des getesteten Schwarms mit einbezogen werden müssen.

5.6.3 Automatisierte Illustration und Korrelationsplots

Nach der so durchgeführten automatisierten Analyse eines Schwarms werden die Resultate automatisiert graphisch aufbereitet: Zum einen wird eine Textdatei geschrieben, welche direkt in eine vorbereitete Datei von Microsoft Excel einkopiert werden kann. Excel stellt dann die Metaparameter der Subschwärme sowie die Subschwarmwichtigkeiten und Fertigkeitenwichtigkeiten pro Subschwarm graphisch dar (Abb. 5.11 auf der folgenden Seite). Die Farben Rot, Grün und Blau stehen in allen mittels der Testsuite generierten Abbildungen für die Subschwärme 1, 2 und 3.

Als weitergehende graphische Aufbereitung und für genauere Verhaltensanalysen im nachfolgenden Kapitel werden Korrelationsplots für die Ein- und Ausgaben der Kontrollstrukturen der Subschwärme erstellt. Ein Korrelationsplot hat den Definitionsbereich einer der 35 Eingaben als x-Achse, sowie den relevanten[9] Wertebereich einer der 17 Ausgaben als y-Achse. Wird ein Korrelationsplot erstellt, so enthält er für alle Subschwärme mit einer Wichtigkeit von mindestens $\frac{1}{5}$ eine repräsentative Auswahl von Datenpunkten der jeweiligen Ein- und Ausgabe. Diese wird mittels *Reservoir Sampling* [63] über den Referenztest gebildet. Ein Korrelationsplot wird erstellt für Kombinationen aus Ein- und Ausgaben, deren oben dargestellte, deaktivierbare Eigenschaften bei mindestens einem Subschwarm eine Wichtigkeit von mindestens $\frac{1}{5}$ ergeben haben[10]. Diese maximal 595 Plots[11] werden automatisiert aus der Testsuite heraus mit Gnuplot erstellt, so dass direkt druckfertige PDF-Dateien entstehen. Diese kann man vorab als Miniaturansichten betrachten (Abb. 5.12 auf Seite 73) und so direkt Korrelationen zwischen Ein- und Ausgaben ausmachen. Später

Die auf diese Weise visuell ansprechend aufbereiteten Daten sind sehr hilfreich bei der Interpretation entstandenen Schwarmverhaltens, aber natürlich nicht unfehlbar. Eine offensichtliche Schwäche ist, dass zwei Eigenschaften e_1 und e_2, welche gegenseitig redundant sind, nicht als wichtig erkannt werden. Übernimmt e_1 komplett die Aufgabe einer zum Test entfernten Eigenschaft e_2, so wird e_2 nicht als wichtig erkannt. Geschieht dies auch umgekehrt – gleicht also e_2 das Fehlen von e_1 aus – so wird keine der beiden Eigenschaften e_1 und e_2 als wichtig erkannt, obwohl eine der beiden Eigenschaften zum Erhalten der Performanz vorhanden sein sollte. Dieser Effekt wurde insbesondere beobachtet bei zwei Subschwärmen, die jeweils alleine lebensfähig sind – nach Aussage der Testsuite ist dann *kein* Subschwarm wichtig. Insofern ist die Testsuite äußerst nützlich und hilft herauszufinden, worauf man achten muss – benötigt aber etwas Übung bei der Ergebnisinterpretation.

5.7 Prinzip der Experimentdurchführung

Zum Schluss dieses Kapitels soll in diesem Abschnitt dargestellt werden, wie ein einzelnes Experiment aufgebaut ist und durchgeführt wird.

[9] Negative Ausgaben bei einem Schalter-Ausgabeneuron sind irrelevant – vgl. Tabelle 5.2 auf Seite 65.

[10] Hier ist zu beachten, dass neben der Wahrnehmung der Fernkommunikation auch einige andere Eigenschaften der Schwärmer nicht zum Test deaktiviert wurden (z.B. Reproduktion, Fortbewegung, Orientierungsänderung, Futteraufnahme), da die Performanz ohne diese ganz offensichtlich einbrechen würde. Diese Eigenschaften wurden von vornherein als „Wichtig“ eingestuft und fanden so in den Korrelationsplots Berücksichtigung.

[11] Wird jedes der 35 Eingabeneurone gegen jedes der 17 Ausgabeneurone auf die beschriebene Weise geplottet, ergeben sich 595 Plots

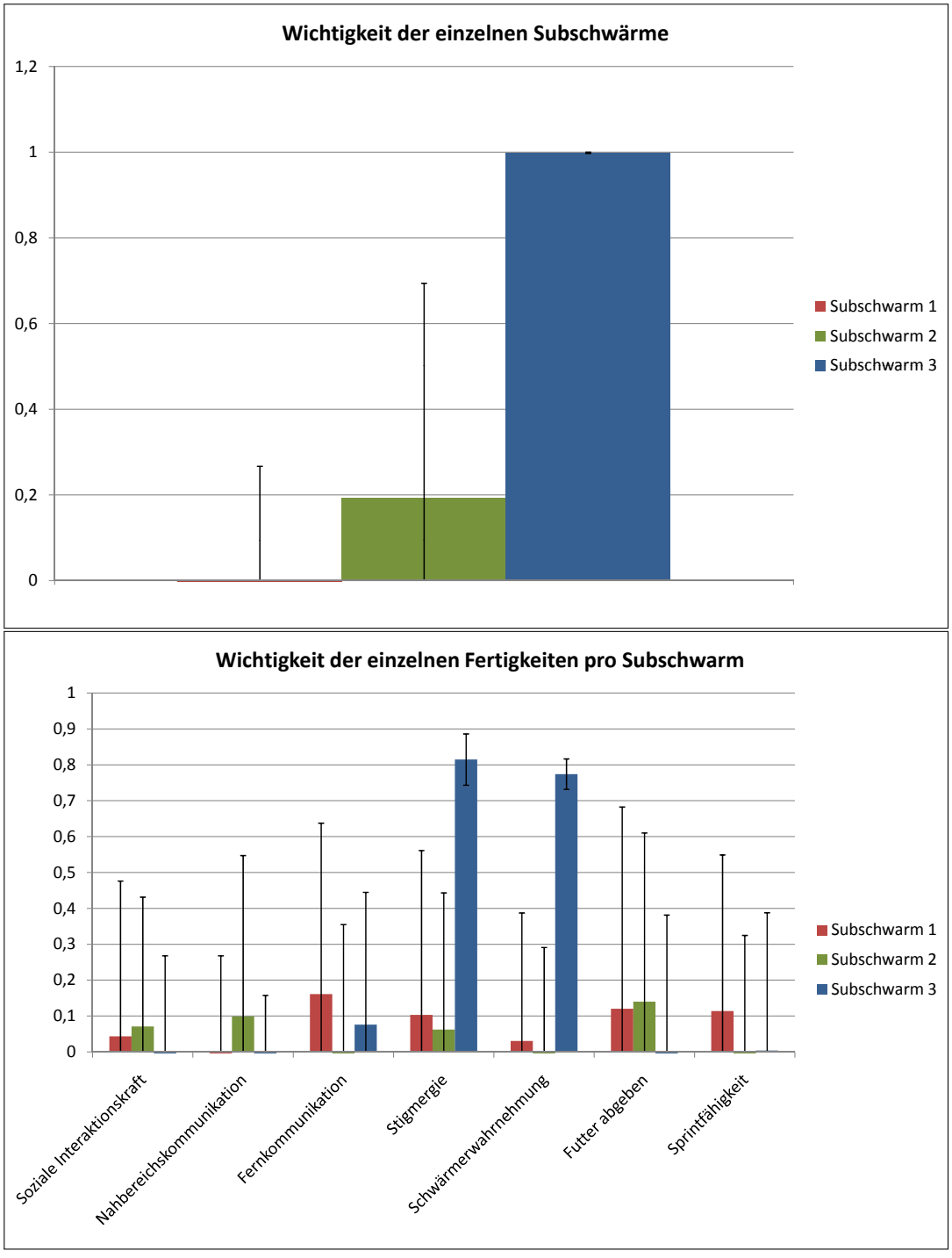


Abbildung 5.11: Wichtigkeitendiagramme in Bezug auf Subschwärme (oben) und Fertigkeiten pro Subschwarm (unten). Die Fehlerindikatoren repräsentieren die Standardabweichung. Im oberen Diagramm ist klar ersichtlich, dass nur Subschwarm 3 (der „blaue“ Subschwarm) wichtig ist: Ohne ihn bricht die Performanz des Gesamtschwarms komplett ein. Insofern brauchen im unteren Diagramm nur die blauen Balken beachtet zu werden. Bei Betrachtung des unteren Diagramms tritt zu Tage, dass für die Performanz des Schwarms insbesondere Stigmergie und Schwärmerwahrnehmung wichtig sind. Die Wichtigkeitendiagramme werden hier nur beispielhaft dargestellt. Im Verlauf von Verhaltensanalysen werden nur Korrelationsplots dargestellt werden.

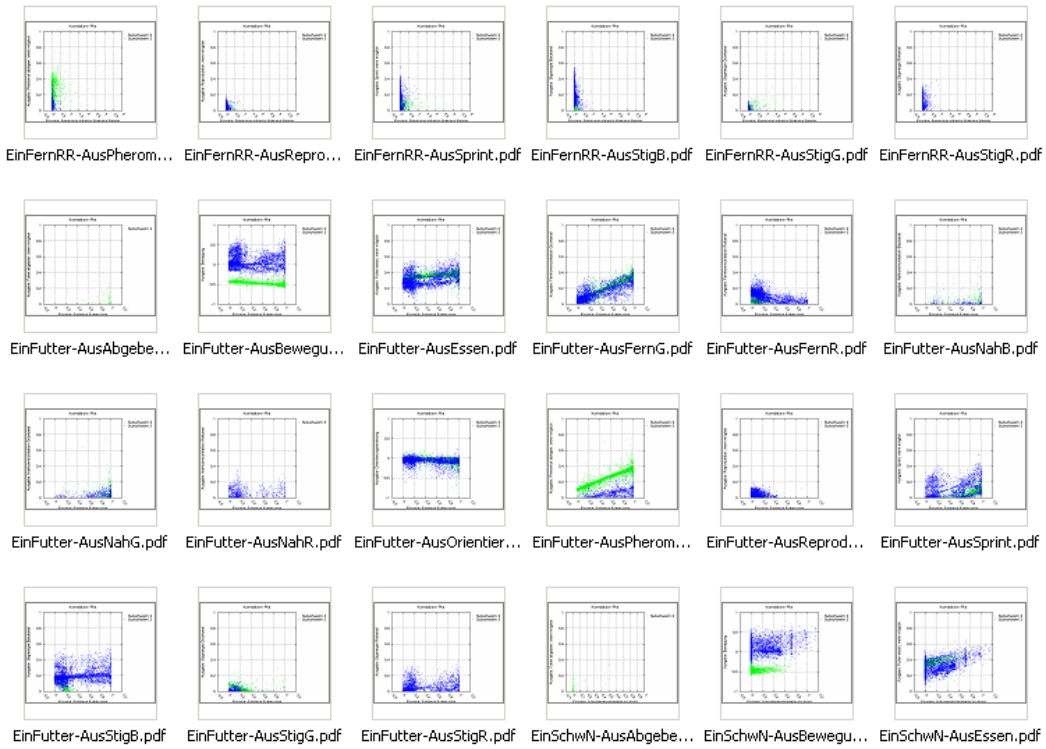


Abbildung 5.12: Miniaturansichten von Korrelationsplots. Die Dateinamen der Plots werden automatisch so gewählt, dass Dateien später auf einfache Weise auffindbar sind.

5.7.1 Evolutionsprinzipien

Ein Experiment findet statt, indem ein Schwarmgenom, also ein Tripel von Schwärmergenomen, unter Benutzung der oben definierten Operatoren durch einen evolutionären Algorithmus optimiert wird. Nach erfolgter Evaluation wird eine neue Population aus folgenden Teilen neu gebildet:

- Elitismus (4% der Population, es werden also die besten 4% der Lösungskandidaten direkt übernommen)
- Zufällig generierte Lösungskandidaten (2%) haben sich vielerorts als nützlich erwiesen, um das Zufallsmoment in der Evolution noch zu verstärken
- Aus Mutation entstandene Lösungskandidaten (70%)
- Aus Kreuzung entstandene Lösungskandidaten (24%)

Dies sind Werte, die dem Diplomanden aus Erfahrung mit verschiedenen Projekten im Bereich evolutionärer Robotik und evolutionärer Schwarmintelligenz als erfolgversprechend erschienen.

Die Lösungskandidaten, welche als Eltern für Mutation und Kreuzung herangezogen werden, werden über eine monotone, rangbasierte Wahrscheinlichkeitsverteilung zufällig ausgewählt (Anhangsabschnitt A.2). Dieses Verfahren ist bewährt, da es zu schnelle Konvergenzen vermeidet, aber trotzdem zielführend ist. Die Populationsgröße ist 250, und pro Evolution werden 100 Generationen durchlaufen. Jedes Experiment wird zehnmal hintereinander durchgeführt, da ein evolutionärer Algorithmus für dieselbe Problemstellung durchaus vielfältige Lösungen finden kann.

5.7.2 Prinzip der Fitnessmessung

Die Fitness wird über eine Simulation mittels der Schwarmsimulationssoftware *SwarmWorld* in einer bestimmten Umwelt ermittelt. Pro Evaluation eines Schwärmertripels wird eine Simulation mit 70.000 Simulationsschritten durchgeführt, die mit einem einzelnen Schwärmer jeder Unterart des Tripels startet. Der Füllwert des internen Futterspeichers der Schwärmer wird mit dem Pessimum für die Größe n des Futterspeichers initialisiert (vgl. Tabelle 5.3 auf Seite 67). Über alle 70.000 Schritte hinweg wird die Anzahl der gerade in der Welt enthaltenen Schwärmer arithmetisch gemittelt. Erzielt die so gebildete Fitnessfunktion einen großen Wert, so ist dies ein Indiz für eine stabile, langlebige Schwärmerpopulation. In jeder Umwelt erscheint kontinuierlich Futter für 20 gleichzeitig lebende Schwärmer, zuzüglich (soweit nicht anders angegeben) einer „Starthilfe“ in Form von Futter, welches direkt zum Experimentanfang am Startplatz der Schwärmer positioniert wird. Die maximal erreichbare Fitness liegt also bei ca. 20. Der für die Experimente relevante Teil der Umwelt (in welchem z.B. Futter entsteht) befindet sich jeweils, soweit nicht anders angegeben, im Radius von 500 Einheiten um den Startpunkt der Schwärmer.

Von der so gebildeten Fitness abgezogen werden Kosten für die entstandenen Kontrollstrukturen, um diese daran zu hindern, ins Unermessliche zu wachsen. Für jedes innere Neuron werden Kosten von $0,03$ von der Fitness abgezogen und für jede Synapse $0,001$. Hierbei verursachen die ersten 5 inneren Neurone und die ersten 5000 Synapsen pro Subschwarm keine Kosten.

5.7.3 Definition eines Experiments

Ein Experiment wird ganz maßgeblich definiert durch die Art der Umwelt, welche die zur Evaluation durchgeführten Simulationen beinhalten. Die verschiedenen Experimentumwelten werden im folgenden Kapitel beschrieben. Als weiterer Parameter sei die Anzahl der zu verteilenden Fertigkeitenpunkte pro Schwärmer genannt. In den Experimenten liegt die Anzahl der zur Verfügung stehenden Fertigkeitenpunkte, soweit nicht anders angegeben, bei 300.

5.7.4 Hardwareinfrastruktur für die Evolutionen

Wie eingangs dargestellt, sollen die in den vergangenen Abschnitten dieses Kapitels beschriebenen Softwarebestandteile in Kombination genutzt werden und so ein evolutionärer Algorithmus auf mehreren Computern ausgeführt werden. Über diese Diplomarbeit hinweg stand eine variable Anzahl Computer zur Verfügung. Zu Spitzenzeiten konnte die Evolution auf ca. 40 Rechner mit ca. 100 CPUs verteilt werden.

Ein Resümee bezüglich der Hardware-Recheninfrastruktur sowie Erfahrungen mit der Gesamtsoftware und ihrer Performanz wird im Zuge der Zusammenfassung und Diskussion der Arbeit (hier speziell: Abschnitt 7.2) gegeben.

Kapitel 6

Experimente und Verhaltensanalysen

Nachdem die technischen Aspekte der freien Schwarmevolution nun ausführlich vorgestellt wurden, sollen in diesem Kapitel konkrete Experimente beschrieben und entstandenes Schwarmverhalten dokumentiert werden. Die Abschnitte 6.1 bis 6.7 beschreiben jeweils separate Experimente. Unterabschnitte in den Experimentabschnitten präsentieren wiederum Beispiele von im Rahmen der jeweiligen Experimente oder deren Variationen entstandenem Schwarmverhalten. Für jedes Experiment wurden zehn separate Evolutionen durchgeführt. So sind oft verschiedene Verhaltensweisen entstanden, welche in Bezug auf das gegebene Experiment ähnliche Performanzwerte erreichen. Es würde den Umfang der Diplomarbeit deutlich überschreiten, jedes entstandene Schwarmverhalten und dessen einzelne Parameter an dieser Stelle zu beschreiben. Im Rahmen jeder Experimentdokumentation wird stattdessen jeweils ein Schwarm ausgewählt, der für das in diesem Experiment entstandene Verhalten repräsentativ ist, aber auch anschaulich dokumentiert werden kann. Das Verhalten des ausgewählten Schwarms wird dann ausführlich anhand der zugehörigen Korrelationsplots dokumentiert, wobei nur die grundlegenden Prinzipien des Verhaltens aufgeschlüsselt werden: Bei vielen Verhaltensweisen wird es noch ein Potential an Einzelphänomenen geben, welche noch diskutiert werden könnten, aber den Rahmen der Arbeit sprengen würden. Andere erwähnenswerte entstandene Verhaltensweisen in Bezug auf das jeweilige Experiment werden gegebenenfalls im Anschluss daran mit niedrigerem Detailgrad skizziert. Sämtliche angegebenen Performanzwerte sind auf eine Nachkommastelle gerundet.

Da es wenig anschaulich ist, Schwarmverhalten mit statischen Bildern zu beschreiben, wurde zu jedem in diesem Kapitel ausführlich präsentierten Verhaltensbeispiel Videomaterial aufgenommen, geschnitten und kommentiert. Es befindet sich auf einer DVD, die der Diplomarbeit beiliegt. Dennoch wurden einige aussagekräftige Bilder entstandenen Verhaltens in dieses Kapitel eingefügt.

In jeder Umwelt erscheint kontinuierlich Futter, das für bis zu 20 gleichzeitig lebende Schwärmer ausreicht. Weiter wird (soweit nicht anders angegeben) eine „Starthilfe“ in Form von etwas Futter direkt zum Experimentanfang am Startplatz der Schwärmer gegeben. Soweit nicht anders angegeben, enthält jedes Futterobjekt 10.000 Futtereinheiten, wobei die spatiale Verteilung der Futterelemente über die Experimente hinweg variiert. Zeitlich werden die Futterelemente der Welt zufällig gleichverteilt hinzugefügt.

Die Experimentfamilie beginnt mit sehr einfach anmutenden Experimenten, d.h. mit um den Ursprung der Welt zufällig gleichverteilt entstehendem Futter. Später wird übergegangen zu verschiedenen Konfigurationen separater Futterstellen, zu dynamischen Futterstellen und komplexeren Umweltstrukturen wie vorgefertigten Pheromonspuren zwischen Futterstellen.

6.1 Feine Futtergranularität

In der Schwarmwelt werden viele kleine Futterobjekte verteilt: Jedes Futterobjekt enthält 5.000 Futtereinheiten. Futterobjekte erscheinen zufällig gleichverteilt positioniert im Radius von 500 Längeneinheiten um den Ursprung (linker Teil der Abb. 6.1 auf der folgenden Seite). Offensichtlich

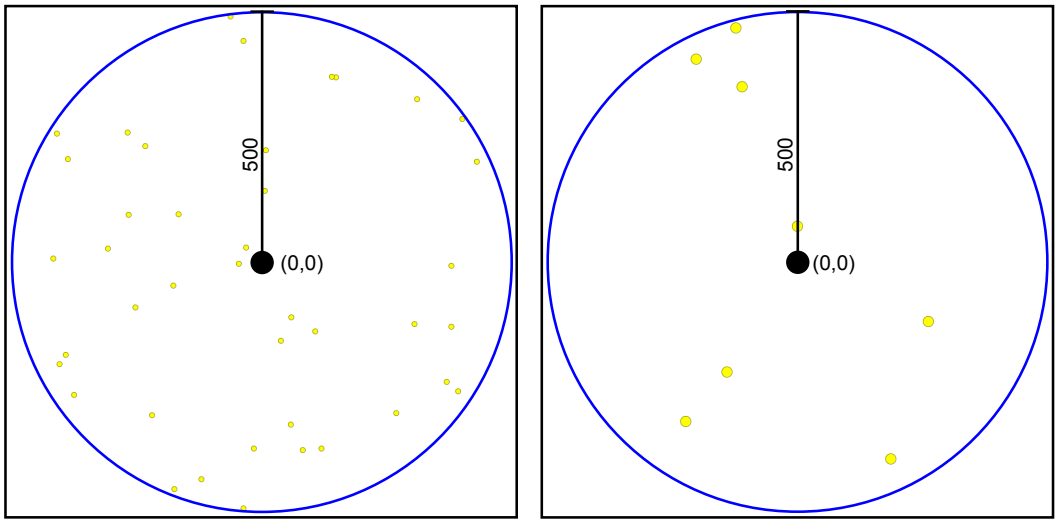


Abbildung 6.1: Typische Weltsituationen bei feiner (links) und grober Futtergranularität (rechts). Beide abgebildeten Welten enthalten ungefähr die gleiche Menge an Futter. Beide Bilder sind 1.000 Einheiten breit, wobei der Ursprung, und damit der Schwärmerstartpunkt, genau in ihrem Zentrum liegt. Der Bereich, in welchem Futter erscheinen kann, ist jeweils mit einem blauen Kreis markiert.

ist es sehr wahrscheinlich, dass ein Schwärmer nach kurzer Fortbewegungszeit auf ein Futterobjekt stößt. Daher ist dieses Experiment nach dem subjektiven Gefühl des Diplomanden das einfachste Experiment, was auch von den Ergebnissen bestätigt wurde: Jeder einzelne der zehn Evolutionsläufe führte zu stabilem Futtersuchverhalten. Alle Referenzperformanzen[1] lagen im Bereich zwischen 8 und 13,6, ihr Durchschnitt lag bei 10,8. Eine konsistente Wichtigkeit bestimmter Fertigkeiten über die zehn Durchläufe hinweg war nicht zu erkennen: Es wurden nur vereinzelt ausgeprägte Wichtigkeiten verschiedener Eigenschaften festgestellt. Insofern wurde für dieses Einstiegsexperiment als repräsentatives Beispiel, welches im Folgenden vorgestellt wird, ein Schwarm ausgewählt, dessen Individuen autark zu agieren scheinen. Dies ermöglicht es, sich bei der Dokumentation dieses ersten Experiments auf einfachere Verhaltensmerkmale – wie z.B. entstandene Fortbewegungsmechanismen – zu konzentrieren, so dass der Leser zunächst in die Analysemethodik hineinfinden kann, bevor komplexere Verhaltensweisen dokumentiert werden.

6.1.1 Beispiel entstandenen Verhaltens

Eine Kurzzusammenfassung wichtiger Größen in Bezug auf das in diesem Beispiel dokumentierte Verhalten findet sich in Tabelle 6.1 auf der rechten Seite. Zugehörige, relevante Korrelationsplots befinden sich in Abb. 6.2 auf der rechten Seite. Das Verhalten eines einzelnen Schwärmers scheint in einem kontinuierlichen, schnellen Rotieren um die eigene Achse entgegen dem Uhrzeigersinn zu bestehen, solange der schwärmereigene Futterspeicher gut gefüllt ist (Korrelationsplot (a)). Durch die schnelle Rotation nimmt der Schwärmer Futterobjekte in seiner Sichtweite selbst dann nicht wahr, wenn die Entfernung zwischen Schwärmer und Futterobjekt deutlich kleiner als die Sichtweite ist. Je niedriger der Füllstand des Futterspeichers ist, desto langsamer findet die Rotation um die eigene Achse statt, Futterobjekte in Sichtweite werden nach und nach wahrnehmbar. Die Bewegungsausgabe[2] des Schwärmers ist grundsätzlich auf verschiedene positive Werte gesetzt (Korrelationsplot (b)) und verringert sich mit abnehmendem Füllstand des Futterspeichers. Der Schwärmer trägt also kontinuierlich Vortriebskräfte in Richtung seiner Orientierung auf seinen

[1] Erinnerung: Die Schwärmer können – bei optimaler Ausnutzung jeglichen erscheinenden Futters – eine maximale Performanz von ca. 20 erreichen, wobei die Vermehrungskosten hier noch unberücksichtigt sind.

[2] Erinnerung: Die Eingaben der Schwärmerkontrollstruktur sind aufgelistet in Tabelle 5.1 auf Seite 64, die Ausgaben in Tabelle 5.2.

Wichtige Subschwärme:	Subschwarm 1 (volle Wichtigkeit)
Stark ausgebildete Fertigkeiten:	Fortbewegung (54 Punkte) und Nahbereichskommunikation (60 Punkte) sind die ausgeprägtesten Metaparameter. Die ausgeprägte Nahbereichskommunikation spiegelt sich jedoch nicht im Verhalten wieder, sie wurde nicht als wichtig erkannt.
Wichtige Fertigkeiten:	Keine der zum Test entfernten Fertigkeiten ist für das Verhalten des Schwarms essentiell.
Referenzperformanz:	8,5 (Erinnerung: Dies bedeutet, dass durchschnittlich kontinuierlich 8,5 Schwärmer in der Welt gelebt haben.)
Kooperation:	Keine feststellbar

Tabelle 6.1: Kurzzusammenfassung: Ergebnisbeispiel feine Futtergranularität.

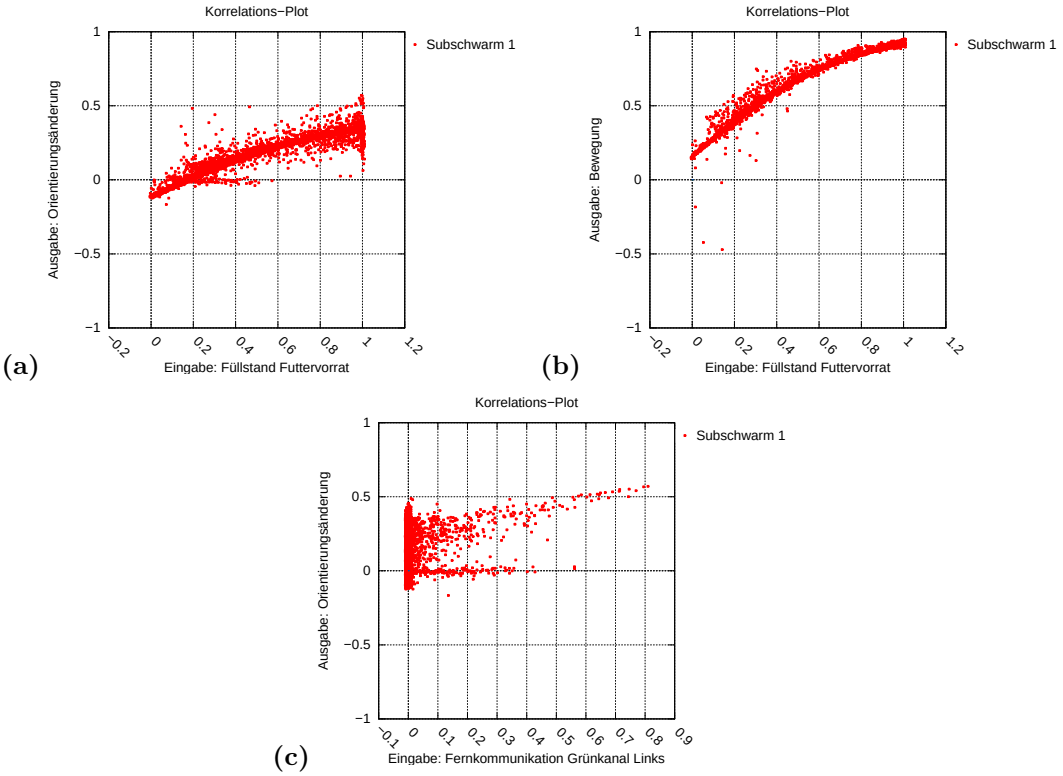


Abbildung 6.2: Korrelationsplots: Ergebnisbeispiel feine Futtergranularität. Bei Korrelationsplots, welche als Ausgabe die Orientierungsänderungen darstellen, ist zu beachten, dass positive Ausgaben einer Drehung entgegen dem Uhrzeigersinn entsprechen.

Körper auf. Wenn er aber gleichzeitig schnell rotiert, ändert sich die Orientierung sehr schnell. So werden diese Kräfte in verschiedenste Richtungen aufgetragen und eliminieren sich gegenseitig. Der Schwärmer rotiert also praktisch auf der Stelle. Er bewegt sich erst fort, wenn die Rotation langsamer wird, was mit Abnahme des Futterfüllstandes der Fall ist.

So ist zu erklären, wie ein Schwärmer sich schlussendlich auf ein Futterobjekt zubewegen kann: Dies wird ermöglicht, indem die maßgeblich vom Futterfüllstand bestimmte Orientierungsänderung einen Nulldurchgang aufweist, wenn der Futterspeicher zu ca. 20% gefüllt ist (Korrelationsplot (a)). Liegt der Füllstand in diesem Bereich, wird das standardmäßig aktivierte Rotationsverhalten also nahezu gestoppt. Es tritt dann folgendes Verhalten zu Tage: Falls der Schwärmer in der linken Zone des Sehsinns Futter sieht, wird eine Orientierungsänderung in diese Richtung ausgelöst (Korrelationsplot (c), wobei sich die rechte Sichtzone und der Rotkanal[3] analog verhalten). Der Schwärmer behält wahrgenommenes Futter also „im Auge“. So werden in der Nähe liegende, wahrgenommene Futterobjekte durch die durchgehend aktivierte Fortbewegung angesteuert und können aufgenommen werden (Die Nahrungsaufnahme-Ausgabe des Schwärmers ist immer aktiv). Die Schwärmer vermehren sich nur in unmittelbarer Nähe zu Futterobjekten. Systematische Kommunikation oder Interaktion zwischen Schwärmern ist nicht erkennbar.

Dies führt insgesamt zu einem effizienten, bedarfsorientierten, jedoch individuellen Verhalten, welches bei äußerer Betrachtung den Bewegungen z.B. eines Moskitoschwarms nicht unähnlich ist. Offensichtlich ist es wenig aussagekräftig, ein Verhalten, das maßgeblich darauf basiert, kontinuierliche Rotation zu bestimmten Zeitpunkten zu stoppen, mittels statischer Bilder zu illustrieren. Insofern sei für die Illustration dieses Verhaltens auf das zugehörige, kommentierte Video[4] verwiesen.

6.2 Grobe Futtergranularität

Bei diesem Experiment werden in der Schwarmwelt wenige, große Futterobjekte verteilt: Jedes Futterobjekt enthält 25.000 Futtereinheiten. Futterobjekte erscheinen wieder zufällig gleichverteilt positioniert im Radius von 500 Einheiten um den Ursprung (rechter Teil der Abb. 6.1 auf Seite 76). Offensichtlich ist es im Rahmen dieses Experiments für die Schwärmer schwieriger, überhaupt Futterobjekte zu finden. Als weitere, weniger offensichtliche Schwierigkeit stellt sich dann die geringere Anzahl Futterobjekte im Vergleich zur Anzahl der Schwärmer dar: Aus ihr folgt, dass tendenziell mehr Schwärmer gleichzeitig vom gleichen Futterobjekt Nahrung aufnehmen möchten. Dies stellt die Schwärmer vor die schwierige Aufgabe des „Zugangsmanagements“: Sie müssen dafür sorgen, dass auch tatsächlich viele Schwärmer gleichzeitig eine Futterstelle nutzen können, und Futterstellen nicht von einigen wenigen Schwärmern z.B. verdeckt werden.

Unter den Umständen dieses Experiments waren nur 4 von 10 Evolutionen überhaupt erfolgreich, brachten also stabiles Futtersuchverhalten hervor. Die Referenzperformanzen der erfolgreichen Durchläufe lagen zwischen 4 und 8,9, ihr Durchschnitt lag bei 5,3. Die Referenzperformanzen sämtlicher anderen – bei diesem und in den folgenden Experimenten als *unerfolgreich* titulierten – Verhaltensweisen lagen im Bereich zwischen 0 und 1 und zeigten auch im Laufe der Evolution keine signifikanten Fortschritte. Verhaltensweisen mit Referenzperformanzen > 1 werden in diesem und den folgenden Experimenten als *erfolgreich*[5] bezeichnet. Fertigkeiten, welche sich bei allen erfolgreichen Schwärmen als wichtig herausstellten, waren Futterabgabe und Stigmergie.

[3] Erinnerung: Futter ist gelb, reizt also Rot- und Grünkanal des Sehsinns.

[4] Hinweis: Die Videos werden von einem Menü aus angewählt. Jedes Video trägt den Namen des zugehörigen Experimentes.

[5] Diese Unterteilung ist insofern plausibel, als dass es nach erfolgtem Experiment in der Regel zwei Parteien von Verhaltensweisen gab: Solche mit Referenzperformanzen < 1 und solche mit Referenzperformanzen *deutlich* größer als 1. Die Ergebnisse sahen durchweg so aus, dass ein Schwarm entweder keinen Erfolg zeigt, oder er löst seine Aufgabe recht performant. Zwischen den unerfolgreichen und erfolgreichen Schwärmen entstand regelmäßig eine klaffende Performanzwert-Lücke.

Wichtige Subschwärme:	Subschwarm 1 (volle Wichtigkeit)
Stark ausgebildete Fertigkeiten:	Keine sehr herausragend ausgebildete Fertigkeit. Stärkste Fertigkeit ist Wahrnehmung der Fernkommunikation bzw. Sicht mit 53 Punkten.
Wichtige Fertigkeiten:	Futterabgabe (Wichtigkeit 0,93) und Stigmergie (Wichtigkeit 0,79).
Referenzperformanz:	6,3
Kooperation:	Schwärmer verteilen Futter, indem sie kurz nach Futteraufnahme wieder welches abgeben. Sie legen weiter Pheromone in Futternähe, die sie und andere Schwärmer daran hindern, den relevanten Teil der Schwarmwelt zu verlassen.

Tabelle 6.2: Kurzzusammenfassung: Ergebnisbeispiel grobe Futtergranularität.

6.2.1 Beispiel entstandenen Verhaltens

Eine Kurzzusammenfassung wichtiger Größen in Bezug auf das in diesem Beispiel dokumentierte Verhalten findet sich in Tabelle 6.2. Zugehörige, relevante Korrelationsplots befinden sich in Abb. 6.3 auf der folgenden Seite. Die Orientierungsänderung der Schwärmer wird in diesem Beispiel durch einen bis auf einige Ausreißer linearen Zusammenhang vom Futterspeicherfüllstand bestimmt (Korrelationsplot (c)). Nehmen sie jedoch mittels der Rot- und Grünkanäle beider Zonen des Sichtsensors Futter wahr, so beschleunigen die Schwärmer stark (Korrelationsplot (a), Korrelationsplots bezüglich Grünkanal und linker Sichtzohne analog). Auf diese Weise wird Futter gefunden und angesteuert. Ein Schwärmer bewegt sich tendenziell schneller, wenn sein Futterfüllstand nahe 1 ist (Korrelationsplot (b)).

Eine weitere Problemstellung im Experiment mit grober Futtergranularität ist es für die Schwärmer, nicht mangels sensorischer Eingaben in der Welt verloren zu gehen. Aus diesem Grund werden – tendenziell bei höherem Futterfüllstand – Pheromone abgelegt (Korrelationsplot (e)), insbesondere solche mit hohen Rot- und Blauwerten (Korrelationsplot (f), roter Pheromonausgabekanal sowie linke Sichtzone und grüner Sichtkanal analog). Weiter gibt es eine Korrelation zwischen dem Wahrnehmen von Pheromonen mit starkem Rot- und Blauanteil – besonders wenn sie subjektiv östlich oder südlich vom Schwärmer sind – und geringerer Fortbewegungsgeschwindigkeit (Korrelationsplot (d), restliche Plots mit rotem Kanal und südlicher Zone analog). Hier bremsen also Pheromone die Schwärmer und hindern sie auf diese Weise für kurze Zeit daran, sich von Positionen zu entfernen, an denen einst Futter lag. Schwärmer vermehren sich in unmittelbarer Nähe zu Futter.

Schließlich sei die Funktion der hier wichtigsten Eigenschaft – der Fähigkeit, Futter in die Welt abzugeben – erklärt. Futter wird tendenziell abgegeben, wenn der Futtervorrat fast voll ist (Korrelationsplot (g)). Die Schwärmer „zeichnen“ so Spuren von kleinen Futterobjekten, sobald sie ausreichend Futter aufgenommen haben (Abb. 6.4 auf Seite 81). Entsteht also eine Futterstelle, so verteilen die Schwärmer das Futter auf diese Weise, so dass jeder Schwärmer Futter aufnehmen kann.

6.2.2 Kurzbeschreibung weiterer entstandener Verhaltensweisen

Oben wurde bereits die in diesem Experiment für die Schwärmer vorhandene Problemstellung angesprochen, dass tendenziell mehr Schwärmer gleichzeitig vom gleichen Futterobjekt Nahrung aufnehmen möchten und sich so gegenseitig behindern können. Ein weiterer entstandener Schwarm löste dieses Problem mittels Nahbereichskommunikation (im Folgenden auch zu „Nahkommunikation“ abgekürzt). Ist ein Schwärmer alleine, so verweilt er einfach auf einer gefundenen Futterstelle. Kommt jedoch ein anderer Schwärmer hinzu, welcher hohe Werte auf dem Rotkanal der Nahbereichskommunikation aussendet, so beschleunigt der auf der Futterstelle verweilende Schwärmer

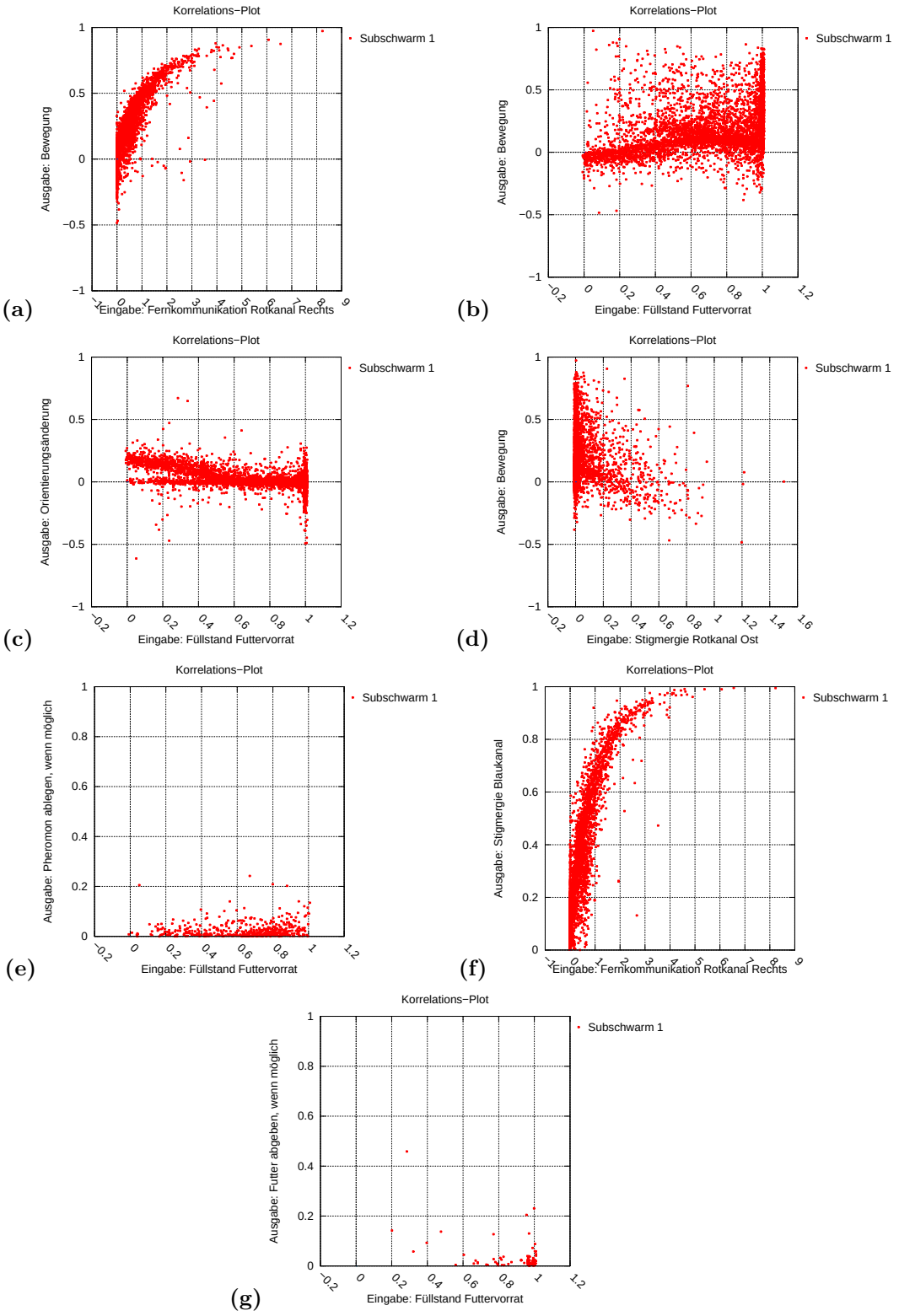


Abbildung 6.3: Korrelationsplots: Ergebnisbeispiel grobe Futtergranularität.

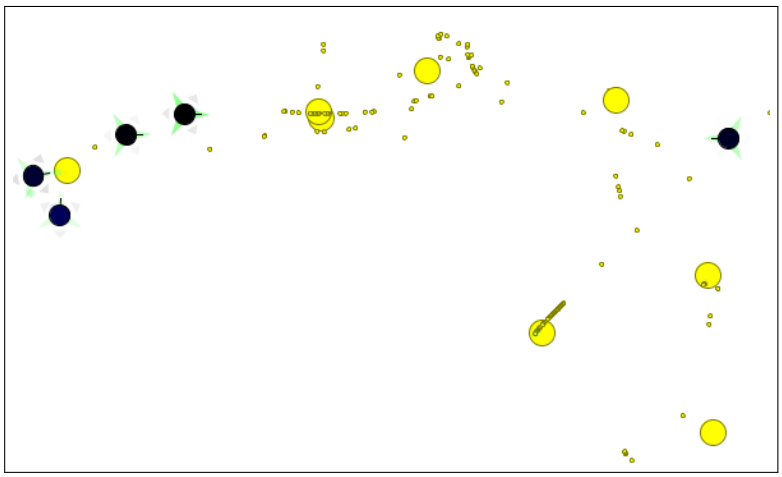


Abbildung 6.4: Von Schwärmern gelegte, deutlich auf den großen Futterobjekten sichtbare Futterspuren im Ergebnisbeispiel zum Experiment mit grober Futtergranularität.

stark, macht also den Weg frei. Rote Nahkommunikation emittieren die Schwärmer tendenziell öfter, wenn ihr Futterfüllstand relativ niedrig ist oder wenn sie rote Nahkommunikation empfangen. Ein einziger hungriger Schwärmer kann so ganze Trauben von anderen Schwärmern dazu bewegen, Platz zu machen. Dennoch lag die Referenzperformanz dieses Schwarms mit 3,4 deutlich unter derjenigen des oben als repräsentativ beschriebenen Schwarms.

6.3 Vier separate Futterstellen

Futterobjekte erscheinen bei diesem Experiment nicht mehr gleichverteilt in einem Kreis mit festem Radius um den Ursprung, sondern gleichverteilt innerhalb von vier separaten, kreisförmigen Futterstellen mit jeweils 75 Längeneinheiten Radius. Eine der Futterstellen hat ihren Mittelpunkt im Ursprung, die Mittelpunkte der drei restlichen sind auf einem Kreis mit Radius 425 um den Ursprung äquidistant verteilt (Abb. 6.5 auf der folgenden Seite).

Bei diesem Experiment ist offensichtlich die Problemstellung gegeben, dass die Schwärmer die Futterstellen, welche nicht direkt beim Ursprung liegen, finden und erreichen müssen. Die Entfernung zu den äußeren Futterstellen wurde so gewählt, dass ein gut ausgeprägter Sehsinn alleine zur stabilen Bewältigung dieser Aufgabe mutmaßlich nicht ausreicht. Die Entfernung von einer Futterstelle zur nächsten liegt zwar unter der maximalen Sichtweite, jedoch in einer Entfernung, bei der das sensorische Rauschen nicht mehr vernachlässigbar ist. 6 von 10 evolvierten Schwärmen lösten diese Aufgabe erfolgreich mit Referenzperformanzen von 2,4 bis 9,9 (durchschnittlich 5,9), indem sie unter den gegebenen Umständen stabiles Futtersuchverhalten entwickelten. Einheitliche Eigenschaftenwichtigkeiten bildeten sich nicht heraus. Ähnlich des im vorherigen Experiment ausführlich dokumentierten Verhaltens war es die Strategie mehrerer Schwärme, vorhandene Futterstellen mit Pheromonen zu markieren, wobei neue Futterstellen durch Schwärmer gefunden wurden, welche „auf gut Glück“ eine Futterstelle verließen (und hierbei oft verhungerten). Bedingt durch die bessere Sichtbarkeit der Futterstellen waren diese Markierungen aber tendenziell weniger wichtig für die Zielerreichung des Schwarms. Insofern soll sich die folgende Dokumentation einem Beispiel widmen, bei der eine andere Fertigkeit essentielle Wichtigkeit innehatte: Die Nahbereichskommunikation.

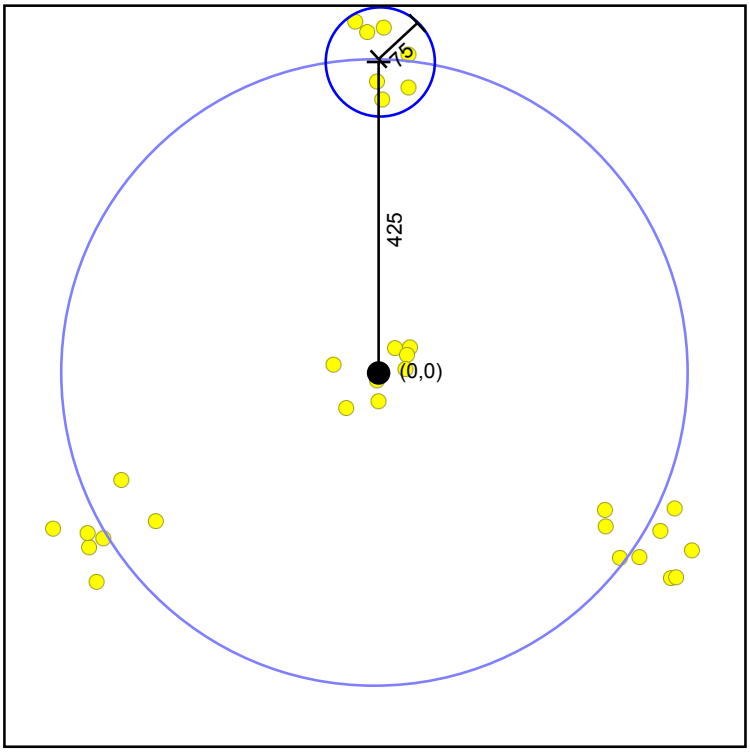


Abbildung 6.5: Typische Weltsituation beim Experiment mit vier separaten, statischen Futterstellen. Die abgebildete Welt enthält ungefähr die gleiche Menge Futter wie schon diejenigen in Abb. 6.1 auf Seite 76. Das Bild ist 1.000 Einheiten breit, wobei der Ursprung, und damit der Schwärmerstartpunkt, genau in ihrem Zentrum liegt. Die oberste Futterstelle ist exemplarisch mit einem dunkelblauen Kreis markiert. Auf dem hellblauen Kreis liegen die Mittelpunkte der äußeren Futterstellen.

Wichtige Subschwärme:	Subschwarm 1 (volle Wichtigkeit)
Stark ausgebildete Fertigkeiten:	Keine sehr herausragend ausgebildete Fertigkeit. Stärkste Fertigkeit ist Wahrnehmung der Fernkommunikation bzw. Sicht mit 48 Punkten.
Wichtige Fertigkeiten:	Nahbereichskommunikation (Wichtigkeit 0,95)
Referenzperformanz:	7,3
Kooperation:	Schwärmer beeinflussen sich intensiv über die Nahbereichskommunikation.

Tabelle 6.3: Kurzzusammenfassung: Ergebnisbeispiel bei vier separaten Futterstellen.

6.3.1 Beispiel entstandenen Verhaltens

Eine Kurzzusammenfassung wichtiger Größen in Bezug auf das in diesem Beispiel dokumentierte Verhalten findet sich in Tabelle 6.3. Zugehörige, relevante Korrelationsplots befinden sich in Abb. 6.6 auf der folgenden Seite.

Die Schwärmer bewegen sich hier kontinuierlich vorwärts, wobei drei Faktoren eine Beschleunigung der Fortbewegung verursachen. Die Geschwindigkeit steigt tendenziell mit dem Füllstand des Futterspeichers (Korrelationsplot (a)). Weiter wird beschleunigt, wenn über den Sehsinn Futter wahrgenommen wird (Korrelationsplot (b), restliche Plots für linke Sichtzone und Rotkanal analog). Zuletzt gibt es eine Korrelation zwischen hoher Fortbewegungsgeschwindigkeit und hohen Eingabewerten in Bezug auf den Blaukanal der Nahbereichskommunikation (Korrelationsplot (c), restliche Zonen analog).

Die Orientierungsänderung wird – ähnlich wie bei den schon behandelten Beispielen – offensichtlich ebenfalls beeinflusst vom Füllstand des Futterspeichers, wobei der Zusammenhang bei vollem Futterspeicher jedoch nicht mehr gut sichtbar ist. Dies wird verursacht dadurch, dass ein Schwärmer aktiv Futter ansteuert, wenn er es sieht (Korrelationsplot (e)). Wenn der Schwärmer links Futter sieht, wird nach links gesteuert, rechte Zone und roter Sichtkanal analog). So kommt es, dass der Schwärmer mit vollem Futterspeicher direkt das nächste Futterelement ansteuert, also viele Orientierungsänderungen bei hohem Futterspeicherfüllstand ausgegeben werden können. Die Orientierungsänderung scheint jedoch mit wachsenden Eingaben des Nahbereichskommunikations-Blaukanals abzunehmen (Korrelationsplot (f)).

Bleibt zu betrachten, wodurch blaue Nahbereichskommunikationssignale verursacht werden. Zum einen ist die Ausgabe des Nahbereichskommunikationssignals hoch, wenn der Schwärmer einen gut gefüllten Futterspeicher hat (Korrelationsplot (g)), zum anderen, wenn er Futter sieht (Korrelationsplot (h), linke Sichtzone und Rotkanal analog).

Zusammengefasst passiert folgendes: Nehmen Schwärmer nichts wahr, so bewegen sie sich langsam fort, wobei eine anfangs leicht gekrümmte Bahn mit wachsender Orientierungsänderung in eine Spirale und Rotation auf der Stelle mündet und der Schwärmer schließlich verhungert. Dieses Explorationsverhalten findet auch statt, wenn eine Futterstelle verschwindet (z.B. weil sie schnell verbraucht wurde), so dass in diesem Fall oft eine der anderen Futterquellen gefunden wird. Sieht der Schwärmer auf seinem Weg Futter, so steuert er es beschleunigt an und beschleunigt nach der Futteraufnahme weiter, wobei er blaue Nahbereichskommunikationssignale emittiert. Nimmt er andere Schwärmer wahr, die solche Signale emittieren, so verstärkt sich die Beschleunigung weiter. Der Schwärmer durchquert also durch die Beschleunigung sehr schnell eine Futterstelle, nimmt dabei Futter auf, sieht kein Futter mehr, wird daher wieder langsamer, und kehrt durch das dann sinkende Verhältnis von Geschwindigkeit zu Orientierungsänderung sehr bald wieder um. Er sieht wieder Futter und beschleunigt dadurch erneut auf die Futterstelle zu. So bilden sich Schwärmergruppen, die um Futterstellen pulsieren (Abb. 6.7 auf Seite 85). Es kommt hierbei jedoch immer wieder vor, dass mehrere Schwärmer die Futterstelle gleichzeitig in ungefähr gleicher Richtung verlassen. In diesem Fall beschleunigen sie durch die Stimulation der blauen Nahbereichskommunikation länger, bewegen sich also weiter von der Futterstelle weg. Durch sol-

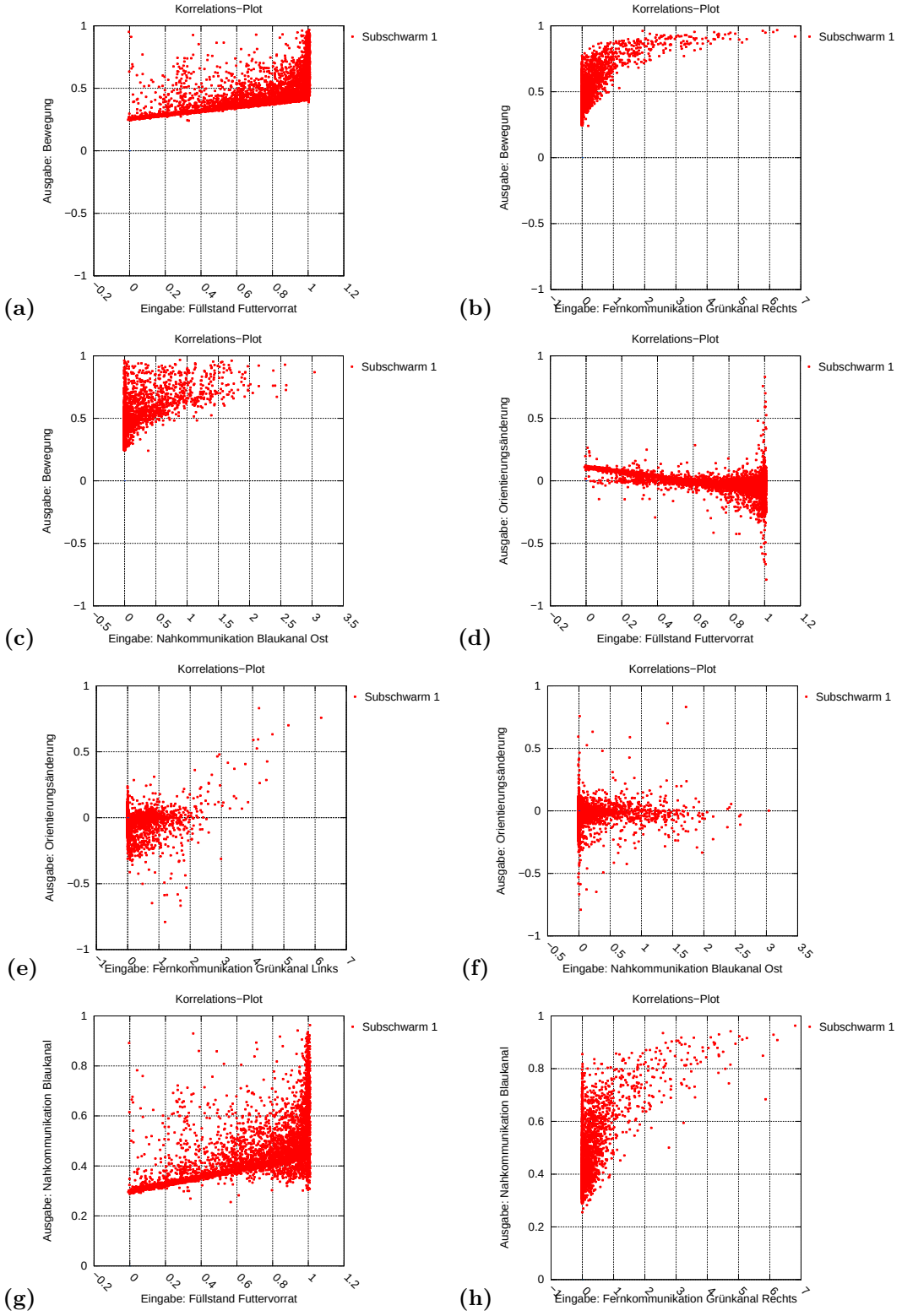


Abbildung 6.6: Korrelationsplots: Ergebnisbeispiel bei vier separaten Futterstellen.

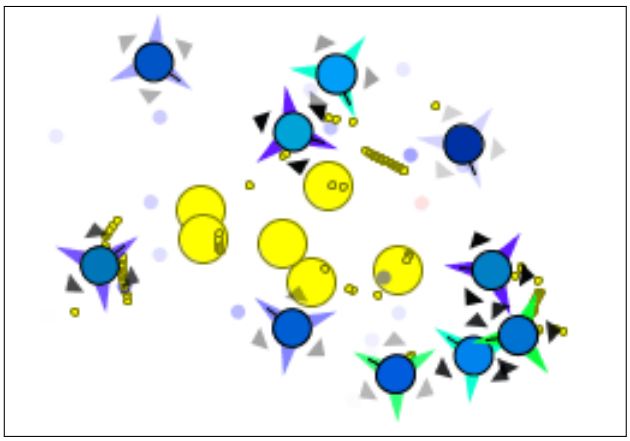


Abbildung 6.7: Schwärmergruppe im Experiment mit 4 separaten Futterstellen. Deutlich zu sehen die blaue Nahbereichskommunikation.

che – sich in zufällige Richtungen von der bekannten Futterstelle entfernenden – „Kundschafter“ werden weitere Futterstellen entdeckt.

Bleibt noch ein wichtiger Aspekt zu beschreiben: Die Vermehrung der Schwärmer. Die Schwärmer vermehren sich, wenn der Füllstand des Futtervorrats nahe 1 ist. Die Vermehrung wird ebenfalls von der Nahbereichskommunikation reguliert. Deaktiviert man die Nahbereichskommunikation, so erfolgt eine rasante Vermehrung des Schwarms, wobei die Schwärmer wahrgenommenes Futter äußerst schnell verzehren, so dass der Schwarm an Nahrungsmangel eingeht. Dieser Verhaltensaspekt wird hier nicht in einem Korrelationsplot vorgestellt, sondern in einem eigenen Videoteil im Beschreibungsvideo zu diesem Verhaltensbeispiel veranschaulicht.

6.4 Zwei alternierende, dynamische Futterstellen

Anstatt der vier Futterstellen im vorigen Experiment existieren in diesem Experiment zwei gleichartige Futterstellen, nun jeweils auf einen Durchmesser von 100 reduziert. Deren Mittelpunkte sind jedoch nicht statisch. Beide Mittelpunkte befinden sich am Start der Simulation am Ursprung und am Ende der Simulation an den Punkten $(450; 0)$ bzw. $(-450; 0)$. Die Futterstellen driften also auseinander. Zwischen Anfang und Ende werden die beiden Mittelpunkte der Futterstellen durch lineare Interpolation zwischen dem Ursprung und den beiden Randpunkten erzeugt. Die Futterstellen alternieren insofern, als dass immer nur eine der beiden Futterstellen zu einem gegebenen Zeitpunkt gültig ist, also mit neuen Futterelementen bestückt wird: Alle 4.000 Simulationsschritte wechselt die gültige Futterstelle jeweils zur gerade ungültigen. Insgesamt beschreibt das Experiment zwei sich abwechselnde Futterstellen, welche sich kontinuierlich voneinander entfernen (Abb. 6.8 auf der folgenden Seite).

Ursprünglich sollte dieses Experiment die Schwärmer zwingen, die Futterstellen zu wechseln und sich auf irgendeine Weise in der Welt zu orientieren (beispielsweise durch das Ablegen von Landmarken in Form von Pheromonen). Doch dadurch, dass die gültige Futterstelle alle 4.000 Simulationsschritte wechselt, ein Futterobjekt jedoch maximal 10.000 Zeitschritte in der Welt verbleiben darf, bietet sich den Schwärmern auch die Möglichkeit, mit dem Futter zu haushalten, so dass die Futterstellen gar nicht erst gewechselt werden müssen. In diesem Fall teilt sich die Schwärmerpopulation mit der Zeit und „folgt“ den auseinanderdriftenden Futterstellen. Sämtliche in diesem Experiment entstandenen Schwärmer haben dieses Verhalten mit leichten Variationen entwickelt. Alle zehn Experimentdurchläufe waren erfolgreich, wobei die Referenzperformanzen zwischen 11 und 14,8 lagen (durchschnittlich 12,9). Die Eigenschaftenwichtigkeiten zeigten kein einheitliches Bild. Im ausführlichen Beispiel wird nun eine Variante des oben bereits beschriebenen Verhaltens vorgestellt.

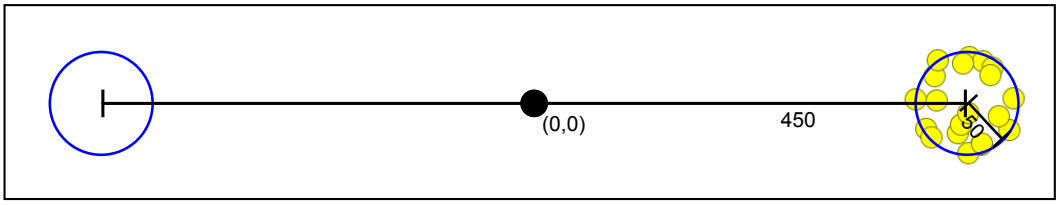


Abbildung 6.8: Typische Weltsituation beim Experiment mit zwei alternierenden, dynamischen Futterstellen zum Ende der Simulation (bei Zeitschritt 70.000). Das Bild ist ca. 1.000 Einheiten breit, wobei der Ursprung, und damit der Schwärmerstartpunkt, genau in seinem Zentrum liegt. Beide Futterstellen (die aktuell gültige und die alternative) sind mit einem blauen Kreis markiert.

Wichtige Subschwärme:	Subschwarm 2 (volle Wichtigkeit)
Stark ausgebildete Fertigkeiten:	Stärkste Fertigkeiten sind Nahbereichskommunikation (46) und Stigmergie (44), spiegeln sich jedoch beide nicht in den wichtigen Fertigkeiten wieder.
Wichtige Fertigkeiten:	Schwärmerwahrnehmung (0,65).
Referenzperformanz:	13,6
Kooperation:	Reduzieren in der Gruppe die Geburtenrate und hemmen die Nahrungsaufnahme, so dass mit dem Futter gehaushaltet werden kann

Tabelle 6.4: Kurzzusammenfassung des Ergebnisbeispiels bei zwei alternierenden, dynamischen Futterstellen.

6.4.1 Beispiel entstandenen Verhaltens

Eine Kurzzusammenfassung wichtiger Größen in Bezug auf das in diesem Beispiel dokumentierte Verhalten findet sich in Tabelle 6.4. Zugehörige, relevante Korrelationsplots befinden sich in Abb. 6.9 auf der rechten Seite.

Die Fortbewegung und Orientierungsänderung erfolgt mit bekannten Mechanismen: Die Schwärmer bleiben in der Nähe der Futterstellen. Kompliziertere Mechanismen, die „Kundschafter“ erzeugen, existieren nicht. Vielmehr teilt sich die Schwärmerpopulation mit wachsender Entfernung der beiden Futterquellen und spart an Vermehrung und Futter, um sich über „Dürreperioden“, welche durch die wechselnden Gültigkeiten hervorgerufen werden, zu retten. Relevant ist also zu beschreiben, warum die Schwärmerwahrnehmung als einzige Fertigkeit bei diesem Subschwarm so wichtig ist: Über diese erfolgt eine Geburtenkontrolle (siehe Korrelationsplot (a), restliche drei Zonen analog). Werden viele Schwärmer wahrgenommen, findet keine Reproduktion mehr statt. Analog hemmen wahrgenommene Schwärmer die Nahrungsaufnahme (Korrelationsplot (b), restliche Zonen analog). Weiter nehmen Schwärmer fast nur dann Futter zu sich, wenn der Futterfüllstand bei 0,2 oder weniger angelangt ist (Korrelationsplot (c)). Im zugehörigen Video ist neben der Dokumentation dieses Verhaltens auch festgehalten, wie sich der Schwarm verhält, wenn man die Schwärmerwahrnehmung deaktiviert: Die Schwärmer vermehren sich viel zu stark und Futter verbraucht sich daher viel zu schnell.

6.5 Zwei scharf alternierende, dynamische Futterstellen

Dieses Experiment unterscheidet sich vom vorherigen in zweierlei Hinsicht: Zum einen alternieren die Futterstellen nicht mehr nach 4.000, sondern nach 8.000 Simulationsschritten, zum anderen

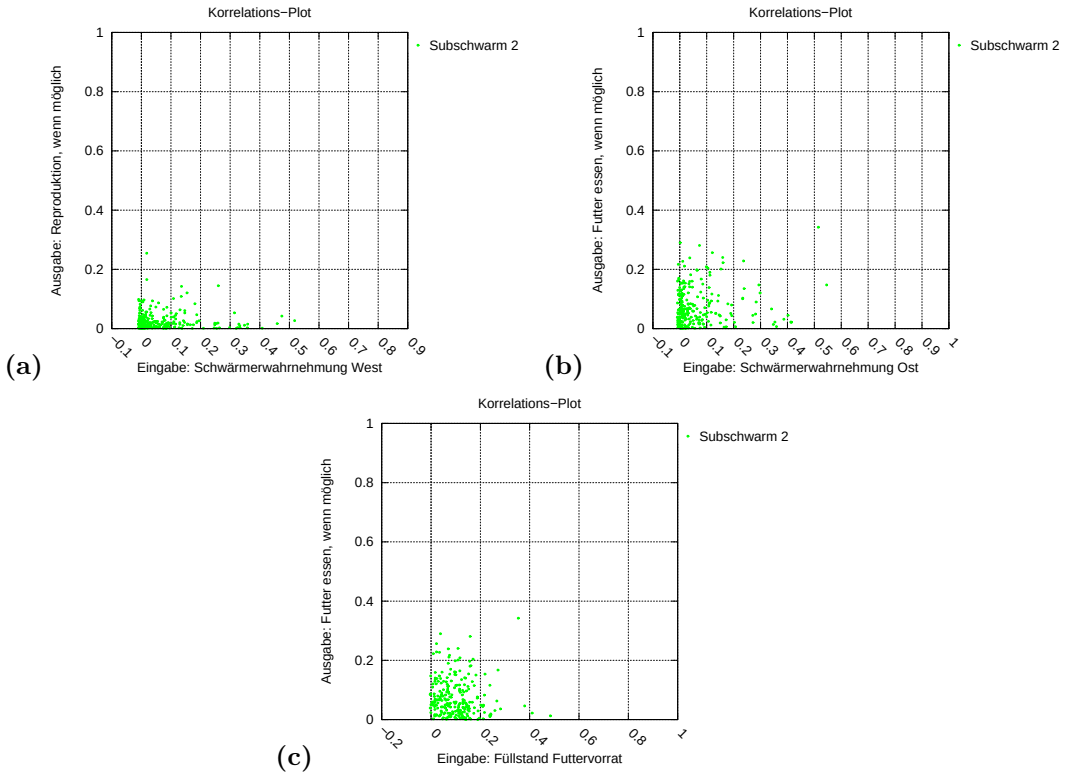


Abbildung 6.9: Korrelationsplots: Ergebnisbeispiel bei vier separaten Futterstellen. Alle drei Korrelationsplots stellen Schalterausgaben dar, weshalb negative Ausgaben schlicht abgeschnitten werden. Erinnerung: Die Plots sind hier grün, da es sich um Plots zu Subschwarm 2 handelt. Korrelationsdatenpunkte von Subschwarm 1 wären rot gefärbt, von Subschwarm 3 blau. Die Färbung ist ein Automatismus, der in die Testsuite implementiert wurde, um eventuelle Spezialisierungen verschiedener, in einer Simulation wichtiger Subschwärme direkt in den Plots sichtbar machen zu können (beispielhaft z.B. in Abb. 5.12 auf Seite 73 dargestellt).

Wichtige Subschwärme:	Subschwarm 2 (volle Wichtigkeit)
Stark ausgebildete Fertigkeiten:	Ausgebildetste Fertigkeit (53) ist die Wahrnehmung der Nahbereichskommunikation. Dies spiegelt sich auch in der Wichtigkeit wieder.
Wichtige Fertigkeiten:	Nahbereichskommunikation (0,9)
Referenzperformanz:	10,9
Kooperation:	Bleiben mittels Nahbereichskommunikation zusammen und hemmen gegenseitig Vermehrung.

Tabelle 6.5: Kurzzusammenfassung des Ergebnisbeispiels bei zwei scharf alternierenden, dynamischen Futterstellen.

wird das Futter nicht mehr zeitlich zufällig verteilt. Beginnt der Gültigkeitszeitraum einer Futterstelle, so erhält diese sofort die Menge Futter, welche dort über den gesamten Gültigkeitszeitraum hinweg deponiert worden wäre (im Rahmen der Videos wird der Versuchsaufbau im Zeitraffer dargestellt). Dies zwingt die Schwärmer zum einen, mit dem Futter zu haushalten, zum anderen ist sichergestellt, dass (bis auf Übergangsperioden von 2.000 Zeitschritten Länge) wirklich nur an einer von beiden Futterstellen Futter existiert (es sei an dieser Stelle daran erinnert, dass Futterobjekte maximal 10.000 Zeitschritte in der Welt verweilen). Die Schwärmer werden also gezwungen, die Futterstellen zu wechseln. Diese zeitliche Konfiguration wird *scharf* alternierend genannt.

Auch bei dieser Variante des Zwei-Futterstellen-Experimentes brachten alle zehn Durchläufe stabile Populationen hervor. Die Referenzperformanzen lagen allerdings zwischen den Werten 6,2 und 10,9 (durchschnittlich 8,8) und damit deutlich niedriger als beim letzten Experiment. Für den höheren Schwierigkeitsgrad des Experiments spricht weiterhin, dass es bei jedem entstandenen Verhalten Eigenschaften gab, die essentiell wichtig waren. Besonders vertreten unter den wichtigen Eigenschaften waren Stigmergie (5×) und Schwärmerwahrnehmung (5×), aber auch Nahbereichskommunikation (4×).

6.5.1 Beispiel entstandenen Verhaltens

Als anschauliches Beispiel sei ein Schwarm genannt, bei dem die Nahbereichskommunikation essentiell wichtig ist.

Eine Kurzzusammenfassung wichtiger Größen in Bezug auf das in diesem Beispiel dokumentierte Verhalten findet sich in Tabelle 6.5. Zugehörige, relevante Korrelationsplots befinden sich in Abb. 6.10 auf der rechten Seite.

Zunächst sei ein sehr offensichtlicher Zusammenhang festgehalten: Falls ein Schwärmer Futter sieht, so emittiert er ein blaues Nahbereichskommunikationssignal (Korrelationsplot (a), linke Zone und Grünkanal analog).

Als nächstes wird auf die Fortbewegung eingegangen: Ein Schwärmer bewegt sich durchgehend rückwärts fort, wobei diese Rückwärtsbewegung von verschiedenen Dingen inhibiert werden kann. Die Rückwärtsbewegung verlangsamt sich, wenn ein Schwärmer Futter sieht und wenn er blaue Nahbereichskommunikation wahrnimmt (Korrelationsplot (b), Rotkanal und rechte Zone analog, und (c), restliche Zonen analog).

Was die Orientierungsänderung angeht, ist folgendes festzustellen: Der Schwärmer dreht sich stark nach links, wenn er Futter sieht (Korrelationsplot (d), Grünkanal und linke Zone analog), was auch in Rotation auf der Stelle münden kann, wenn der Sensorwert hoch genug ist. Ebenfalls dreht der Schwärmer sich tendenziell schneller nach links, wenn er ein blaues Nahbereichkommunikationssignal wahrnimmt (Korrelationsplot (e), restliche Zonen analog). Da ein Schwärmer sich rückwärts fortbewegt, beschreibt er so Kreisbahnen gegen den Uhrzeigersinn um Futter bzw. bei Futterwahrnehmung. Da blaue Nahbereichssignale durch Schwärmer veranlasst werden, welche Futter sehen, dreht sich eine Schwärmergruppe auf Futter kontinuierlich stark nach links, während

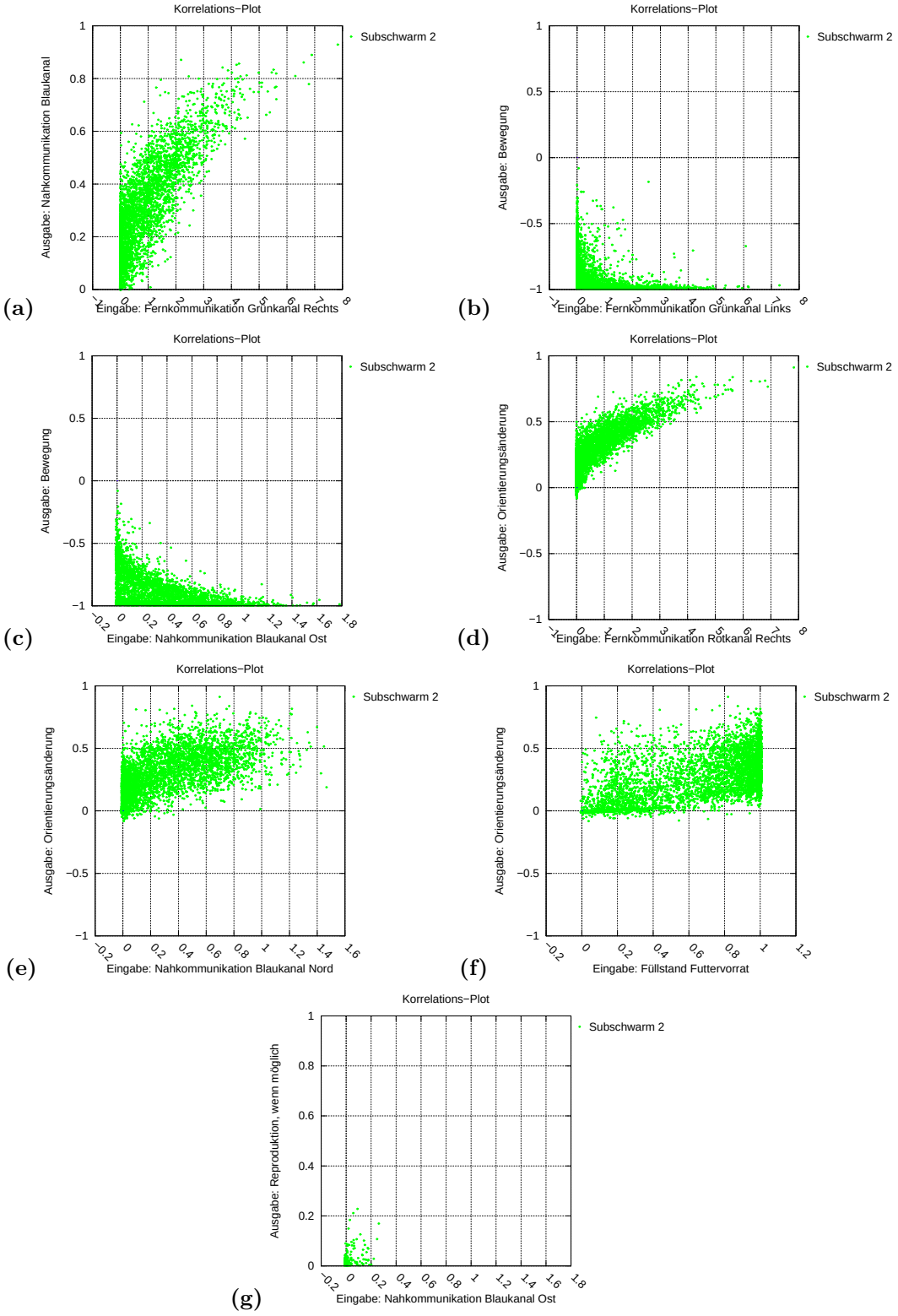


Abbildung 6.10: Korrelationsplots: Ergebnisbeispiel bei zwei scharf alternierenden, dynamischen Futterstellen.

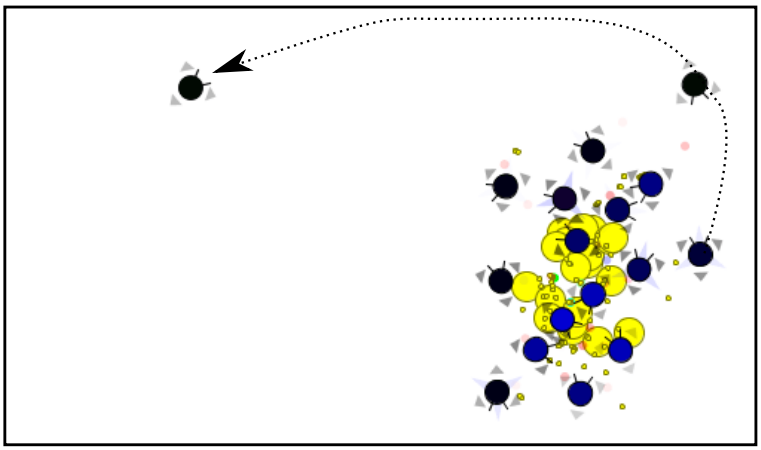


Abbildung 6.11: Schwarm mit Kundschafter im Experiment mit zwei scharf alternierenden, dynamischen Futterstellen. Zwei Kundschafter bewegen sich rückwärts vom Schwarm weg. Ihre ungefähre Bahn ist mit einem gepunkteten Pfeil skizziert. Auch zu sehen ist die blaue Nahbereichskommunikation des Schwarms im Zentrum der Futterstelle.

die Rückwärtsfortbewegung gehemmt wird. Hin und wieder – insbesondere, wenn das Futter zur Neige geht – schafft es ein Schwärmer, sich aus diesem Schwarm loszureißen, und wird zum „Kundschafter“: In diesem Fall bewegt er sich, da die Sensorreize immer weniger werden, immer schneller rückwärts vom Schwarm weg, wobei er Nahrung verbraucht. Mit sinkendem Füllstand des Futterspeichers nimmt zusätzlich die Geschwindigkeit der Rotation tendenziell ab (Korrelationsplot (f)). Die Bahn, welche der Kundschafter vom Schwarm weggehend beschreibt, ist also ungefähr eine Spiralbahn (Abb. 6.11) – und eine Spiralbahn ist, wie von Online-Bahnplanungstheoretikern nachgewiesen wurde, ein kompetitiver Weg, um in zweidimensionalen Räumen Objekte zu suchen [1]. Gleichzeitig hemmt das blaue Nahkommunikationssignal die Vermehrung (Korrelationsplot (g)), so dass mit dem Futter gehaushaltet wird – was bei der zeitlichen Verteilung erscheinenden Futters bei diesem Experiment noch viel wichtiger ist als beim letzten.

Deaktiviert man die Nahbereichskommunikation, so erfolgt erstens keine Geburtenkontrolle mehr und zweitens werden sofort alle Schwärmer zu Kundschaftern – der Schwarm vermehrt sich rasant, zerstreut sich und stirbt. Dies ist ebenfalls per Video dokumentiert.

6.5.2 Kurzbeschreibung weiterer entstandener Verhaltensweisen

Für eine weitere entstandene Schwarmverhaltensweise sind gleich vier Fertigkeiten von Wichtigkeit: Stigmergie (0,86), soziale Interaktionskraft (0,82), Schwärmerwahrnehmung (0,62) und Nahbereichskommunikation (0,53). Entsprechend komplex ist das beobachtete Verhalten. Die grundlegende Strategie dieser Schwärmer ist, im relevanten Teil der Welt kleine Futterportionen zu verteilen, die regelmäßig ausschwärmenden Kundschaftern sowohl als Wegweiser, als auch als Proviant dienen. Zusätzlich werden Plätze, an denen einmal Futter erschienen ist, mit grünen Pheromonen markiert, was die Schwärmer zum Warten animiert. Direkt an einer Futterstelle laufen die Schwärmer hintereinander kreisförmige Bahnen im Uhrzeigersinn, wobei sich jeder mittels Schwärmerwahrnehmung am Vordermann orientiert. Dabei stoßen sich die Schwärmer mittels der sozialen Interaktionskraft gegenseitig ab, was die individuelle Bewegungsfreiheit erhält. Wird von einem Schwärmer eine neue Futterstelle gefunden, so erfolgt eine kurze, sprunghafte Vermehrungsphase, die aber sofort endet, wenn „genug“ Schwärmer vorhanden sind (Geburtenkontrolle durch Schwärmerwahrnehmung und Nahbereichskommunikation). In regelmäßigen Abständen löst sich eine durch die Nahbereichskommunikation leuchtend rot kommunizierende, durch gegenseitige Anziehung mittels sozialer Interaktionskraft zusammenhängende Schwärmertraube von einem Schwarm an der Futterstelle und entfernt sich sehr schnell von dieser. Nach kurzer gemeinsamer Fortbewegung erlischt die rote Kommunikation, die Schwärmer lösen sich voneinander, bewegen

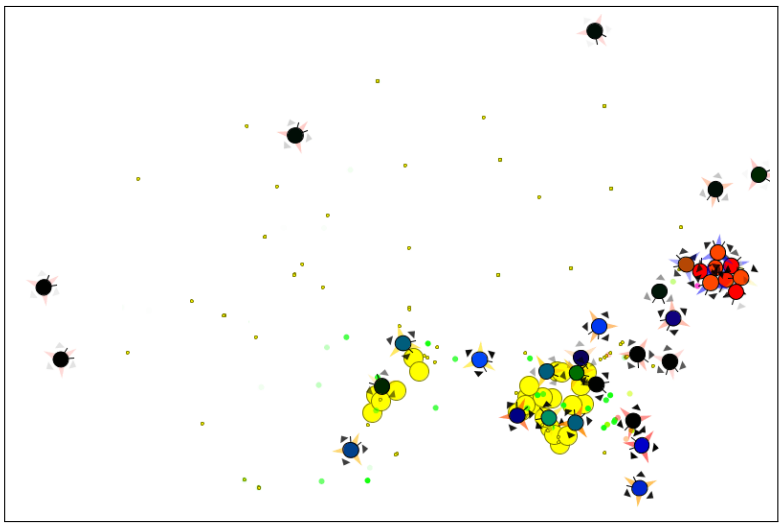


Abbildung 6.12: Illustration der im Text beschriebenen, weiteren entstandenen Verhaltensweise im Experiment mit zwei scharf alternierenden, dynamischen Futterstellen. In der oberen Hälfte des Bildes sieht man Kundschafter. Über die Welt hinweg sind kleine Futterobjekte verstreut. Gut zu sehen ist auch die Markierung von Futterstellen durch intensiv-grüne Pheromone. Von dem Schwarm der rechten Futterstelle hat sich, wie im Text beschrieben, eine Traube Kundschafter gelöst. Anhand der Pfeile der sozialen Interaktionskraft um die Schwärmer ist deutlich zu sehen, wie sich Mitglieder des Schwarms am Futter stark gegenseitig abstoßen, Mitglieder der rot kommunizierenden Kundschaftertraube sich jedoch teilweise stark anziehen. Die Traube ist zum Zeitpunkt des Bildes kurz vor ihrer Auflösung, daher stoßen sich auch hier schon einige Schwärmer gegenseitig ab.

sich in verschiedene Richtungen, verteilen kleine Futterobjekte in der Welt und sind gleichzeitig Kundschafter. Eine Illustration dieses Verhaltens findet sich in Abb. 6.12. Aufgrund des beeindruckenden Aussehens dieser Verhaltensweise findet sich eine Videoaufzeichnung dieses Verhaltens in Bonusvideo 1. Der Schwarm erreichte eine Referenzperformanz von 9,9.

6.5.3 Experimentvariation: Reduzierte Fertigkeitenpunkte

Der Experimentaufbau bleibt identisch, die Schwärmer haben jedoch nicht mehr 300, sondern nur 150 Fertigkeitenpunkte zu verteilen. Diese weiter verschärfte Variante des Zwei-Futterstellen-Experimentes brachte ebenfalls stabile Populationen hervor, wobei allerdings die durchnittliche Referenzperformanz mit 7,5 etwas niedriger lag als im vorherigen Experiment. Die Referenzperformanzen lagen zwischen den Werten 5,1 und 11,7. Wie im letzen Experiment gab es bei jedem entstandenen Verhalten Eigenschaften, welche essentiell wichtig waren. Besonders vertreten unter den wichtigen Eigenschaften waren erneut Stigmergie und Schwärmerwahrnehmung (jeweils 5×) – auch sonst waren die Ergebnisse sehr ähnlich zu denen, welche im Rahmen dieses Experiments schon präsentiert wurden. Was jedoch an Bedeutung gewann, war die soziale Interaktionskraft: Sie war bei 3 entstandenen Verhaltensweisen essentiell wichtig. In diesem Zusammenhang soll eine entstandene Verhaltensweise nicht unerwähnt bleiben, welche sich die soziale Interaktionskraft sehr zunutze macht. Es handelt sich hierbei um Schwärmer, welche sich sehr fest mittels sozialer Interaktionskraft zu einer dichten Gruppe zusammenschließen (Abb. 6.13 auf der folgenden Seite). Während der langsamen Bewegung dieser Gruppe über eine Futterstelle wird diese durch eine (bedingt durch die Anzahl der Schwämer sehr dichte) Pheromonspur markiert. Wird die Futterstelle nach und nach kleiner, so bewegt sich die Traube Schwärmer nicht nur über den mit Futter bedeckten Bereich der Welt fort, sondern auch über den mit Pheromonen markierten.

Mitglieder dieses Schwarms sind zufällig verteilt in alle Richtungen orientiert und beschleunigen stark, wenn sie Futter sehen. Erscheint nun eine neue Futterstelle, so wird sie anhand der Markierung mit Pheromonen mit hoher Wahrscheinlichkeit irgendwann von mehreren Schwärmern

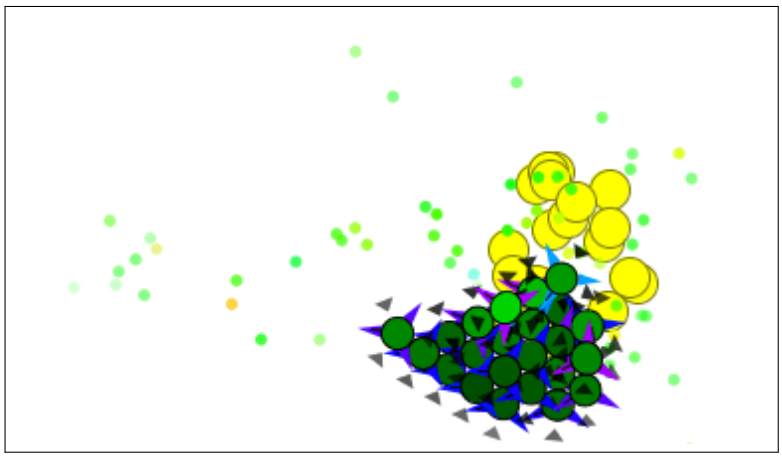


Abbildung 6.13: Illustration der im Text beschriebenen, entstandenen Verhaltensweise im Experiment mit zwei scharf alternierenden, dynamischen Futterstellen bei reduzierten Fertigkeitenpunkten.

gesehen. Diese beschleunigen dann stark in Richtung ihrer aktuellen Orientierung und ziehen durch die soziale Interaktionskraft die restlichen Mitglieder der Traube mit. Obwohl so zunächst viele Schwärmer kontinuierlich in der Welt leben, zeigt dieses Verhalten leider im letzten Drittel des Experiments keinen Erfolg mehr, da so keine größeren Distanzen überwunden werden können. Die Referenzperformanz dieses Verhaltens liegt bei 6,6. Aufgrund des beeindruckenden Aussehens dieser Verhaltensweise findet sich eine Videoaufzeichnung dieses Verhaltens in Bonusvideo 2.

6.6 Zwei scharf alternierende, entfernte Futterstellen

Der Experimentaufbau ist identisch mit dem vorherigen, bis auf die Tatsache, dass die Mittelpunkte der alternierenden Futterstellen statisch bei $(450; 0)$ bzw. $(-450; 0)$ festgelegt sind. Da die maximale Sichtweite der Schwärmer bei voller Ausprägung 400 beträgt, können die Schwärmer die Futterstellen initial knapp nicht sehen. Der Versuch wurde durchgeführt, um festzustellen, ob die Schwärmer z.B. Explorationsverhalten analog zum Experiment mit vier Futterstellen oder eine Spiralbahn als Suchstrategie entwickeln. Wie schon in Abschnitt 5.7.2 beschrieben, wird den Schwärmern hierzu ein gewisser Futtervorrat mit auf den Weg gegeben. Weitere Hilfen erfolgen nicht. Die Ausbildung einer Suchstrategie fand nicht statt. Kein Schwarm entwickelte Futtersuchverhalten. Der Diplomand nimmt an, dass dies der – im Vergleich zur Natur – kleinen Anzahl Schwärmer geschuldet ist, die in den Experimenten verwendet werden.

6.7 Pheromonstraße zwischen zwei scharf alternierenden Futterstellen (reduzierte Fertigkeitenpunkte)

Dieses Experiment modelliert eine komplexere Umwelt: Es verbindet zwei statisch positionierte, jedoch genau wie in den letzten Experimenten scharf alternierende Futterstellen mit einer „Straße“ von ca. 80 künstlich gesetzten Pheromonen. Diese entstehen im Bereich der Straße örtlich und zeitlich zufällig gleichverteilt (Abb. 6.14 auf der rechten Seite), so dass zu jedem Zeitpunkt ca. 80 Stück vorhanden sind. Zwischen der Pheromonstraße und dem Futter, sowie innerhalb der Pheromonstraße sind Lücken vorgesehen, die zusätzliche, zu überwindende Hindernisse darstellen, da es für die Schwärmer hier schwieriger ist, sich zu orientieren. Es werden also gewisse Ausprägungen von Pheromonwahrnehmung und Sehsinn vorausgesetzt. Zusätzlich dürfen nur 150 Fertigkeitenpunkte verteilt werden. Der Abstand vom Ursprung zum Beginn der Pheromonspur ist in beide Richtungen 50, so dass beim Entlanglaufen der Straße eine Lücke von 100 Einheiten

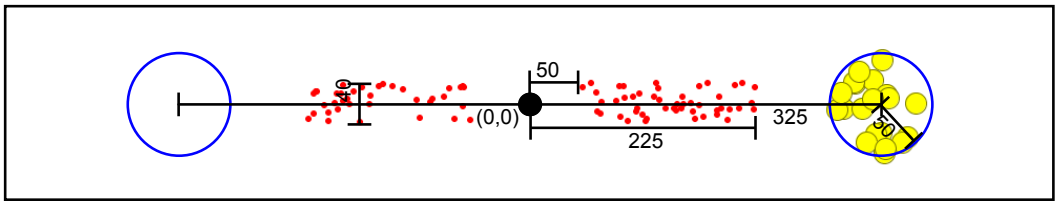


Abbildung 6.14: Typische Weltsituation beim Experiment mit einer kurzen Pheromonstraße zwischen zwei scharf alternierenden Futterstellen. Die beiden, vom Ursprung genau entgegengesetzt und waagerecht wegführenden Straßensegmente beginnen jeweils im Abstand von 50 zum Ursprung und enden beim Abstand 225. Sie haben eine Breite von 40 Einheiten. Die Mittelpunkte der beiden Futterstellen liegen im Abstand von 325 Einheiten zum Ursprung ebenfalls auf der Waagerechten durch den Ursprung.

Wichtige Subschwärme:	3 (volle Wichtigkeit)
Stark ausgebildete Fertigkeiten:	Keine besonderen Ausprägungen
Wichtige Fertigkeiten:	Stigmergie (0,81), Schwärmerwahrnehmung (0,77)
Referenzperformanz:	16
Kooperation:	Geburtenkontrolle durch Schwärmerwahrnehmung

Tabelle 6.6: Kurzzusammenfassung des Ergebnisbeispiels bei Pheromonstraße zwischen zwei scharf alternierenden Futterstellen mit reduzierten Fertigkeitenpunkten.

zu überwinden ist. Die beiden Straßenabschnitte enden ¡bei Entfernung 225 vom Ursprung. Die Mittelpunkte der Futterstellen liegen bei $(325; 0)$ bzw. $(-325; 0)$, wobei die Futterstellen einen Radius von 50 Einheiten haben. Die „Straßenbreite“ beträgt 40 Einheiten. Siehe Abb. 6.14 für das genaue Experiment-Layout.

Bei diesem Experiment erzeugten 7 der 10 evolvierten Schwärme stabiles Futtersuchverhalten mit weit gestreuten Referenzperformanzen zwischen 4,5 und 16 (durchschnittlich 10,4). Erwartungsgemäß wurde übergreifend die Eigenschaft „Stigmergie“ bei den entstandenen Schwärmen als wichtig erkannt. Eine weitere, sehr übergreifend wichtige Fertigkeit war die Schwärmerwahrnehmung (6× erreichte sie eine höhere Wichtigkeit als 0,7). Aufgrund seines sehr ausgeprägten und anschaulich analysierbaren Verhaltens wird der Schwarm mit Referenzperformanz 16 als ausführlich dokumentiertes Beispiel gewählt.

6.7.1 Beispiel entstandenen Verhaltens

Eine Kurzzusammenfassung wichtiger Größen in Bezug auf das in diesem Beispiel dokumentierte Verhalten findet sich in Tabelle 6.6. Zugehörige, relevante Korrelationsplots befinden sich in Abb. 6.15 auf der folgenden Seite.

Zunächst wird die Fortbewegung der Schwärmer beschrieben. Ein Schwärmer läuft standardmäßig vorwärts (Korrelationsplot (a)). Diese Vorwärtsbewegung wird sofort stark beschleunigt, wenn Futter in Sicht ist (Korrelationsplot (b), grüner Kanal und linke Zone analog). Auf der anderen Seite hemmen Wahrnehmungen roter Pheromone diese Fortbewegung und kehren sie mit steigender Intensität in Rückwärtsfortbewegung um (Korrelationsplot (c), restliche Zonen analog).

Die Orientierungsänderung besteht hingegen tendenziell in einer langsamen Drehung im Uhrzeigersinn (Korrelationsplot (d)). Zusammen mit der standardmäßigen Fortbewegung und dem Abbremsen in der Nähe roter Pheromone umläuft ein Schwärmer die Straße im Uhrzeigersinn: Er läuft so lange an der Straße „entlang“, bis er sich ihr aufgrund der kontinuierlich beschriebenen Rechtskurve zu sehr annähert, was seine Fortbewegung extrem verlangsamt. Folglich dreht er

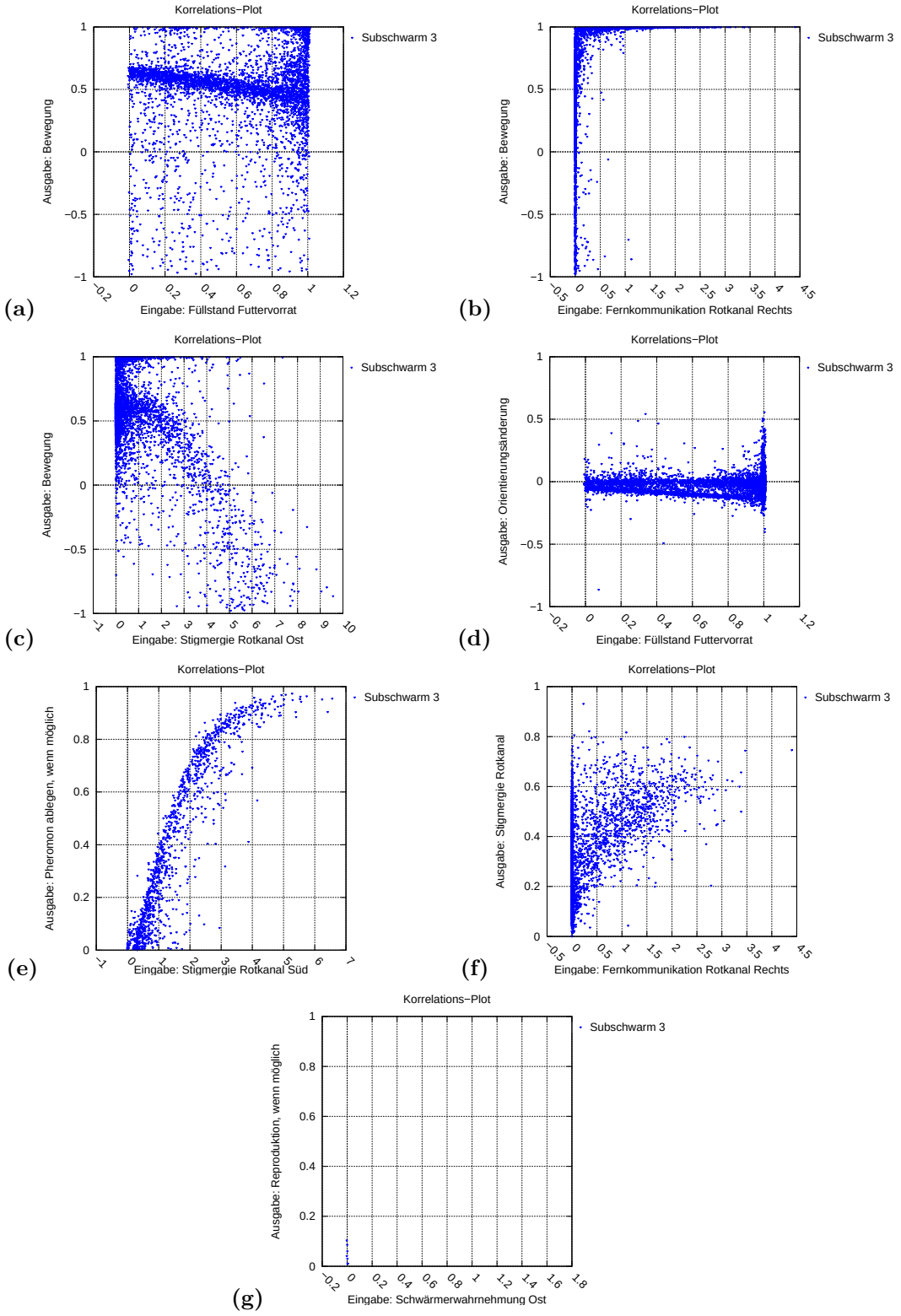


Abbildung 6.15: Korrelationsplots: Ergebnisbeispiel bei Pheromonstraße zwischen zwei scharf alternierenden Futterstellen mit reduzierten Fertigkeitenpunkten.

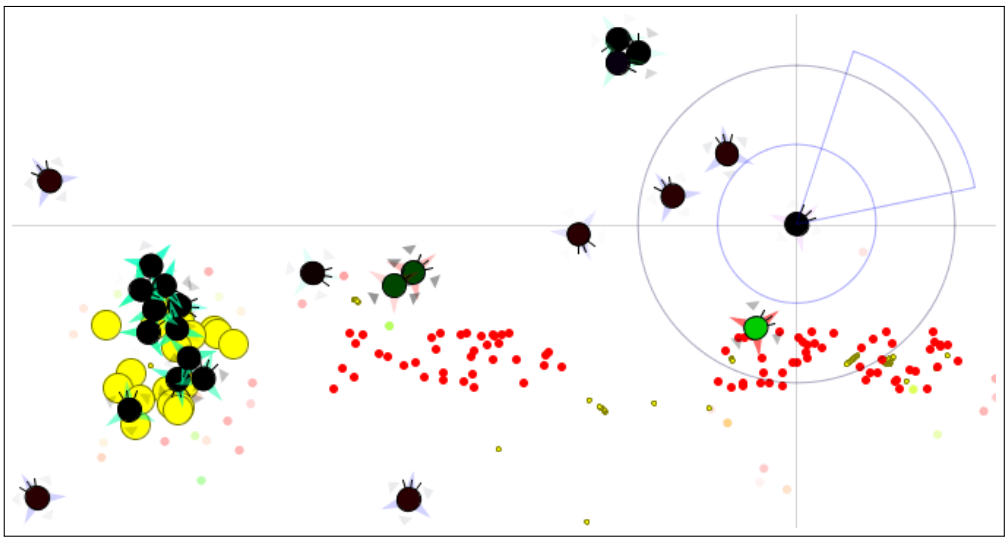


Abbildung 6.16: Schwärmer im Experiment mit Pheromonstraße zwischen zwei scharf alternierenden Futterstellen und reduzierten Fertigkeitenpunkten. Der äußere der beiden Kreise um den markierten Schwärmer veranschaulicht die Reichweite der Pheromonwahrnehmung. Gut zu sehen ist hier auch eine Ansammlung rötlicher Pheromone um die gerade gültige Futterstelle.

sich durch die standardmäßige langsame Drehung fast auf der Stelle. Wenn der Schwärmer nun wieder von der Straße wegorientiert ist, ist nur noch in der für ihn subjektiv südlichen Pheromonwahrnehmungszone ein Reiz gegeben, der nicht ganz so stark die Fortbewegung hemmt wie die anderen Zonen – der Schwärmer bewegt sich vorwärts, beschreibt wieder eine Rechtskurve, und wiederholt das Ganze von vorn. Über diesen Mechanismus wird entlang der Straße geforscht (Abb. 6.16), da sich der Schwärmer nach und nach entlang der gesamten Straße bewegt. Dieser Vorgang ist genau in einem Teilvideo des zu diesem Experiment gehörenden Videos dokumentiert. Zufällig (z.B. durch Kollisionen mit anderen Schwärmern) lösen sich auch Schwärmer von der Straße. Gleichzeitig ist das entlang der Straße stattfindende Explorationsverhalten robust gegen die Lücke in der Straße und ermöglicht, die Straße wieder zurückzulaufen, falls das Ende erreicht ist, an diesem aber kein Futter ist. Insbesondere auf den letzteren Aspekt wird im Video separat eingegangen.

Zusätzlich verstärken die Schwärmer die Straße, was auch die Überbrückung der Straßenlücke einschließt. Nehmen die Schwärmer rote Pheromone wahr, so werden tendenziell mehr Pheromone gelegt (Korrelationsplot (e)), die mehr Rotanteil erhalten, wenn Schwärmer Futter sehen (Korrelationsplot (f), grüner Kanal und linke Sichtzone analog). Die Straße wird also in Richtung des Futters „verlängert“.

Die Schwärmerwahrnehmung sorgt ähnlich wie bei vorher diskutierten Verhaltensweisen für eine Geburtenkontrolle. Dies ist aus Korrelationsplot (g) nur schemenhaft zu erkennen, wird aber im Video dokumentiert.

6.7.2 Experimentvariation: Verlängerte Straße

Im Rahmen dieser Experimentvariation wurde die Aufgabe der Futtersuche durch Vergrößern verschiedener Abstände deutlich erschwert. Der Abstand vom Ursprung zum Beginn der Pheromonspur wurde in beide Richtungen auf 70 erhöht, so dass nun insgesamt beim Entlanglaufen der Straße eine Lücke von 140 Einheiten zu überwinden ist. Die beiden Straßenabschnitte enden nicht mehr bei einer Entfernung von 225 vom Ursprung, sondern bei 300. Die Mittelpunkte der beiden Futterstellen liegen nun 450 Einheiten entfernt vom Ursprung und haben weiterhin einen Radius von 50. Die Lücke zwischen Pheromonspur und Beginn der Futterstelle hat sich also ebenfalls in ihrer Größe verdoppelt. Die „Straßenbreite“ ist bei 40 Einheiten geblieben.

Der erhöhte Schwierigkeitsgrad des Experiments äußerte sich zunächst in der Erfolgsquote: Nur 3 von 10 Evolutionen erzeugten stabiles Futtersuchverhalten, wobei die Referenzperformanzen zwischen 5,4 und 6,6 lagen (durchschnittlich 6,1). In allen drei Fällen war die Stigmergie essentiell wichtig (Wichtigkeit $> 0,8$), wobei jeweils Verhaltensweisen entstanden, welche der obigen – im Kontext der kürzeren Straße beschriebenen – sehr ähnlich sind. Stärkere Spezialisierungen auf bestimmte Metaparameter als beim Originalexperiment waren nicht festzustellen, so dass der gravierendste Unterschied bei den Ergebnissen mit kurzer und langer Straße einzig in der Erfolgsquote lag.

Im folgenden Kapitel sollen die Experimente und entstandenen Verhaltensweisen diskutiert und in den Kontext der Arbeit gesetzt werden.

Kapitel 7

Zusammenfassung und Diskussion

Das abschließende Kapitel dieser Arbeit soll in Kürze die wichtigsten Aspekte der durchgeführten Forschung beleuchten. In diesem Zusammenhang sollen auch mögliche Forschungswege aufgezeigt werden, die auf der entwickelten Software und der im Zuge der Arbeit gewonnenen Erfahrung aufbauen.

Bei der Einordnung und der Beschreibung der verwandten Arbeiten wurde festgehalten, dass Schwarmverhalten bis jetzt zumeist sehr zielgerichtet synthetisiert wurde – unabhängig davon, ob die Synthese traditionell oder durch evolutionäre Optimierung erfolgte. Ziele waren hier beispielsweise die Nachbildung des in der Natur beobachteten Verhaltens oder die automatisierte Ausbildung robotischer Systeme. Im Laufe der Diskussion existierender Arbeiten wurden verschiedene Aspekte identifiziert, welche die Möglichkeiten der evolutionären Verhaltensentstehung signifikant einschränken. Diese Einschränkungen sollen im folgenden Abschnitt rezitiert werden. Dort wird weiter dargestellt, wie den genannten Einschränkungen durch das in dieser Arbeit implementierte Softwareframework begegnet wurde. In Abschnitt 7.2 wird beschrieben, auf welcher Hardwareinfrastruktur das dies leistende Softwareframework betrieben wurde und welche Erfahrungen bezüglich seiner Performanz gemacht wurden. Im Anschluss daran werden in Abschnitt 7.3 entstandene Verhaltensweisen diskutiert und insbesondere wiederkehrende Mechanismen herausgestellt. Dies mündet schließlich in Abschnitt 7.4, in welchem mögliche Wege aufgezeigt werden, diese Experimente weiterzuführen. Abschnitt 7.5 zieht ein Fazit und schließt diese Arbeit.

7.1 Begegnung der bei existierenden Experimenten identifizierten Einschränkungen

Die in Abschnitt 2.3 identifizierten Einschränkungen der Möglichkeiten freier Verhaltensentstehung sind in der folgenden Aufzählung aufgelistet. Jeder Einschränkung wird gegenübergestellt, wie ihrem Einfluß im Rahmen dieser Arbeit begegnet wurde.

Sensorik, Aktorik und Morphologie von Individuen wurde bisher meist statisch definiert, also nicht mitevolviert. In dieser Arbeit werden Sensorik und Aktorik über Metaparameter von der Evolution maßgeblich beeinflusst (Abschnitt 4.4).

Kommunikations- und Interaktionsmöglichkeiten zwischen Individuen waren meist entweder nicht vorhanden (bis auf z.B. möglicherweise auftretende Kollisionen) oder stark eingeschränkt. Die für diese Arbeit modellierten Schwärmer verfügen hingegen über mehrkanalige Kommunikationsmöglichkeiten (Fernkommunikation, Nahkommunikation und Stigmergie, Abschnitt 4.2), welche grob gerichtet wahrgenommen werden können und sich in ihrer Handhabung deutlich voneinander unterscheiden.

Kontrollstrukturen der Individuen waren meist nur stark eingeschränkt evolvierbar, z.B. bei der reinen Evolution gewichteter Verbindungen zwischen Sensoren und Aktoren, bei Verwendung von Kontrollstrukturen ohne innere Zustände, der Verwendung von Single Layer

Perceptrons bzw. vergleichbarer neuronaler Modelle oder neuronalen Netzen mit statischer Topologie. Im Rahmen dieser Arbeit werden neuronale Netze mit beliebiger Topologie verwendet, insbesondere auch mit Rekurrenzen (siehe ebenso Abschnitt 4.2).

Homogenität: Meist wurden nur Schwärme mit homogenen Individuen berücksichtigt, was Effekte wie Spezialisierungen bestimmter Individuen völlig ausklammert. Über ein einfaches Spezialisierungsmodell mit drei potentiell unterschiedlichen Subschwärmen pro evolviertem Schwarm werden im Rahmen dieser Arbeit Spezialisierungen ermöglicht (Abschnitt 4.1). Im Rahmen des Abschnittes 7.3 wird noch darauf eingegangen, wie diese Spezialisierungsmöglichkeit genutzt wurde.

Speziell vorgegebene Evolutionsziele, z.B. robotische Systeme bestimmter Art zu erhalten oder aber partikuläre Aufgaben zu lösen (wie die Überwindung eines bestimmten Hindernisses), ließen meist keine allgemeine Verhaltensentwicklung zu. Im Rahmen der in dieser Arbeit durchgeführten Experimente ist nicht das Ziel, ein partikuläres Verhalten zu entwickeln, sondern vielmehr, eine stabile Population von Schwärmern unter verschiedenen Umweltbedingungen und insbesondere Konfigurationen von Futterstellen zu bilden (Abschnitt 4.4).

Umfang von Simulationen: Wenn eine simulationsbasierte Evaluationsfunktion im Rahmen der Evolution gewählt wurde, so wurde die Simulationszeit oft aus Gründen des Rechenaufwandes sehr kurz gewählt, da Evaluationsfunktionen naturgemäß sehr oft ausgeführt werden müssen. Im Rahmen dieser Arbeit werden simulationsbasierte Evaluationen erheblichen Umfangs durchgeführt (Abschnitt 5.7.2). Insbesondere ist die Simulationszeit signifikant länger als das Leben eines einzelnen Schwärmers. Dieser Aspekt ermöglicht erst, die Stabilität einer Population von Schwärmern über die Zeit als naturmotivierte Fitnessfunktion zu verwenden.

7.2 Hardwareinfrastruktur und Evolutionsdauer

Die beteiligten Rechner wurden vom Diplomanden bei Freunden, befreundeten Unternehmen sowie aus dem Institut für Informatik VI akquiriert. Dies impliziert eine gewisse Heterogenität der verwendeten Rechner in Bezug auf Rechenleistung und Anbindung. Die Rechner umfassen sowohl High-End-PCs mit 4 oder 8 CPUs, als auch kleine Büro- und Heimcomputer. Die Anbindungsgeschwindigkeit variierte von langsamen Internetverbindungen bis zu Gigabit-Anbindungen direkt über ein lokales Netzwerk.

Performanzvergleiche zwischen verschiedenen Simulationsframeworks oder gar hochspezialisierter Software wie des vorgestellten Systems zur verteilten Schwarmevolution sind aufgrund der unterschiedlichen Möglichkeiten, die jedes System bietet, äußerst schwierig durchzuführen. Zudem wird die Durchführungsgeschwindigkeit von Experimenten ganz maßgeblich von der Natur des speziellen Experiments bestimmt. Die Erfahrungswerte bezüglich der Performanz des implementierten Gesamtsystems sollen hier dennoch angegeben werden.

Wie schon beschrieben, wurde zur Evaluation eines Schwarmgenoms eine Simulation mit 70.000 Simulationsschritten eingesetzt. In einer solchen Simulation „lebten“ durchschnittlich ca. 10 bis 50 Schwärmer gleichzeitig in der Schwarmwelt, zuzüglich ggf. tausender passiver Objekte (Futter, Pheromone). Bedingt durch eine Populationsgröße von 250 Schwarmgenomen und 100 evolvierten Generationen pro Evolution, wurden pro Evolution 25.000 solcher Evaluationen, also $1,75 \cdot 10^9$ Simulationsschritte durchgeführt. Je nach Experiment und vorhandener Anzahl an beteiligten Computern dauerte ein Zehnfachdurchlauf an Evolutionen mit insgesamt $1,75 \cdot 10^{10}$ Simulationsschritten zwischen 12 Stunden und 4 Tagen. Dies entspricht einer effektiven Simulationsgeschwindigkeit zwischen ca. 50.000 und 400.000 durchgeführten Simulationsschritten pro Sekunde – jeweils unter Berücksichtigung physikalischer Gesetze.

Hierbei ist zu berücksichtigen, dass die Anzahl der beteiligten Rechner (und die Zahl der verfügbaren CPUs pro Rechner) über die Zeit hinweg variabel war, und die Evolution einen Großteil der

Zeit künstlich verlangsamt wurde (vgl. Abschnitt 5.2.8). Die Parallelisierung der Evolution war hierbei zu ca. 85%-90% effizient, es wurde also dieser Anteil der zur Verfügung gestellten Rechenkraft tatsächlich genutzt. Die restliche ungenutzte Zeit verstrich insbesondere für die Übertragungszeiten der zu evaluierenden Genome an die Rechenclients. Dies ist der Tatsache geschuldet, dass der Rechner, auf dem der Evolutionsserver lief, an einer verhältnismäßig upload-schwachen privaten Internetleitung angeschlossen war und daher viele (auch verhältnismäßig schnelle) Rechner nur über diese Verbindung erreichbar waren. Der Rechner wurde trotz der langsamen Anbindung als Evolutionsserver gewählt, da der Diplomand hier Verfügbarkeit garantieren konnte.

7.3 Entstandene Verhaltensweisen

Insgesamt wurden 90 Evolutionen durchgeführt, von denen 60 erfolgreiches Futtersuchverhalten hervorbrachten. In Kapitel 6 wurde dargestellt, wie vielfältig Verhaltensweisen sein können, die durch weniger beschränkte Schwarmevolutionen entstehen. Einige davon wurden ausführlich in Text, Bild und Video dokumentiert und mittels Korrelationsplots analysiert. Ein Diagramm, welches die Referenzperformanzen und Erfolgsquoten der durchgeführten Experimente miteinander in Relation setzt, findet sich in Abb. 7.1 auf der folgenden Seite. Es sei an dieser Stelle nochmals daran erinnert, dass die Schwärmer – bei optimaler Ausnutzung des verfügbaren Futters – eine maximale Referenzperformanz von ca. 20 erreichen können, wobei die Vermehrungskosten hier unberücksichtigt sind. Die Referenzperformanzwerte erreichter erfolgreicher Verhaltensweisen decken einen recht großen Teil des Bereichs der möglichen Fitnesswerte ab; es scheint gelungen zu sein, sich nach und nach an die Grenze der Leistungsfähigkeit des gegebenen beispielhaften Schwärmerorganismus und Evolutionsmodells heranzutasten.

7.3.1 Entstehung von kooperativen Verhaltensweisen

Zunächst ist festzuhalten, dass kooperative Mechanismen unter gleichartigen Individuen bei den gegebenen Modellen recht einfach evolvierbar zu sein scheinen: In jedem Experiment (sogar vereinzelt in demjenigen mit feiner Futtergranularität, bei welchem auch offensichtlich reines Individualverhalten zur Zielerreichung ausreichte) waren Fertigkeiten von essentieller Wichtigkeit, welche auf Individualebene wenig Sinn (z.B. Stigmergie) oder gar keinen Sinn (z.B. Nahbereichskommunikation, soziale Interaktionskraft, Schwärmerwahrnehmung) machen.

Im Rahmen der 60 erfolgreichen freien Evolutionen ist eine Fülle an kooperativen Verhaltensweisen entstanden. Von gegenseitiger Hemmung der Vermehrung zum Zwecke der Rationierung vorhandenen Futters, über ausgeprägtes Markierungs-, Aggregations- und Explorationsverhalten bis hin zu Kommunikationsmechanismen, welche einfacher Natur, jedoch für die Funktion des Gesamtschwarms unerlässlich waren, sind Verhaltensmerkmale entstanden, welche in der Natur allerorten beobachtet werden können. Es ist hier zu beachten, dass in einer relativ sterilen Welt und mit Schwärmern einfachen Körperbaus evolviert wurde, so dass tendenziell einfachere Verhaltensweisen stabil zum Erfolg führen können, als in der Natur. Dennoch stellen die gezeigten – mithilfe eines einfachen, exemplarischen Lebewesens- und Umweltmodells evolutionär entstandenen – Verhaltensweisen stellen das Potential von Experimenten mit freien, umfangreichen Evolutionen eindrucksvoll dar.

7.3.2 Entstehung von Spezialisierungen

Weiter ist zu bemerken, dass keines der Experimente Spezialisierungen hervorgebracht hat: Bei jeglichem entstandenen Verhalten überlebt jeweils nur ein Subschwarm. Es wurde in diesen Experimenten kein Verhalten festgestellt, bei dem Fähigkeiten *mehrerer unterschiedlicher* Subschwärme zur Zielerreichung relevant waren. Mit ansteigendem Schwierigkeitsgrad der Experimente sind nicht etwa Spezialisierungen entstanden, sondern der Schwarm war ganz einfach häufiger nicht erfolgreich. Dieser Zusammenhang zwischen Erfolgsquote und Referenzperformanz ist aus dem Diagramm in Abb. 7.1 klar zu erkennen.

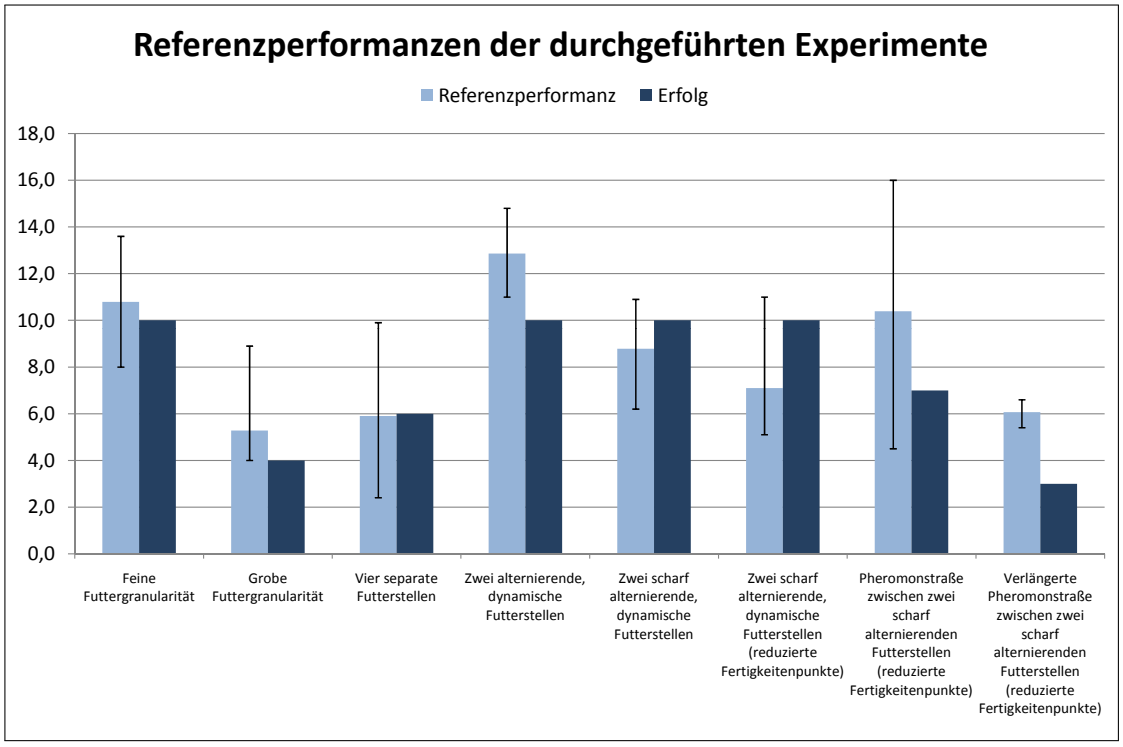


Abbildung 7.1: Referenzperformanzen der durchgeführten Experimente. Die Säulen der Datenreihe „Referenzperformanz“ geben die durchschnittlichen Referenzperformanzen der erfolgreichen Evolutionen des jeweiligen Experiments an. Die unteren bzw. oberen Enden der Fehlerindikatoren liegen beim jeweiligen Minimum und Maximum der Referenzperformanzen, über die der Durchschnitt gebildet wurde. Die Datenreihe „Erfolg“ gibt die Anzahl der Evolutionsdurchläufe an, die erfolgreich waren, also stabiles Futtersuchverhalten erzeugt haben. Offensichtlich müssen Werte dieser Datenreihe im Bereich von 0 bis 10 liegen. Das Experiment „Zwei scharf alternierende, entfernte Futterstellen“ war komplett unerfolgreich und findet in diesem Diagramm keine Berücksichtigung.

Es stellt sich die Frage nach der Ursache, warum auch bei Experimenten, in denen sie sich möglicherweise gelohnt hätten[1], keine Spezialisierungen stattgefunden haben. Es wäre denkbar, dass die Ursache hierfür in der exemplarischen, nicht naturmotivierten Modellierung eines Schwarm-Genoms liegt. Dies wurde schlicht als Tripel von unabhängigen Schwärmergenomen modelliert. Durch diese wurde die Dimensionszahl des Parameterraumes potentiell verdreifacht, die bei solch freien Evolutionen ohnehin schon groß ist. Auf diesen Aspekt und Lösungsansätze dazu wird in Abschnitt 7.4 noch eingegangen.

In welchem Rahmen Experimente wie die in dieser Arbeit ermöglichten und durchgeführten von großem Nutzen sein können und wo Potential für Erweiterungen liegt, soll ebenfalls im folgenden Abschnitt angesprochen werden.

7.4 Zukünftige Arbeiten und Erweiterungsvorschläge

Durch Begegnung der bisher vorhandenen Einschränkungen in der evolutionären Verhaltensentstehung von Schwärmen wird bei der Durchführung von weniger eingeschränkten Experimenten das Problem des nun stark vergrößerten Parameterraumes, der durch die Evolution abgesucht werden muss, offensichtlich. Insbesondere der exemplarischen, nicht naturmotivierten Modellierung eines Schwarm-Genoms als Tripel von unabhängigen Schwärmergenomen ist es nach Meinung des Diplomanden zuzuschreiben, dass es für die Evolution extrem schwierig zu sein schien, Spezialisierungen innerhalb eines Schwarms auszubilden. Diesem Problem des Parameterraums muss möglichst natürlich motiviert begegnet werden.

7.4.1 Generative Genomrepräsentationen

Ein – beispielhaft genannter – Ansatz hierfür ist die Verwendung generativer Genomrepräsentationen. Diese stellen grob gesagt evolvierbare Grammatiken da, durch welche die tatsächlichen Parameter der simulierten Wesen erst generiert werden. Dieser Ansatz wurde beispielsweise im Rahmen der evolutionären Robotik und Designautomatisierung von Hornby et al. verfolgt [28]. Er wird als naturinspiriert beschrieben, da er fundamentale Prinzipien beinhaltet, die auch natürlichen Genomen zugesprochen werden (beispielsweise Modularität, Skalierbarkeit und Wiederverwendung einzelner Mechanismen durch hierarchische Auswertung). Mittels dieses Ansatzes werden in der genannten Veröffentlichung komplexe robotische Strukturen mit einem (bei nicht-generativer Repräsentation) extrem großen Parameterraum auf einfache Weise evolutionär erzeugt. Es erwies sich hier als deutlich effizienter, evolutionäre Operatoren auf einer generativen Genomrepräsentation zu implementieren, als dies direkt auf dem Genom zu tun. Nicht zu unterschätzen ist allerdings der Aufwand, geeignete Grammatikbestandteile zur Modellierung der generativen Genomrepräsentation zu bestimmen.

7.4.2 Modellierung von Landmarken und Komparten

Als weiterer Schlüssel zur Evolution komplexen Verhaltens können Landmarken oder Komparten dienen, mittels derer sich die Modellorganismen orientieren und so definierte Orte zum Informations- oder Stoffaustausch auf einfache Weise finden und nutzen können. So wären die einzelnen Organismen nicht mehr darauf angewiesen, zufällig aufeinanderzutreffen und anschließend beieinander zu bleiben, sondern es gäbe eine Art „Treffpunkt“, wie er auch in der Natur bei vielen Schwärmen in Form eines Nestes vorhanden ist.

[1] Der Diplomand hätte beispielsweise Spezialisierungen beim Experiment mit der verlängerten Straße für möglich gehalten: Hier hätte von Vorteil sein können, wenn Schwärmer mit sehr ausgeprägtem Sehvermögen und andere mit sehr ausgeprägtem Pheromon-Folgeverhalten zusammenarbeiten, um die Lücke in der Straße und die Lücken zwischen Straße und Futterstellen zu vereinfachen.

7.4.3 Experimente mit direkt biologisch motivierten Modellorganismen und Umwelten

In dieser Arbeit wurden zur Demonstration des Systems Experimente mit fiktiven Organismen durchgeführt, wenngleich die meisten ihrer Eigenschaften biologisch motiviert waren. In der Zukunft sollen Experimente mit biologisch noch motivierteren Modellorganismen und zugehörigen Umwelteinflüssen (Wetter, Tag-Nacht-Rhytmus, Jahreszeiten) durchgeführt werden. Auf diese Weise könnte viel zum Feld der *synthetischen Biologie* beigetragen werden, indem man Schemata existierender Lebewesen evolutionär herausbildet und dann in den Kontext weiterer Wesen stellt, wie sie unter leicht veränderten Umweltbedingungen hätten entstehen können. Vorbilder auf verschiedenen Organismenebenen können hier bestimmte Arten von Einzellern, Insekten oder Säugetieren sein. Auch koevolutionäre Mechanismen zur Optimierung der Genome sind in diesem Zusammenhang denkbar.

7.4.4 Betrachtung der evolutionären Zeitachse

Bei der oben skizzierten Durchführung ähnlicher Experimente im Rahmen von Fragestellungen der synthetischen Biologie liegt es weiter nahe, nicht nur die evolutionär ermittelten Ergebnisse zu betrachten, sondern auch deren Werdegang über die Evolution hinweg. So könnte der evolutionäre Werdegang existierender Wesen synthetisiert und untersucht werden, der bei vielen Organismen nicht klar ist. Experimente solcher Art könnten also Anhaltspunkte liefern, welche Umweltbedingungen nötig sind für die Entstehung von Organismen, welche nach heutigem Stand noch nicht sicher in die evolutionäre Ordnung der Lebewesen eingeordnet werden können.

7.4.5 Technische Weiterentwicklung

Über die Implementierungsphase hinweg taten sich im Rahmen von ausgiebigen Tests einige Schwächen des zugrundeliegenden Physikframeworks *Phys2D* auf. Manche dieser Schwächen konnten vom Diplomanden direkt behoben werden. Andere lassen darauf schließen, dass die Gesamtarchitektur von *Phys2D* noch Optimierungspotential innehat. Ein Umstieg auf ein anderes Physikframework ist durchaus möglich, sollte aber aus naheliegenden Gründen nicht zur Laufzeit der Arbeit geschehen; gleiches gilt für eine Umstrukturierung von *Phys2D*. Technische Verbesserungsvorschläge wären also, entweder die Architektur *Phys2D* weiter zu optimieren, oder aber z.B. auf *JBox2D* umzusteigen. Ein weiterer Gewinn an Realismus wären Sichtbarkeitsmodelle für die Sensorik, die allerdings u.U. rechenaufwändig und nicht für alle Experimente vonnöten sind.

7.5 Schluss

Im Rahmen dieser Arbeit sind Erfahrungen und Software zur Durchführung umfangreicher Experimente und freier Evolutionen entstanden. Es wurde demonstriert, auf welche Weise umfangreichere und weniger beschränkte Evolutionen als bisher durchgeführt werden können, so dass vielfältige Schwarmverhaltensweisen entstehen.

Auf dieser Grundlage erscheint es möglich, das existente Verhalten von Schwärmen und auch anderen Kollektiven in den Kontext des „Verhaltens, wie es noch sein könnte“ zu setzen. Zusätzlich können, wie in Kapitel 5 skizziert, die einzelnen Softwarebestandteile auch unabhängig voneinander eingesetzt werden, z.B. in der Lehre.

Der Diplomand hofft, die gesammelten Erfahrungen und Softwarebestandteile wie oben skizziert im Kontext größerer Experimentfamilien über diese Arbeit hinaus einsetzen und weiterentwickeln zu können.

Anhang A

Ergänzungen zum Text

Folgend finden sich verschiedene Aspekte, welche aus Platzgründen oder Übersichtsgründen in den Anhang verschoben wurden.

A.1 Generatorfunktion der sozialen Interaktionskraft

Soziale Interaktionskräfte werden in der Literatur verwendet, um schwarmhafte Bewegungen modellhaft zu erklären [39, 49]. Sei s_1 der Schwärmer, welcher die soziale Interaktionskraft ausübt, und s_2 derjenige, auf dessen Körper sie aufgetragen wird. Die im Rahmen der Implementierung verwendete, von der Distanz d zwischen zwei Schwärmern abhängige Generatorfunktion $\mathrm{soz}(d)$ (Abb. A.1 auf der folgenden Seite) besitzt analog zu den in den genannten Arbeiten vorgestellten Funktionen einen Nulldurchgang. Der Definitionsbereich (hier nach oben durch das Optimum von β mit dem Wert 150 und nach unten durch den minimalen Abstand $d_{\min} = 16$ zwischen zwei Schwärmern begrenzt) wird also in zwei Intervalle geteilt, hier ungefähr an der Stelle $d = 52$. Abhängig davon, wie weit entfernt s_2 von s_1 ist, wirkt also eine attraktive oder repulsive Kraft auf s_2. Bei einem positiven Vorfaktor von $\mathrm{soz}(d)$ entsprechen anziehende Kräfte negativen Werten von $\mathrm{soz}(d)$, abstoßende positiven; bei negativem Vorfaktor umgekehrt. Ein Schwärmer s_1 kann also steuern, ob er mittels negativem Vorfaktor einen weit entfernten Schwärmer s_2 eher anzieht (Fall 1) oder mittels positivem Vorfaktor eher abstößt (Fall 2), wobei sich bei geringer werdender Distanz d zwischen s_1 und s_2 anziehende Kräfte in abstoßende verwandeln (Fall 1) oder abstoßende in anziehende (Fall 2).

Bei Fall 1 bilden die Schwärmer offensichtlich relativ stabile Distanzen zueinander: Der Schwarm bleibt beieinander, aber jedes Individuum hat noch Freiraum – die Kraft wirkt schwarmbildend. Bei Fall 2 ist keine solche Stabilität vorhanden, die Kraft wirkt schwarmzerstörend. Die in Abb. A.1 dargestellte Funktion wird durch den Term

$$\mathrm{soz}(d) \quad = \quad \frac{100}{(d-16)^2} - \frac{25}{d-16} + \frac{1}{2}$$

definiert und wurde nach dem Vorbild der genannten Veröffentlichungen von Hand auf die in dieser Diplomarbeit durchgeführten Experimente zugeschnitten. Dem Realismus tut dieses händische Design keinen weiteren Abbruch, da soziale Interaktionskräfte an sich nicht natürlich motiviert sind.

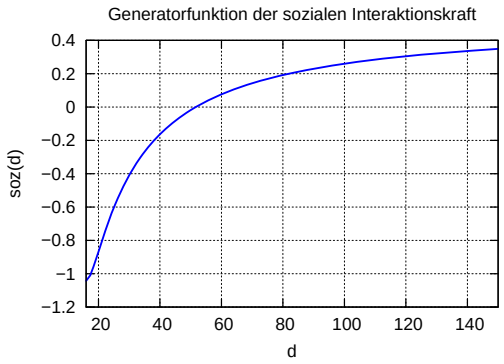


Abbildung A.1: Plot der Generatorfunktion der sozialen Interaktionskraft soz(d).

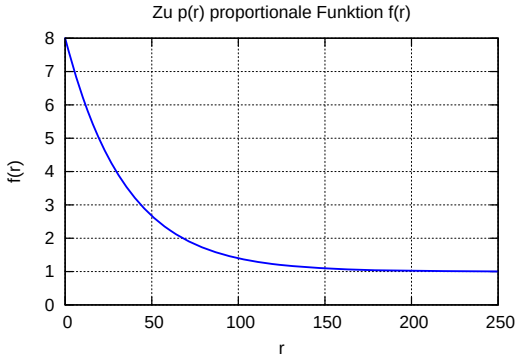


Abbildung A.2: Plot der Funktion $f(r)$, die zur Wahrscheinlichkeitsverteilung $p(r)$ der rangbasierten Elternauswahl proportional ist.Der Rang r ist naturgemäß begrenzt durch die Populationsgröße popsize. Im Rahmen dieser Erläuterung wurde absichtlich die Funktion $f(r)$ gewählt, um den durch s gegebenen Zusammenhang zwischen den Wahrscheinlichkeiten, mit denen das beste bzw. schlechteste Individuum gewählt wird, anschaulicher darzustellen.

A.2 Wahrscheinlichkeitsverteilung der rangbasierten Elternauswahl bei Evolutionen

Als vom Rang r eines Individuums abhängige Wahrscheinlichkeitsverteilung $p(r)$, als Elter gewählt zu werden, wurde eine Wahrscheinlichkeitsverteilung proportional zur Funktion

$$f(r) \quad = \quad (1 + (s-1) \cdot \mathrm{e}^{-(s-1)\cdot\frac{r}{\mathrm{popsize}}})$$

gewählt. Hierbei sei s der selektive Druck (im Rahmen dieser Arbeit gilt $s = 8$) und popsize die Populationsgröße (im Rahmen dieser Arbeit gilt popsize $= 250$). Der selektive Druck s beschreibt das Verhältnis der Wahrscheinlichkeiten zwischen den beiden Individuen mit höchstem und niedrigsten Rang: Es ist s-mal wahrscheinlicher, das beste Individuum zu wählen, als das schlechteste. Die Funktion $f(r)$ ist für $s = 8$ und popsize $= 250$ in Abb. A.2 aufgetragen.

Register der Abbildungen

Register der Tabellen

Literaturverzeichnis

[1] S. Alpern and S. Gal. *The theory of search games and rendezvous*. Kluwer Academic Publishers, 2003.

[2] C. Anderson and N.R. Franks. Teams in animal societies. *Behavioral Ecology*, 12(5):534–540, 2001.

[3] S. Arunachalam, R. Zalila-Wenkstern, and R. Steiner. Environment mediated Multi Agent Simulation Tools–A Comparison. *Engineering Environment-Mediated Coordination in Self-Organizing and Self-Adapting Systems, ECOSOA 2008*, page 57, 2008.

[4] R. Beckers, J. Deneubourg, and S. Goss. Trail laying behaviour during food recruitment in the antLasius niger (L.). *Insectes Sociaux*, 39(1):59–72, 1992.

[5] R. Belew, J. McInerney, and N.N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. *Artificial Life*, 2:511–547, 1992.

[6] J. Bongard, V. Zykov, and H. Lipson. Resilient Machines Through Continuous Self-Modeling, 2006.

[7] Stefan Bornholdt and Dirk Graudenz. General asymmetric neural networks and structure design by genetic algorithms. *Neural Networks*, 5(2):327 – 334, 1992.

[8] G. Bracha. Generics in the Java Programming Language. *Tutorial, Sun Microsystems, Mar*, 2004.

[9] S. Camazine, J. Deneubourg, N.R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[10] N. Campbell. *Biology*. Pearson Education, San Francisco, 2008.

[11] E. Catto. Box2d. Online. http://www.box2d.org.

[12] I. Couzin and NR Franks. Self-organized lane formation and optimized traffic flow in army ants. In *Proc. R. Soc. Lond. B*, volume 270, pages 139–146, 2003.

[13] M. Crichton. *Prey*. HarperCollins, 2002.

[14] C. Darwin. On the Origin of Species by Natural Selection. *Murray, London, Great Britain*, 1859.

[15] P. Deutsch. RFC1952: GZIP file format specification version 4.3. *Internet RFCs*, 1996.

[16] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 26(1):29–41, 1996.

[17] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[18] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T.H. Labella, G. Baldassarre, S. Nolfi, J. Deneubourg, F. Mondada, D. Floreano, et al. Self-Organizing Behaviors for a Swarm-Bot. *Autonomous Robots*, 17(2):223–245, 2004.

[19] D.B. Fogel, L.J. Fogel, and V.W. Porto. Evolving neural networks. *Biological Cybernetics*, 63(6):487–493, 1990.

[20] E.R. Gansner. Drawing graphs with GraphViz. Technical report, Technical report, AT&T Bell Laboratories, Murray, 2007.

[21] S. Garnier, J. Gautrais, and G. Theraulaz. The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31, 2007.

[22] D. Geer. Eclipse becomes the dominant Java IDE. *Computer*, 38(7):16–18, July 2005.

[23] K. Glass. Phys2d – the 2d game physics engine in java. Online. http://www.cokeandcode.com/phys2d/.

[24] H. Haken and A. Wunderlin. *Synergetik: eine Einführung: Nichtgleichgewichts-Phasenübergänge und Selbstorganisation in Physik, Chemie und Biologie*. Springer, 1983.

[25] P.J.B. Hancock. Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification. In *Proceedings of the International Workshop on Combinations of genetic algorithms and neural networks*, pages 108–122, 1992.

[26] J. Handl and B. Meyer. Ant-based and swarm-based clustering. *Swarm Intelligence*, 1(2):95–113, 2007.

[27] B. Hölldobler and EO Wilson. *The Ants*. Belknap Press of Harvard University Press, 1990.

[28] G. Hornby, H. Lipson, and J. Pollack. Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719, 2003.

[29] S. Jha, R.G. Casey-Ford, J.S. Pedersen, T.G. Platt, R. Cervo, D.C. Queller, and J.E. Strassmann. The queen is not a pacemaker in the small-colony wasps Polistes instabilis and P. dominulus. *Animal Behaviour*, 71(5):1197–1203, 2006.

[30] E.R. Kandel, J.H. Schwartz, and T.M. Jessell. *Principles of Neural Science*. Appleton & Lange, 2000.

[31] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks, 1995. Proceedings.*, volume 4, 1995.

[32] D.L. Kirk. *Volvox: Molecular-genetic Origins of Multicellularity and Cellular Differentiation*. Cambridge University Press, 1998.

[33] M. Krieger, J.B. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature (London)*, 406(6799):992–995, 2000.

[34] D. Kriesel. Ein kleiner Überblick über Neuronale Netze. Online – dkriesel.com, Bonn, Germany, Feb 2008.

[35] D. Kriesel, E. Cheung, M. Sitti, and H. Lipson. Beanbag Robotics: Robotic Swarms with 1-DoF Units. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A. Winfield, editors, *ANTS Conference*, volume 5217 of *LNCS*, pages 267–274. Springer, 2008.

[36] D. Kriesel, M. Sitti, and H. Lipson. Beanbag Robotics: Robotic Swarms with 1-DoF Units. *Swarm Intelligence*, 2008. In review.

[37] P. Kuntz, D. Snyers, and P. Layzell. A Stochastic Heuristic for Visualising Graph Clusters in a Bi-Dimensional Space Prior to Partitioning. *Journal of Heuristics*, 5(3):327–351, 1999.

[38] R. Laughlin and H. Reuter. *Abschied von der Weltformel: Die Neuerfindung der Physik*. Piper, 2009.

[39] C. Lee, M. Hoopes, J. Diehl, W. Gilliland, G. Huxel, E. Leaver, K. McCann, J. Umbanhowar, and A. Mogilner. Non-local Concepts and Models in Biology. *Journal of Theoretical Biology*, 210(2):201–219, 2001.

[40] H. Lipson and J. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(8):974–978, 2000.

[41] S. Luke. ECJ Evolutionary Computation System. Online, 2002. http://cs.gmu.edu/ eclab/projects/ecj/.

[42] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. MASON: A Multiagent Simulation Environment. *Simulation*, 81(7):517, 2005.

[43] M. Maeterlinck. *The Life of the White Ant*. Allen & Unwin, London, Great Britain, 1927.

[44] P. Marrone. Joone – java object oriented neural engine. Online, 2007. http://www.jooneworld.com/.

[45] K. Meffert. Java genetic algorithms package. Online, 2008. http://jgap.sourceforge.net/.

[46] Sun Microsystems. Javadoc 1.5.0. Online, 2004. http://java.sun.com/j2se/javadoc/.

[47] M.L. Minsky and S.A. Papert. *Perceptrons: expanded edition*. MIT Press Cambridge, MA, USA, 1988.

[48] M. Mitchell. *An Introduction to Genetic Algorithms*. Bradford Books, 1996.

[49] H.S. Niwa. Self-Organizing Dynamic Model of Fish Schooling. *Journal of theoretical biology*, 171(2):123–136, 1994.

[50] B. Partridge. The structure and function of fish schools. *Sci Am*, 246(6):114–23, 1982.

[51] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization, an Overview. *Swarm Intelligence*, 1(1):33–57, 2007.

[52] J.B. Pollack and H. Lipson. The GOLEM project: Evolving hardware bodies and brains. In *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 37–42, 2000.

[53] quixote arg and ewjordan. Jbox2d. Online. http://www.jbox2d.org.

[54] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34. ACM Press New York, NY, USA, 1987.

[55] David E. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP research group., editors, *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations*. MIT Press, 1986.

[56] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[57] F. Schätzing. *Der Schwarm: Roman*. Verlag Kiepenheuer & Witsch GmbH & Co KG, 2004.

[58] T.D. Seeley and P.K. Visscher. Group decision making in nest-site selection by honey bees. *Apidologie*, 35(2):101–116, 2004.

[59] E. Tatara, M. North, T. Howe, NT Collier, and JR Vos. An introduction to RePast simphony modeling using a simple predator-prey example. In *The Agent 2006 Conference on Social Agents: Results and Prospects. Chicago, IL: Argonne National Laboratory and the University of Chicago*, 2006.

[60] G. Theraulaz and E. Bonabeau. Modelling the Collective Building of Complex Architectures in Social Insects with Lattice Swarms. *Journal of Theoretical Biology*, 177(4):381–400, 1995.

[61] G. Theraulaz, J. Gautrais, S. Camazine, and J. Deneubourg. The formation of spatial patterns in social insects: from simple behaviours to complex structures. *Philos Transact A Math Phys Eng Sci*, 361(1807):1263–82, 2003.

[62] V. Trianni and M. Dorigo. Self-organisation and communication in groups of simulated and physical robots. *Biological Cybernetics*, 95(3):213–231, 2006.

[63] J.S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[64] Christopher R. Ward, Fernand Gobet, and Graham Kendall. Evolving collective behavior in an artificial ecology. *Artif. Life*, 7(2):191–209, 2001.

[65] J. Watson and F. Crick. Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–8, 1953.

[66] C. Wei, Z. He, Y. Zhang, and W. Pei. Swarm directions embedded in fast evolutionary programming. In *Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC'02.*, volume 2, 2002.

[67] D. Whitley. Genetic algorithms and neural networks. *Genetic Algorithms in Engineering and Computer Science*, pages 203–216, 1995.

[68] E.O. Wilson and Harvard University. *The insect societies*. Belknap Press of Harvard University Press Cambridge, Mass, 1971.

[69] X. Yao. A review of evolutionary artificial neural networks. *International journal of intelligent systems*, 8(4):539–567, 1993.

[70] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[71] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, May 1997.

[72] A. Zell. *Simulation neuronaler Netze*. Addison-Wesley, 1994.