Universität Paderborn

Fakultät für
Elektrotechnik, Mathematik und Informatik

Diplomarbeit

Strukturbildung durch koordinierte und evolutionäre Partikelschwärme

Andreas Goebels

vorgelegt bei

Prof. Dr. Hans Kleine Büning

und

Dr. habil. Benno Stein

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte Hilfe Dritter angefertigt habe. Alle Stellen, die inhaltlich oder wörtlich aus anderen Veröffentlichungen stammen, sind kenntlich gemacht. Diese Arbeit lag in gleicher oder ähnlicher Weise noch keiner Prüfungsbehörde vor und wurde bisher noch nicht veröffentlicht.

Andreas Goebels

Paderborn, 23. Februar 2003

Danksagung

Diese Diplomarbeit entstand im Rahmen meines Informatikstudiums an der Universität Paderborn. Sie wurde durch Prof. Dr. Hans Kleine Büning und Dipl.-Inform. Alexander Weimer betreut. Für ihre Hilfen und Ratschläge bei der Erstellung der Arbeit und die zur Verfügung gestellten Hilfsmittel bin ich sehr dankbar. Besonderen Dank sage ich allen, die bei der orthographischen Überarbeitung dieses Textes geholfen und nützliche Hinweise gegeben haben. Allen anderen, die zum Erfolg dieser Diplomarbeit beigetragen haben, gilt ebenfalls mein herzlicher Dank.

Andreas Goebels

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit der Erzeugung von geordneten Strukturen durch Partikelschwärme. Sie wurde motiviert durch die komplexen und funktional sehr erfolgreichen Nester diverser in Gruppen lebender Insektenspezies, zum Beispiel Termiten- und Ameisenpopulationen. Es wird ein Überblick über vorhandene Arbeiten aus diesem Bereich gegeben. Die beiden Hauptthesen, die untersucht werden, beschäftigen sich mit der Fähigkeit von Stigmergy als Kommunikationsmittel und den Möglichkeiten automatischer Regelgenerierungen durch evolutionäre Algorithmen.
Die Versuche von Bonabeau u.a. werden als Grundlage für ein Softwarepaket genommen, welches mit Hilfe von evolutionären Anpassungen von Schwärmen arbeitet, um Strukturen zu erzeugen. Dabei wird untersucht, ob die Multiagenten-Simulationsumgebung *SWARM* für diese und gegebenenfalls weitere Arbeiten in diesem Gebiet sinnvoll einzusetzen ist.

Inhaltsverzeichnis

1. Einleitung **1**

2. Grundlagen **4**

2.1. Stigmergy 4
2.1.1. Quantitative Stigmergy 5
2.1.2. Qualitative Stigmergy 6
2.2. Grundbegriffe 6
2.2.1. Agent 6
2.2.2. Schwarm 7
2.2.3. Individuum 8
2.2.4. Population 8
2.3. Selbstorganisierende Systeme 8
2.4. Strukturbildung durch Stigmergy - Selbstorganisation 10
2.5. Schwarm-Intelligenz 12
2.5.1. Partikelschwärme 13
2.5.2. Evolutionäre Partikelschwärme 14
2.6. Abgrenzung von Nachbargebieten 15
2.6.1. Abgrenzung von der klassischen KI 15
2.6.2. Abgrenzung von Multiagentensystemen 16
2.6.3. Abgrenzung von zellulären Automaten 16
2.6.4. Schwarm Intelligenz Algorithmen vs. klassische sequentielle Algorithmen 17

3. Strukturbildung nach Bonabeau **19**

3.1. Modell 19
3.1.1. Modell-Parameter 20
3.1.2. Generierung des Agenten-Algorithmus 20
3.1.3. Generierte Strukturen 21
3.1.4. Analyse 21
3.2. Suchmethoden 22
3.2.1. Koordinierte Suche 22
3.2.2. Zufällige Suche 23
3.2.3. Evolutionäre Suche 23
3.3. Erweitertes Modell 24

4. VisualSwarm **26**
4.1. Einleitung . . . 26
4.2. Simulationsaufbau . . . 27
4.2.1. Server-Worker Architektur . . . 27
4.2.2. MicroRule . . . 28
4.2.3. ConstructionGraph . . . 29
4.3. Suchmethoden . . . 30
4.3.1. Zufällige Suche . . . 30
4.3.2. Vollständige Suche . . . 31
4.3.3. Fokussierte Suche . . . 31
4.3.4. Evolutionäre Suche durch genetische Algorithmen . . . 32
4.3.4.1. Bewertungsfunktion *Strukturfindung* . . . 32
4.3.4.2. Bewertungsfunktion *Flaschenhals-Findung* . . . 36
4.3.4.3. Bewertungsfunktion *PatternMatching* . . . 36
4.3.4.4. Genetische Operatoren . . . 38
4.3.4.5. Selektion . . . 38
4.3.4.6. Crossover . . . 39
4.3.4.7. Mutation . . . 41
4.3.5. „Simulated Annealing“ Suche . . . 41
4.4. Programmsteuerung und Hilfsmittel . . . 42
4.4.1. Graphical User Interface - ThreadObserver . . . 42
4.4.2. Hilfsmittel . . . 43
4.4.3. Die SWARM Simulationsumgebung . . . 45
4.4.3.1. Elemente von SWARM . . . 46
4.4.3.2. Simulationen mit SWARM . . . 47
4.4.3.3. Probleme beim Einsatz von SWARM . . . 47

5. Simulationsergebnisse **49**
5.1. Ergebnisse von Bonabeau . . . 49
5.1.1. Koordinierte Suche . . . 49
5.1.2. Evolutionäre Suche . . . 50
5.2. Ergebnisse mit VisualSwarm . . . 51
5.2.1. Koordinierte Suche . . . 51
5.2.2. Evolutionäre Suche . . . 52
5.2.3. Zielgerichtete Suche . . . 53
5.2.4. Simulated Annealing Suche . . . 54

6. Fazit und Ausblick **56**

A. XML-Steuerdatei **59**

B. Konfigurationsdatei-Beispiel **66**

C. POV-Ray Beschreibungsdateien **69**
C.1. Szenenbeschreibung 69
C.2. Sequenzbeschreibung 71

D. Kommandozeilen-Parameter **72**

Literaturverzeichnis **75**

Abbildungsverzeichnis

1.1. Beispiel für einen Termitenbau 3

2.1. Schematische Darstellung der verwendeten Begriffe 7
2.2. Einordnung des Komplexitätsbegriffes 10

3.1. Generierte Strukturen 25

4.1. Aufbau der verteilten Berechnung: Server-Worker Prinzip 27
4.2. Anwendung einer einzelnen MicroRule 28
4.3. Beispiel für einen ConstructionGraph 30
4.4. Strukturfindung im ConstructionGraph 34
4.5. Die GUI des Servers 43
4.6. Schwarm-Hierarchien 46

5.1. Vergleich einer natürlichen mit einer künstlichen Struktur 49
5.2. Vergleich von manuell und evolutionär erzeugten Strukturen 51
5.3. Strukturen aus manuell erzeugten Regelmengen 52
5.4. Strukturen aus evolutionär erzeugten Regelmengen 53
5.5. Strukturen aus zielgerichtet erzeugten Regelmengen 54
5.6. Fitnessentwicklung bei Suche mit Simulated Annealing 55

1. Einleitung

In der Natur sind Bauwerke von Insektenarten bekannt, zum Beispiel die Nester von bestimmten Termiten *(Macrotermes bellicosus)*, die im Vergleich zur Körpergröße eines einzelnen Exemplares der Spezies riesige Ausmaße erreichen; das Verhältnis von Individuumgröße zur Nestgröße kann dabei Werte von 10^4 bis 10^5 erreichen. Ähnliche Verhältnisse sind nur bei Menschen bekannt. Außerdem verfügen die Bauwerke über ein ausgeklügeltes Belüftungssystem mit sehr konstanten Temperaturen (Schwankungsbereich +- 0,1°C). In Abbildung 1.1 ist ein solcher Bau dargestellt.

Die Frage, die sich stellt, ist, wie eine Koordination eines solchen riesigen Bauvorhabens durchgeführt wird. Grassé hat 1959 [Gra59] bei der Untersuchung solcher Spezies herausgefunden, dass es keine übergeordnete Instanz gibt, die das gesamte Bauvorhaben koordiniert, und dass nicht einmal direkte Kommunikation zwischen den einzelnen Individuen auftritt. Vielmehr wird die Umwelt als Kommunikations- und Speichermedium genutzt, indem sie die einzelnen Individuen zu Handlungen stimuliert (Stigmergy). Von ihm wurde jedoch nicht untersucht, wie diese externe Stimulierung zu komplexen, geordneten Strukturen führen kann.

Eric Bonabeau und andere Mitarbeiter des Santa Fe Instituts setzten an dieser Stelle an [BGS+00], indem sie ein einfaches Multi-Agenten Modell zur 3-dimensionalen Strukturbildung konstruierten. Dieses System reagiert lediglich auf begrenzte, lokale Informationen aus der Umwelt, auch hier existiert keine übergeordnete Instanz und keine direkte Kommunikation unter den Agenten. Da der Raum möglicher Algorithmen zur Strukturbildung sehr groß ist, wurde ein evolutionärer Algorithmus eingesetzt, der diesen Raum effizient durchsuchen soll. Die grösste Schwierigkeit stellte dabei das Bestimmen einer geeigneten Fitnessfunktion dar, um die einzelnen Strukturen zu bewerten.

Hier setzt diese Diplomarbeit an, sie gibt einen Überblick über den Themenkomplex und vollzieht die Ergebnisse von Bonabeau nach. Zu diesem Zweck wurde ein Softwaresystem entwickelt, welches vergleichbar zu dem System von Bonabeau arbeitet, an einigen Stellen jedoch zusätzlich eine erweiterte Funktionalität bietet. Genutzt wurde als Hilfsmittel die von Bonabeau u. a. entwickelte Simulationsumgebung SWARM. Dabei wurde untersucht, ob ein Einsatz dieser Umgebung für auf diese Arbeit aufbauende weitere Untersuchungen geeignet ist.

In diesem Dokument wird zuerst ein Überblick über grundlegende Eigenschaften von Partikelschwärmen und über den Themenkomplex der Stigmergy gegeben, als Vergleich werden dazu biologische Systeme herangezogen. Gleichzeitig wird eine Ab-

grenzung zu ähnlichen, benachbarten Gebieten der Informatik vorgenommen. Danach wird das Vorgehen von Bonabeau bei Durchführung seiner Versuchsreihen beschrieben. Dieses wird in Kapitel 4 in einem dazu entwickelten Softwaresystem nachvollzogen und die Ergebnisse von Bonabeau werden überprüft. Am Ende werden einige Simulationsergebnisse vorgestellt und die Resultate diskutiert (Kapitel 5 und 6).
Überprüft werden soll in dieser Arbeit einmal, ob das Konzept der Kommunikation ausschließlich mit Hilfe von Stigmergy ausreicht, natürliches Verhalten und Fähigkeiten von sozialen Spezies zu beschreiben. Die zweite These, die untersucht wird, ist die Möglichkeit, eine Regelmenge zu generieren, welche ein Bauvorhaben koordiniert. Hierbei wird besonders auf die evolutionäre Erzeugung eingegangen und diese wird mit anderen Möglichkeiten verglichen.

Die Ergebnisse aus diesen Untersuchungen können in den unterschiedlichsten Bereichen weiterverwendet werden. Einmal geben Untersuchungen über Stigmergy und die Auswirkungen von Stigmergy auf das Verhalten von Gruppen von Individuen Einblick in die biologischen Zusammenhänge, die bei einzelnen Exemplaren von sozialen Spezies ablaufen. Aber die Ergebnisse aus der Biologie sind auch gut auf andere, informationstechnische Probleme übertragbar [HM], wenn man beispielsweise den Bereich der Wegsuche bei Ameisen betrachtet, der schon erfolgreich auf Problemfelder in der Telekommunikation angewandt worden ist [BM01].
Speziell der Bereich der Strukturbildung, der hier untersucht wird, kann zum Beispiel in der Robotik eingesetzt werden, wenn selbst-zusammenbauende oder selbstkonfigurierbare Systeme entwickelt werden [BDT99], oder auch bei Konstruktionen in schwer erreichbaren Gebieten, bei denen ein robustes und flexibel reagierendes System notwendig ist.

Insgesamt kann gesagt werden, dass bisher vergleichsweise wenig Applikationen entwickelt wurden, die mit Hilfe von Schwarm-Intelligenz arbeiten [BDT99], dieses Feld aber vielversprechende Ansätze liefert.

Abbildung 1.1.: Beispiel für einen Termitenbau. *(c) Masson*

2. Grundlagen

In diesem Kapitel werden einige grundlegende Begriffe erläutert, besonders *Stigmergy* und *Selbstorganisation* werden definiert. Im Abschnitt „Abgrenzung von Nachbargebieten“ erfolgt eine Einordnung der Schwarm-Intelligenz in den Bereich der Wissensbasierten Systeme.

2.1. Stigmergy

Eine große Anzahl von Lebewesen konstruiert komplexe Bauwerke, die viele unterschiedliche Funktionen erfüllen, wie zum Beispiel Brutpflege, Verteidigung gegen Angreifer, thermische Regulierung und andere. Bei in Gruppen (Schwärmen) lebenden Spezies benötigt die Konstruktion solcher Architekturen irgendeine Art von Koordination [BGS+00].
Diese Koordination kann einmal durch einen Supervisor, also einen allwissenden, übergeordneten Teil der Gruppe geschehen, durch direkte Kommunikation (Absprache) zwischen einzelnen gleichberechtigten Mitgliedern einer Gruppe oder aber durch indirekte Kommunikation, die über die Umwelt bzw. mit der Umwelt als Medium durchgeführt wird. Dieser letzte Punkt wird im Folgenden näher untersucht.

Koordination durch Stigmergy[1] wurde 1959 von Grassé [Gra59] untersucht. Er beschreibt die Möglichkeiten von großen Gruppen sozialer Lebewesen, sehr komplexe Gebilde zu errichten (zum Beispiel Termitennester) und dabei nur mit Hilfe von sehr einfachen Regeln zu agieren. In diesen Gruppen existiert kein koordinierendes Individuum mit Kenntnissen über das gesamte Bauwerk, es existiert lediglich bei jedem einzelnen Lebewesen Kenntnis über einen sehr kleinen Teil der direkten momentanen Umgebung. Grassé entdeckte dabei das folgende einfache Schema, mit dem zum Beispiel eine spezielle Termitenart *(Bellicositermes natalensis)* den Bau ihrer komplexen Nester steuert [BGS+00]:

1. Eine stimulierende Konfiguration in der direkten Umwelt einer einzelnen Termite triggert eine Reaktion.

2. Diese Reaktion modifiziert oder ergänzt die momentane Konfiguration in eine andere,

3. welche dann wieder neue Reaktionen derselben oder einer beliebigen anderen Termite anstößt.

[1] Handlungsstimulierung [*stigma* - Antrieb, *ergon* - Arbeit]

Die Konfiguration kann dabei sehr unterschiedlich aussehen. Sie ist aber immer eine Summe von Sinneseindrücken, seien es zum Beispiel optische Reize von platzierten Teilkonstruktionen eines Nestes, Pheromone, die andere Lebewesen der Gruppe abgegeben haben oder eine Kombination von beiden. Diese Art der Informationsweitergabe über die Umwelt wird als *indirekte Kommunikation* bezeichnet, die Umwelt selbst übernimmt die Rolle eines Kommunikationsmediums, welches die einzelnen Individuen leitet [BDT99, HM]. Stigmergy bietet sich besonders in den Gebieten an, in denen keine direkte Kommunikation möglich oder diese zu teuer ist [Mas].
Durch dieses Vorgehen kann aus sehr simplen Handlungen bzw. Fähigkeiten[2] der einzelnen Individuen eine Komplexität entstehen, die um ein vielfaches größer ist als die einfache Summe der einzelnen Fähigkeiten [LMB95].

2.1.1. Quantitative Stigmergy

Quantitative Stigmergy tritt an den Stellen auf, bei denen eine Erhöhung eines Reizes eine Erhöhung der Reaktionswahrscheinlichkeit verursacht. Die Variable, die hierbei betrachtet wird, besitzt einen reellen Zahlenwert und ist auf ihrem gesamten Definitionsbereich stetig [MGH01]. Bei Säulenbauten von Termiten strömen beispielsweise abgelegte Elemente einen Pheromongeruch aus, der sich proportional zu der Anzahl der abgelegten Elemente verstärkt, eine größere Gesamtstruktur strahlt einen penetranteren Geruch aus als eine kleinere, und durch die Stärke einer Geruchsquelle werden andere Termiten motiviert, an dieser Stelle weiterzubauen. Mit dieser Art von Stigmergy kann also zum Beispiel ein Bauplatz fokussiert oder können Nahrungsmittel gruppiert werden.
Ein weiteres Beispiel für quantitative Stigmergy ist ein bereits erfolgreich in der Praxis angewandter Routing-Algorithmus für (Telekommunikations-) Netzwerke, der durch die Wegsuche von Ameisen motiviert wurde [BM01]. In diesem Fall besteht das Ziel in der Natur darin, die dem Bau nächstgelegene Futterquelle zu finden. Hierzu bewegen sich mehrere Ameisen in zufällige Richtungen und hinterlassen eine Pheromonspur. Wenn sie eine Futterquelle ausfindig gemacht haben, kehren sie zum Bau zurück und verstärken damit die Pheromonspur des Hinweges. Weitere Ameisen handeln nach dem einfachen Prinzip, der deutlichsten Pheromonspur mit größerer Wahrscheinlichkeit zu folgen. Da die Pheromone mit der Zeit schwächer werden, besitzt die Spur die höchste Konzentration, die am nächsten zum Bau liegt, da sich hierauf bisher mehr Individuen bewegt haben. Aus demselben Grund ist eine bestimmte Anzahl von Individuen notwendig, die alle dasselbe Ziel verfolgen. Wenn zu wenige Individuen handeln und Pheromone verteilen, verdampfen diese, ehe andere Individuen die Spur aufnehmen und verstärken können; es kann also zu keiner Organisation kommen.

[2] Hierbei wird die Handlung eines Individuums von einem gewissen Abstraktionslevel aus betrachtet, die motorischen und sensorischen Fähigkeiten, die zum Beispiel ein Insekt benötigt, um sich fortzubewegen, können durchaus als komplex bezeichnet werden

2.1.2. Qualitative Stigmergy

Im Gegensatz zur quantitativen gibt es bei der qualitativen Stigmergy keine Unterscheidung nach Stärke, sondern nach Art der Reizung. Durch Unterschiede in der Art der Stimulierung können unterschiedliche Reaktionen der einzelnen Individuen gesteuert werden. Das bedeutet, dass beispielsweise ein *type-1* Stimulus eine Aktion A anstößt. Diese Aktion A verwandelt den *type-1* Stimulus in einen *type-2* Stimulus, der wiederum eine Aktion B desselben oder eines beliebigen anderen Individuums anstoßen kann. Wieder fungiert die Umwelt als ein Kommunikationsmedium. Ein Beispiel für diese qualitative Stigmergy ist der Nestbau bei den Papierwespen *Polistes dominulus* [BDT99].
In der Natur werden häufig Mischformen von quantitativer und qualitativer Stigmergy vorgefunden, so existieren zum Beispiel Stimuli, die unterschiedliche Intensitäten innerhalb verschiedener Typen besitzen oder ein Schwellenwert, der erst beim Überschreiten zu qualitativen Aktivierungen weiterer Regeln führt.
In [HM] wird noch eine dritte Art von Stigmergy beschrieben, die dann auftritt, wenn eine Aktion weder die Art weiterer Aktionen verändert noch die Aktivierung einer solchen auslöst; es wird lediglich die Umwelt modifiziert bzw. angepasst.

2.2. Grundbegriffe

Dieser Abschnitt beschreibt einige Begriffe, die im Verlauf dieser Arbeit häufig benutzt werden, die aber in der Literatur nicht eindeutig definiert sind. In Abbildung 2.1 sind diese Begriffe zusätzlich noch graphisch dargestellt.

2.2.1. Agent

Agenten sind in dieser Arbeit die Aktoren, die die eigentliche Konstruktionsarbeit vollbringen, indem sie Blöcke anhand vorgegebener Regeln in der Umwelt platzieren. Dabei kann der Agentenbegriff folgendermaßen definiert werden: „Ein Agent ist ein System, das seine Umgebung durch Sensoren wahrnimmt und mittels Effektoren in dieser Umgebung handelt“ [LS00].
Bei Lettmann und Schulz werden folgende Typen von Agenten beschrieben:

Table-lookup Agent Das gesamte Wissen ist fest innerhalb des Agenten selbst (zum Beispiel in Form einer Tabelle) gespeichert. Bei einer gegebenen Eingabe wird der passende Datensatz und damit die Handlung des Agenten durch direkten Vergleich mit diesem Wissen herausgesucht.

Simple Reflex Agent Dieser Agententyp handelt aufgrund einfacher „wenn-dann“-Regeln, er hat gegenüber Table-lookup Agenten den Vorteil, dass mehrere Fälle zusammengefasst werden können und damit nicht alle Fälle einzeln im Speicher des Agenten abgelegt sein müssen.

Reflex Agent Eine Erweiterung des Simple Reflex Agenten, bei dem zusätzlich eine interne Repräsentation der Umwelt und das Wissen um die Konsequenzen eigener Handlungen existiert. Dadurch wird das interne Abbild der Umwelt sowohl durch Sensoren als auch durch die eigenen, ausgeführten Handlungen aktualisiert.

Goal-based Agent Ziel-basierte Agenten verfolgen ein Ziel, welches durch ihre Handlungen erreicht werden soll. Dazu berechnen sie die Resultate aller möglichen Handlungen, bevor diese ausgeführt werden und führen dann die Handlung aus, die sie dem Ziel am nächsten bringt.

Utility-based Agent Diese Agenten haben die Möglichkeit, einen Plan aufzustellen und entsprechend dieses Planes Teilziele zu erfüllen, um dem Gesamtziel näherzukommen. Die Auswahl der jeweiligen Teilziele geschieht dabei aufgrund einer *utility_function*.

Wir betrachten hier nur die einfachste Form von Agenten, den Table-lookup Typ, um zu zeigen, dass auch mit deren simplen Fähigkeiten Ergebnisse erzielt werden können. Eine Erweiterung auf den nächstkomplexen Simple-Reflex Typen ist denkbar, wenn nicht jede Regel einzeln im Agenten abgelegt wird, sondern auch Wildcards zugelassen werden, die die Umwelt nur teilweise einbeziehen.

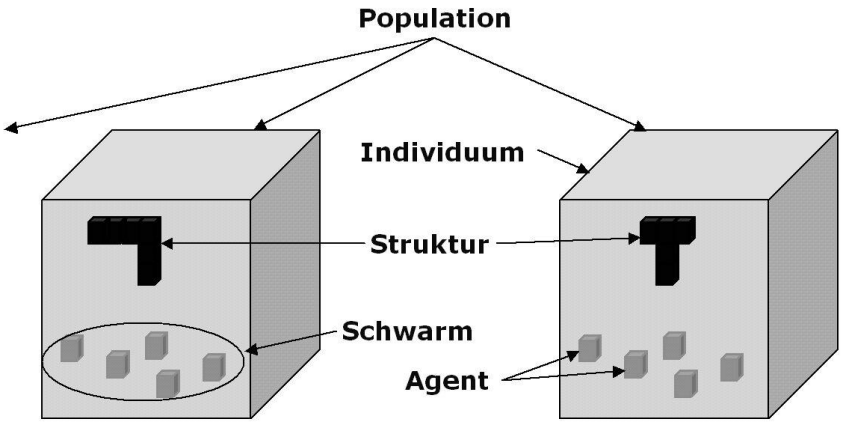

Abbildung 2.1.: Schematische Darstellung der grundlegenden Begriffe, die in dieser Arbeit benutzt werden.

2.2.2. Schwarm

Als Schwarm wird in dieser Diplomarbeit eine Menge von einzelnen, sehr simplen Agenten bezeichnet, welche exakt dasselbe Wissen und dieselben Fähigkeiten besitzen und aufgrund dieser Daten gemeinsam an der Lösung eines Problemes arbeiten. Innerhalb eines Schwarms ist keine direkte Kommunikation zwischen den einzelnen Agenten möglich, hierzu muss die Umwelt genutzt werden.

2.2.3. Individuum

Der Individuum-Begriff wird im Umfeld des genetischen Algorithmus genutzt. Dabei wird ein gesamter Schwarm bzw. die Menge von (identischen) Regeln, die jeder Agent innerhalb eines Schwarmes besitzt, als Individuum bezeichnet. In diesem Bereich symbolisiert die gesamte Regelmenge (in der Biologie auch als *Genotyp*[3] bezeichnet, die durch ein Individuum erzeugten Strukturen bilden somit den Phänotypen[4] des Individuums) die genetischen Informationen eines Individuums, ein einzelnes Gen ist eine bestimmte Regel aus der Regelmenge.

2.2.4. Population

Ein weiterer Begriff aus dem Bereich der genetischen Algorithmen: Als gesamte Population wird bei Simulationen mit *VisualSwarm* die Menge von Individuen bezeichnet, die innerhalb einer Generation durch Simulation Strukturen erzeugen. Aus einer berechneten Population wird dann durch genetische Operatoren eine neue Population von Individuen erzeugt. Ziel ist es, durch gezielten Einsatz und passende Parametrisierung der genetischen Operatoren die (durchschnittliche) Fitness der Strukturen zu erhöhen, die innerhalb einer Population durch Simulationen der Regelmengen erzeugt werden.

2.3. Selbstorganisierende Systeme

Die Theorie der Selbstorganisation wurde ursprünglich im physikalisch-chemischen Umfeld entwickelt, um das Auftreten makroskopischer Muster aus Prozessen und Interaktionen im mikroskopischen Level zu beschreiben. Diese Theorie kann ausgeweitet werden auf ethologische Systeme und auf (teilweise) soziale Insekten, um zu zeigen, dass komplexes Gruppenverhalten aus Interaktionen zwischen Individuen mit einfachen Fähigkeiten entstehen kann [BTD$^+$].
Gruppenaktivitäten können aber nicht nur durch Selbstorganisation entstehen. Gerade bei Insekten gibt es viele Beispiele für eine aktive Koordination der einzelnen Individuen; sei es bei Bienen durch Steuerung und Kontrolle durch die Königin, die ihre Informationen durch Stimulierung an die einzelnen Arbeiter(gruppen) weitergibt. Oder es wird ausgenutzt, dass das zu erzeugende Muster bereits in anderer Form existiert, beispielsweise Temperatur- oder Feuchtigkeitsgradienten, wie es einige Ameisenarten nutzen, um ihre Nester zu bauen und die Larven und Eier zu verteilen. Diese Faktoren können mit der Selbstorganisation kombiniert werden, es werden dabei nicht notwendigerweise ausschließlich identische Individuen benötigt, im Gegenteil, gerade unterschiedliche Klassen von Individuen können sehr effektiv Aufgaben lösen. Es können dabei unterschiedliche Schwellenwerte benutzt werden,

[3] Die Gesamtheit der Erbfaktoren
[4] Erscheinungsbild

um kollektives Verhalten zu organisieren. Selbstorganisation ist nicht notwendigerweise adaptiv oder kooperativ, dieses wird besonders klar, wenn man wieder zurückblickt auf physikalische Zusammenhänge, denen dieses Verhalten abgesprochen werden kann [BTD+].
Von mehreren Wissenschaftlern wird vermutet, dass Selbstorganisation deshalb von der Evolution favorisiert wurde, weil sie die Möglichkeit bietet, effiziente Zusammenarbeit zu ermöglichen, ohne auf komplexes Verhalten einzelner Individuen angewiesen zu sein:

> *„Selbstorganisation widerspricht nicht der Evolutionstheorie, sondern vervollständigt diese. Wenn man die Selbstorganisation als eine große Menge von Organisationsmechanismen erkennt und versteht, wie Selbstorganisation an vielen Beispielen von Gruppenverhalten beteiligt ist, erlaubt dieses ein besseres Verständnis der Evolution selbst“* [BTD+].

Es gibt einige Bedingungen und Voraussetzungen, die notwendigerweise von einem System erfüllt werden müssen, damit es als selbstorganisierend bezeichnet werden kann. Decker [Dec00] nennt unter anderem folgende Indikatoren für ein selbstorganisierendes System:

Thermodynamisch offen: Das System muss einen Energiefluss von außen in das System besitzen, damit es nicht lediglich aufgrund des Fehlens einer Stimulanz an einer Stelle stagniert.

Dynamisches Verhalten: Es muss eine dauernde Änderung des Systemzustandes beobachtbar sein, die sich manifestiert in einer Umwandlung von Energie, die zugeführt wird, in Entropy, die wieder nach außen an die Umwelt zurückgegeben wird.

Lokale Interaktionen: Die einzelnen Teile im System müssen miteinander interagieren.

Vorhandensein vieler einzelner Teile: Da die Mächtigkeit des Systems aus der Interaktion der einzelnen Teile entsteht, müssen im System viele solcher lokalen Teile vorhanden sein.

Neue Erscheinungen (Emergenz): In selbstorganisierenden Systemen muss aus der Summe der Verhaltensweisen der einzelnen Teile ein neues Gesamtverhalten entstehen, welches komplexer ist als die einfache Summe der einzelnen Verhaltensweisen.

Nicht-lineare Dynamik: Es müssen Schleifen mit negativen und positiven Reaktionen im Gesamtsystem möglich sein, einmal zwischen einzelnen Teilen auf der selben Ebene, aber auch zwischen hierarchisch unterschiedlich angeordneten Teilen.

Alle diese Punkte sind schwer zu formalisieren, teilweise redundant und es ist nicht beweisbar, ob in einem System mit diesen Eigenschaften Selbstorganisation auftritt, sie liefern aber einen wichtigen Ansatzpunkt zur Charakterisierung von Systemen. Wenn ein System als selbstorganisierend bezeichnet wird, bedeutet das damit, dass es nicht durch top-down Regeln gesteuert wird - obwohl einige global gültige Constraints für alle einzelnen Komponenten gelten können - sondern stattdessen die lokalen Veränderungen der Individuen die Muster erzeugen, die auf dem höheren Level geordnet und dynamisch erscheinen; daher kann ein solches System nicht durch eine Dekomposition der Resultate verstanden oder interpretiert werden [Dec00].
Langton hat 1990 [Lan90] untersucht, unter welchen Bedingungen selbstorganisierende Systeme größtmögliche Komplexität zeigen. Da man den Komplexitätsbegriff bisher noch nicht eindeutig definieren konnte[5], betrachtete Langton die Berechnungs- und Speichermöglichkeiten eines zellulären Automaten und definierte Komplexität als die Anzahl der Übergänge, die ein Automat durchläuft, bis ein stabiler Zustand erreicht wird. Ein Resultat aus dieser Arbeit war die Feststellung, dass die größte Komplexität in der Nähe chaotischer Zustände zu beobachten ist *(„edge of chaos“)*. Die Grafik in Abbildung 2.2 nach Langton [Lan90] versucht, dieses schematisch zu

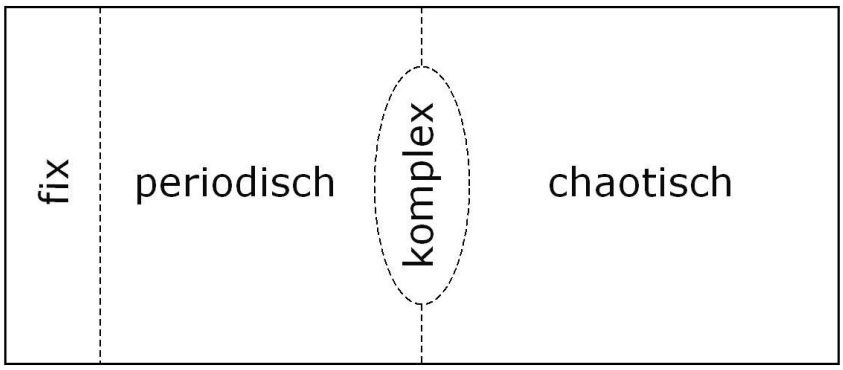

Abbildung 2.2.: Die Grafik beschreibt, wo nach Langtons Versuchen komplexes Verhalten bei der Simulation von zellulären Automaten auftrat.

verdeutlichen, eine exakte Abgrenzung der einzelnen Teilbereiche ist hierbei nicht möglich. Bei Jacob [Jacb] sind noch weitere Möglichkeiten beschrieben, die Komplexität eines speziellen Verhaltens zu bestimmen.

2.4. Strukturbildung durch Stigmergy - Selbstorganisation

Stigmergy ist ein wesentlicher Mechanismus, durch den sich die Umwelt selbstständig durch Aktivitäten von Agenten in ihr strukturieren kann. Sowohl der Zustand der Umwelt als auch die momentane Verteilung der Agenten darin entscheidet, wie die

[5] Man kann lediglich intuitiv sagen, dass Komplexität irgendwo zwischen Ordnung und Unordnung/Chaos angesiedelt werden muss

Umwelt sich entwickeln wird und wie die Agenten weiter verteilt werden. Jede so entstehende Struktur entwickelt sich in einem Prozess von dynamischer Selbstorganisierung (self-organization) im globalen System aus Interaktionen der Low-Level Komponenten. Die Regeln, die diese Interaktionen spezifizieren, werden nur aufgrund von lokalen Informationen ausgeführt, globale Informationen sind den einzelnen Agenten nicht zugänglich [HM]. Zusammengefasst werden kann der Aspekt der Stigmergy durch folgende Beschreibung:

> *„The worker does not direct his work but is guided by it“* [HM].

In dem selben Papier und bei Bonabeau [BDT99] werden vier grundsätzliche Bestandteile und drei charakteristische Ausprägungen bzw. Eigenschaften von Selbstorganisation durch Stigmergy angeführt. Die Bestandteile sind:

Positive Rückkopplung (positive feedback) Beispiele hierfür sind Rekrutierung und Reinforcement. Bei Ameisen wird ein erfolgreicher Weg (zum Beispiel zu einer Nahrungsquelle) durch positive Rückkopplung automatisch verstärkt und dadurch werden immer mehr einzelne Individuen motiviert, diese Quelle aufzusuchen.

Negative Rückkopplung (negative feedback) Sie hilft, das gesamte Muster zu stabilisieren, da sie Möglichkeiten bietet, das positive Feedback auszubalancieren, indem sie hiermit in Wettbewerb tritt.

Erweiterung der Schwankungsbreiten (amplification of fluctuations) Hierzu zählen zufällige Bewegungen, fehlerhafte Sensordaten usw., mit denen neue Lösungen gefunden werden können.

Vorkommen unterschiedlicher Interaktionen (presence of multiple interactions) Durch viele unterschiedliche Interaktionen können die stochastischen Eigenschaften ausgenutzt werden und somit wird das Entstehen von großen, stabilen Strukturen unterstützt.

Als Ausprägungen, die die Selbstorganisation beschreiben, werden genannt:

Erzeugung von räumlich-zeitl. Strukturen in einem urspr. homogenen Medium Hierzu gehören einmal die Erzeugung von Strukturen oder Pheromonpfaden, aber auch die Bildung von sozialen Strukturen unter den Individuen.

Erreichbarkeit unterschiedlicher stabiler Zustände (multistability) Das System konvergiert zu einem von vielen stabilen Zuständen, abhängig von den initialen Startparametern der Simulation.

Existenz durch Parameter steuerbarer Verzweigungen Bei (kleinen) Veränderungen von Parametern kann sich das Verhalten der Individuen grundlegend ändern.

Selbstorganisierung durch Stigmergy unterscheidet sich dabei von der reinen physikalischen Selbstorganisierung (zum Beispiel Wolkenbildung, Sanddünen-Wanderung oder Schneeflocken [MGH01]), da sie durch mobile Agenten entsteht, die aufgrund der beobachteten (lokalen) Umwelt mit Hilfe ihrer physikalischen und berechnungstechnischen Bestandteile handeln können und nur durch ihre eigenen Sensoren bzw. ihre Mobilitätsgrenzen beschränkt sind. Da Strukturen sowohl durch die Umwelt selbst als auch durch die Verteilung der Agenten erzeugt werden können, gibt es unendlich mehr Möglichkeiten als bei der Strukturerzeugung durch die reinen physikalischen Eigenschaften der Umwelt; aber nur die Instanzen von Stigmergy, die Selbstorganisiertheit unterstützen, können bemerkbare oder nützliche Effekte produzieren [HM].
Grassé hat in seinem originalen Papier [Gra59] darauf hingewiesen, dass es zwei Arten gibt, das Verhalten von Insekten zu strukturieren. Einmal das bei einzeln lebenden Spezies beobachtbare Verhalten, dass die initiale Ausführung einer Handlung intern in einem Zustand gespeichert wird und dann, häufig zusammen mit externen Einflüssen, eine nächste Handlung auslöst und so weiter. Die zweite Art kommt ohne einen internen Zustand aus und tritt sowohl bei einzeln als auch sozial lebenden Spezies auf. Bei dieser reicht der externe Zustand aus, Handlungen zu provozieren. Eine externe Stimulierung eines Zustandes setzt dabei häufig voraus, dass die vorhergehende Aktion erfolgreich abgeschlossen wurde. So kann auch hier eine Sequenz von Aktionen geordnet durchgeführt werden, es besteht aber kein Zwang, die aktuelle Position im Bauplan intern zu speichern und diese Position an andere am Bau beteiligte Individuen weiterzugeben, der aktuelle Zustand wird alleine durch die Umwelt vor- und weitergegeben; die Umwelt bietet somit eine Speicherfunktionalität [Fer01].
Sehr gute Einblicke in den Themenkomplex „Stigmergy“ bieten Simulationen, die versuchen, Stigmergy in der Natur nachzubilden bzw. Strukturen nachzubauen, bei denen vermutet wird, dass sie durch Stigmergy-Effekte entstanden sein können. So wurde nachgewiesen, dass Vorgänge in Ameisenkolonien - Gruppierung von zerstreut liegenden Objekten und Sortierung von Objekten (hier Larven) - allein durch Stigmergy-Prozesse nachgebildet werden können. Dieses kann zwar nicht nachweisen, dass Ameisen wirklich auf eine solche Art arbeiten, aber es bietet eine realistische Erklärungsmöglichkeit für dieses komplexe Verhalten [HM]. Holland und Melhuis merken hierzu zusätzlich an, dass Simulationen eher der Realität entsprechen, wenn sie unter Einbeziehung der realen Physik durchgeführt werden, etwa durch simple Roboter. Eine Nachbildung einer realistischen Physik in einer reinen Softwaresimulation ist dagegen deutlich komplizierter durchzuführen.

2.5. Schwarm-Intelligenz

Der Bereich der Schwarm-Intelligenz beschäftigt sich mit der algorithmischen Nachbildung von natürlichen, gruppendynamischen Prozessen, wie sie beispielsweise bei sozialen Insektenkolonien vorkommen. Der Begriff der *Schwarm-Intelligenz* wurde

in den 80er Jahren eingeführt, Bonabeau hat ihn in einer allgemein akzeptierten Definition mit folgenden Charakteristika beschrieben [BDT99]:

1. Existenz vieler, autonomer Agenten

2. Mehr-dimensionale Umwelt

3. Selbstorganisation der einzelnen Agenten durch Interaktionen mit anderen, benachbarten Agenten

4. Inspiration des Gesamtsystems durch real existierende, soziale Insektenkolonien oder andere Tiergemeinschaften.

In dieser Diplomarbeit wird auf einen Spezialbereich der Schwarm-Intelligenz eingegangen, die sogenannten Partikelschwärme. Diese sind ein Teilbereich der Schwarm-Intelligenz, da die einzelnen Partikel, aus denen der gesamte Schwarm besteht, nur sehr beschränkte Fähigkeiten besitzen und daher kaum als wirkliche Agenten bezeichnet werden können. Partikelschwärme versuchen nicht direkt, eine Lösung zu finden, da sie selbst kein Wissen über das Problem oder dessen Lösung besitzen, vielmehr finden sie *automatisch* eine Lösung des Gesamtsystems, indem sie reaktiv vorgehen und Reflex-gesteuert den einfachen Regeln eines Algorithmus folgen.

2.5.1. Partikelschwärme

Partikelschwärme sind Mengen von einzelnen Agenten, die Lösungen durch soziale Interaktionen in der gesamten Gruppe finden. Sie handeln dabei sowohl aufgrund ihrer eigenen Erfahrungen als auch derer benachbarter Individuen in der Gruppe. Die Anzahl der benachbarten Individuen, die berücksichtigt werden, und der Prozentsatz, der die eigene Erfahrung im Vergleich zur Gruppenerfahrung ausmacht, werden durch Parameter festgelegt [AC01].
Kennedy definiert 4 verschiedene Modelle für Partikelschwärme [Ken97]:

1. Vollständiges Modell: Es sind sowohl Erfahrungen der Gruppe als auch eigene Erfahrungen an der Entscheidungsfindung beteiligt

2. Soziales Modell: Es sind nur Gruppenerfahrungen und der eigene Zustand an der Entscheidungsfindung beteiligt, der eigene Zustand wird aber nicht angepasst

3. Selbstloses Modell: Es sind ausschließlich Gruppenerfahrungen an der Entscheidungsfindung beteiligt

4. Kognitives Modell: Es sind nur eigene Erfahrungen an der Entscheidungsfindung beteiligt

Partikelschwärme bzw. Algorithmen, die mit Hilfe von Partikelschwärmen Probleme zu lösen versuchen, unterscheiden sich deutlich sowohl von klassischen sequentiellen Algorithmen als auch von der klassischen KI [BDT99]. Der grösste Unterschied zu diesen Ansätzen besteht darin, dass die einzelnen Agenten (Partikel) keine explizite interne Repräsentation der gesamten Umwelt besitzen. Sie erfüllen Aufgaben, ohne zu planen, sie handeln vielmehr, indem sie auf die Umwelt reagieren. Außerdem existiert bei ihren Handlungen ein gewisser Grad Zufälligkeit, entweder bedingt nur durch eine zufällige Anfangsposition in der Umwelt und dann vollständiges deterministisches Verhalten oder zusätzlich durch randomisierte Bewegungen in der Umwelt.
In dieser Arbeit werden nur homogene Partikelschwärme betrachtet, das sind Mengen von Agenten, die jeweils vollkommen identische Regeln besitzen. Vorstellbar sind auch Szenarien, in denen die einzelnen Individuen unterschiedliche Aufgaben erledigen müssen, indem sie auf unterschiedliche Eingaben reagieren oder auf identische Eingaben unterschiedliche Reaktionen zeigen; nach Bonabeau [BDT99] ist die Untersuchung solcher heterogener Schwärme deutlich schwieriger im Vergleich zur Untersuchung homogener Schwärme.
Partikelschwärme können teilweise mit zellulären Automaten verglichen werden (siehe auch Abschnitt 2.6.3), sie ändern ihren Zustand, indem sie auf festgesetzte Art und Weise auf die Umwelt reagieren und diese damit verändern. Sie sind dabei selbst in keiner Weise lern- oder planfähig, ihre einzige Handlung besteht im Suchen einer geeigneten, intern abgelegten Aktion, die in der aktuellen Situation angewandt werden kann, und der Ausführung derselben. Bei einigen sozialen Spezies konnte gezeigt werden, dass ihr Verhalten durch sehr einfache *if-then/yes-no* Regeln bestimmt werden kann, in mehreren Simulationen entwickelten solch eingeschränkte Agenten ähnlich komplexes Verhalten, wie es in der Natur vorkommt [HM]. Bonabeau erweitert die Definition von *Schwarm-Intelligenz* dahingehend, dass sie durch soziale Spezies motiviert werden muss und dadurch reales, in der Biologie existierendes Verhalten nutzt, um Probleme zu lösen [BDT99].
Ein Problem der Partikelschwärme ist, dass aufgrund der fehlenden globalen Sicht der einzelnen Agenten nicht entschieden werden kann, ob die aktuelle Situation ein globales oder nur ein lokales Optimum in der Gesamtlandschaft aller möglichen Lösungen darstellt.
In dem oben beschriebenen Stigmergy-Szenario kann die Selbstorganisation und damit die Strukturbildung parallel durch beliebig viele Agenten und damit an beliebig vielen Stellen des Nestes durchgeführt werden, wenn die Individuen keinen internen Zustand benötigen, um Aktionen ausführen zu können. In einem solchen Szenario ist es nicht ausschlaggebend, welches Individuum eine Handlung durchführt, sondern allein die Tatsache, dass die Handlung durchgeführt wird, zählt; gesteuert wird alles durch die Umwelt selbst [HM].

2.5.2. Evolutionäre Partikelschwärme

Evolutionäre Partikelschwärme sind klassische Partikelschwärme, die durch evolutionsähnliche Operationen an die einzelnen Problemstellungen angepasst werden. Eine einzelne Generation von unterschiedlichen Partikelschwärmen arbeitet dabei wie eine Menge klassischer Partikelschwärme, aber am Ende einer Simulation wird das Ergebnis jedes einzelnen Schwarms bewertet und untereinander in Beziehung gesetzt. Ziel bei der evolutionären Vorgehensweise ist es, erfolgreiche Schwärme zu fördern und weniger erfolgreiche Schwärme zu bestrafen. Eine Förderung kann beispielsweise durch Bevorzugung oder Kombination mit anderen erfolgreichen Schwärmen forciert werden.
Bei Einsatz von evolutionären Partikelschwärmen wird der Lernprozess nicht bei einem einzelnen Individuum oder in einer einzelnen Population erzielt, d.h. Lernfähigkeit ist kein individuelles Merkmal eines Agenten, sondern Lernen ist eine Veränderung der Individuen über die Zeit betrachtet. Der Lernprozess ist eine Aneignung und Beibehaltung von Wissen; dieses wird als *Evolution* bezeichnet [KE01], [Fer01].
Evolutionäre Partikelschwärme bieten eine effiziente Möglichkeit, „intelligente“ Systeme zu entwickeln, ohne den klassischen Softwareprozess zu durchlaufen. Der resultierende Schwarm ist im Idealfall nach ausreichend Generationen so trainiert und damit so tolerant, dass er auf Ausfälle einzelner Individuen genauso reagieren kann wie auf ungenaue bzw. von dem trainierten Fall abweichende Vorgaben. Dabei ermöglicht das evolutionäre Vorgehen, den sehr großen Raum möglicher Strukturen effizient zu untersuchen.
In dieser Arbeit wird besonders auf diese spezielle Art von Partikelschwärmen eingegangen und die Möglichkeit untersucht, wie sich durch Evolution von Regeln Strukturen erzeugen lassen.

2.6. Abgrenzung von Nachbargebieten

In diesem Abschnitt wird der Themenkomplex der *Partikelschwärme* zu anderen ähnlichen oder verwandten Bereichen der Informatik in Beziehung gesetzt und Unterschiede und Gemeinsamkeiten werden herausgearbeitet.

2.6.1. Abgrenzung von der klassischen KI

Partikelschwärme sind, im Gegensatz zu Systemen der künstlichen Intelligenz und den meisten konventionellen, sequentiellen Computerprogrammen, keine „innerhalb ihrer logischen Argumentation eingeschlossenen Denkapparate“, sondern Mengen von autonomen Agenten, die jeweils einen Teil der Umwelt wahrnehmen, untereinander kommunizieren, sich behindern und unterstützen können. Es werden neue Themenkomplexe / Ideen untersucht, die sich mit Koordination, Kooperation und eingeschränkter Wahrnehmung beschäftigen, häufig motiviert aus der Soziologie, der Biologie und der Verhaltensforschung.

Bei Systemen der künstlichen Intelligenz wird das Einzelsystem als intelligent bezeichnet, bei Partikelschwärmen dagegen das Gesamtsystem, welches aus einzelnen, meist nicht-intelligenten Teilsystemen (Agenten) besteht. Wenn die einzelnen Teilsysteme ein eigenes, intelligentes Verhalten besitzen, ist dieses nur Mittel zum Zweck, Ziel ist es immer, das Verhalten des Gesamtsystems zu optimieren und dazu eine bestmögliche Organisations- und Kommunikationsstruktur unter den Agenten aufzubauen [Fer01].

2.6.2. Abgrenzung von Multiagentensystemen

Eine Abgrenzung von Partikelschwärmen und Multiagentensystemen (MAS) ist schwierig, da sich beide Systeme sehr ähneln. MAS werden bei Ferber [Fer01] mit folgender Definition spezifiziert:

Definition MAS Der Begriff Multiagentensystem bezeichnet ein System, das aus folgenden Elementen besteht:

1. Einer Umwelt E. E ist ein Raum, der im Allgemeinen ein Volumen hat.
2. Einer Menge von Objekten, O, die situiert[6] sind und von Agenten wahrgenommen, erzeugt, modifiziert oder gelöscht werden können.
3. Einer Menge von Agenten, A, mit $A \subseteq E$.
4. Einer Menge von Beziehungen, R, die Objekte miteinander verbinden.
5. Einer Menge von Operatoren, Op, damit Agenten Objekte empfangen, erzeugen, konsumieren, verändern und löschen können.
6. Operatoren mit der Aufgabe, die Anwendung dieser Operationen und die Reaktion der Umwelt auf die entsprechenden Veränderungsversuche darzustellen. Diese Operatoren werden als die Gesetze des Universums bezeichnet.

Diese Punkte treffen jeweils auch auf Partikelschwärme zu, wodurch diese als ein Teilbereich von MAS bezeichnet werden können. Der Punkt, der Partikelschwärme, beziehungsweise weitgefasster, der den Bereich der *Schwarm Intelligenz* von MAS abgrenzt, ist die von Bonabeau geforderte notwendige Nachbildung der Eigenschaften realer sozialer Insektenkolonien oder anderer Tiergemeinschaften [BDT99]. Brückner [Brü00] bezeichnet Schwarm-Intelligenz als einen Teilbereich der Informatik, der sich an der Überschneidung zwischen der Multiagentensystemforschung, der Künstlichen Intelligenz und dem Forschungsgebiet *Artificial Life* befindet.

[6] „situiert“ bedeutet in diesem Zusammenhang, dass zu einem beliebigen Zeitpunkt jedem Objekt aus O eine eindeutige Position in der Umwelt E zugeordnet werden kann.

2.6.3. Abgrenzung von zellulären Automaten

Zelluläre Automaten (CA)[7] besitzen - im Gegensatz zu Partikelschwärmen - nicht notwendigerweise eine Beziehung zu sozialen Spezies in der Natur, wodurch sie sich schon in diesem Punkt von Bonabeaus Definition von *Schwarm-Intelligenz* grundlegend unterscheiden. Zusätzlich besteht bei CA die Randbedingung, dass alle Zellen des Automaten vollständig identische Regeln besitzen, auf eine bestimmte Position in der Umwelt fixiert sind und ihre Zustandsübergänge synchronisiert ablaufen [Jaca]. Diese Constraints müssen bei Partikelschwärmen nicht erfüllt sein, hier können die einzelnen Agenten unterschiedliche Verhaltensweisen zeigen und zusätzlich ihre Position in der Umwelt dynamisch verändern, außerdem ist ein kontinuierliches Zeitsystem möglich. Zusätzlich zur Positionsänderung können in Partikelschwärmen auch die Nachbarn dynamisch verändert und angepasst werden, ein weiterer Punkt, der auf zelluläre Automaten nicht zutrifft [Fer01].
Aus diesen Gründen können zelluläre Automaten als sehr stark spezialisierte Partikelschwärme angesehen werden, sie bilden also maximal eine stark entartete Teilmenge derselben.
Melanie Mitchell hat in einer interessanten Arbeit gezeigt, dass auch schon sehr einfache, eindimensionale zelluläre Automaten dazu in der Lage sind, globale Zustände gemeinsam mit Hilfe aller einzelnen Zellen zu verarbeiten. Sie hat Regeln für zelluläre Automaten evolutionär erzeugen lassen, diese waren nach mehreren hundert Generationen in der Lage, einen globalen Anfangszustand mit grosser Wahrscheinlichkeit korrekt zu klassifizieren (siehe [Mit]).

2.6.4. Schwarm Intelligenz Algorithmen vs. klassische sequentielle Algorithmen

Um die Mächtigkeit von Algorithmen zu verdeutlichen, die mit Hilfe von Partikelschwärmen arbeiten, bietet sich ein Vergleich mit klassischen sequentiellen Algorithmen an. Ein herausstechender Unterschied ist einmal die Tatsache, dass die Partikelschwärme, die in dieser Arbeit betrachtet werden, nicht deterministisch arbeiten, sie sind wenigstens in ihrer initialen Phase zufällig verteilt. Meistens spielt aber auch bei den Aktionen, die durchgeführt werden, speziell bei der Bewegung der einzelnen Partikel (Agenten), der Zufall eine Rolle.
Aber auch aufgrund der Nicht-Linearität des Verhaltens eines Schwarms kann seine Handlung nur sehr schwer und aufwändig vorherbestimmt werden. In einem Schwarm beeinflusst das Verhalten eines Individuums das Verhalten der gesamten Gruppe und die Umwelt. Die Umwelt selbst beeinflusst aber wiederum das Verhalten jedes einzelnen Individuums. Dadurch ist es notwendig, um das komplexe Verhalten eines Schwarms zu bestimmen, einmal eine Analyse durchzuführen, die die einzel-

[7] Ein sehr populäres Beispiel für zelluläre Automaten ist das „Game of Life“ von Conway, das versucht, lebende Organisationen nachzubilden und dabei nur mit zwei sehr einfachen Regeln zur Übergangsbeschreibung für den Automaten arbeitet, welche schon komplexe stabile Konfigurationen erzeugen können [Fer01].

nen Teile (Partikel) untersucht und versucht, daraus das Verhalten des Ganzen zu bestimmen. Zusätzlich zu dieser Analyse ist aber noch eine Synthese notwendig, die vom Schwarm ausgehend die Handlung des Ganzen untersucht. Durch diese Rückkopplung (feedback loop) ist eine Vorhersage über die Veränderungen in der Umwelt kaum möglich [LMB95].
Die in der Natur vorkommenden Insektenkolonien mit ihren kooperativen, sozialen Verhaltensweisen können als dezentralisierte Problemlösungssysteme angesehen werden. Zwei Haupteigenschaften von diesen biologischen Gemeinschaften können auch auf Partikelschwärme übertragen werden:

Robustheit Die gesamte Gruppe kann ohne Probleme weiter effektiv arbeiten, wenn einzelne Individuen ausfallen

Flexibilität Änderungen in der Umwelt oder fehlgeschlagene Aktionen werden direkt kompensiert

In diesen beiden Punkten ist auch ein deutlicher Unterschied zu klassischen Algorithmen zu erkennen. Zwar können auch sie auf den Ausfall eines Teilbereiches reagieren und diesen Verlust ausgleichen, das geschieht aber nicht selbstständig, sondern muss vor Ablauf der Simulation eingeplant worden sein; das heißt, es ist die Aufgabe des Entwicklers der Software, alle Eventualitäten einzuplanen und dafür Vorkehrungen zu treffen. Ähnlich sieht es bei einer Veränderung der Randbedingungen aus. Auch hier muss vorher feststehen, welche Änderungen vorkommen können und diese müssen bei der Entwicklung berücksichtigt werden.
Dies bedeutet, dass die Hauptaufgabe sich bei den klassischen Algorithmen zum Programmierer verschiebt und dass, je flexibler bzw. robuster ein System sein soll, umso umfangreicher die Software vom ihm konzipiert bzw. angepasst werden muss. Bei Partikelschwärmen ist beides bereits durch die Art des Vorgehens automatisch vorhanden, das bedeutet, der Schwarm reagiert selbstständig auf Veränderungen und führt notwendige Anpassungen ohne einen manuellen Eingriff des Entwicklers durch. Diese Vorteile werden durch eine höhere Laufzeit und ein nicht deterministisches Verhalten bezahlt, allerdings haben natürliche Systeme gezeigt, dass diese Art des Vorgehens durchaus erfolgreich sein kann [BDT99].
Vorteilhaft bei Partikelschwärmen sind die geringen Anforderungen an die einzelnen Agenten. Da sie lediglich lokale Informationen benötigen, ist zur Simulation keine übergeordnete Instanz notwendig, bei der Gefahr von einem Ausfall bestehen würde, wodurch die gesamte Simulation stoppen könnte. Außerdem kann aufgrund von Messfehlern, Ungenauigkeiten oder Kommunikationsfehlern realer Sensoren ein fehlerhaftes Abbild der Umwelt entstehen, was bei sehr wenigen, lokalen Informationen, wie sie die einzelnen Agenten sammeln, deutlich seltener vorkommt als bei einer internen Repräsentation der gesamten Umwelt. Die einzelnen Agenten benötigen also bei einer Realisierung von Simulationen deutlich weniger Hardware und bieten somit auch einen Kostenvorteil. Bei einer Umsetzung in einem realen System besteht zusätzlich die Möglichkeit, dass Aktionen nicht erfolgreich ausgeführt werden, ohne dass eine direkte Fehlermeldung aus der Umwelt zum Agenten gelangt. Klassische

Algorithmen müssen auch solche Fälle berücksichtigen, Partikelschwärme führen die Handlung einfach ein zweites Mal aus, da sie keine interne Repräsentation der Umwelt besitzen, die durch die Kenntniss der eigenen Handlungen angepasst wird, und daher solche Fehler automatisch korrigieren.

3. Strukturbildung nach Bonabeau

Eric Bonabeau und andere Mitarbeiter des Santa Fe Instituts haben in einer mehrjährigen Versuchsreihe versucht, Strukturen zu erzeugen, diese mit verschiedenen Fitnessfunktionen zu bewerten und den gesamten Prozess durch Evolution der Regelmengen zu verbessern. Die Ergebnisse aus diesen Arbeiten werden im Folgenden erläutert. Der Hauptaspekt, der dabei betrachtet wurde, war die Suche nach einem simplen Agentenmodell, welches aus einfachem Verhalten komplexe Muster generieren kann, ähnlich den in der Natur auftretenden Strukturen.

3.1. Modell

Das sehr einfache Modell, welches untersucht wurde, ist ein Automat, der sich zufällig in einem diskreten dreidimensionalen Raum bewegt und nur aufgrund lokaler Stimulierungen Handlungen ausführt. Die einzige Art von Handlung ist dabei die Möglichkeit, einen einzelnen Block an die aktuelle Position zu setzen. Ein Löschen von bereits existierenden Blöcken ist nicht möglich. Da bei einer einzelnen Blockart in Experimenten keine Strukturen entstanden, wurde eine Aufteilung der Blöcke in zwei unterschiedliche Arten vorgenommen, wodurch bereits komplexe, geordnete Strukturen zu erzeugen waren.
Begonnen wurde mit Versuchsreihen, die quadratische Blöcke im Raum platzierten, aber es wurden auch Simulationen durchgeführt, bei denen die Blöcke eine hexagonale Form besaßen. Die hieraus entstandenen Formen hatten stärkere Ähnlichkeit mit den in der Natur beobachteten Strukturen [HM], allerdings konnten daraus keine grundlegend neuen Erkenntnisse gewonnen werden, weshalb ein Großteil der Simulationen mit den einfacher zu berechnenden, quadratischen Blöcken durchgeführt wurde.
Um die erzeugten Strukturen vergleichbar zu machen und ein unstrukturiertes Auffüllen des gesamten Simulationsraumes zu verhindern, wurde in jeder Simulation ein initialer Block platziert und eine Regel generiert, die direkt auf diesen einzelnen Block reagieren kann. Bei der Regelgenerierung wurde verhindert, dass Regeln entstehen, die in einer Umgebung ohne benachbarte Blöcke aktiviert werden, da diese Regel aufgrund der zufälligen initialen Positionierung der Agenten im Raum bei jedem Durchlauf zufällige Strukturen erzeugen würde, die als Gesamtstruktur nicht miteinander in Beziehung stehen und nur zu einer schnellen Füllung des gesamten Simulationsraumes beitragen.
Die qualitative Stimgmergy (siehe Abschnitt 2.1.2) ist in den Simulationen das Muster, das die direkt um den Agenten platzierten Blöcke bilden. Dieses Muster kann

eine Regel im Agenten (MicroRules, siehe Abbildung 4.2) aktivieren und damit das Ablegen eines Blockes auslösen. Der Algorithmus, nach dem die einzelnen Agenten bei der Simulation handeln, kann damit durch eine einfache LookUp-Table beschrieben werden, in der der Agent für jede Position, an der er sich aufhält, überprüfen kann, ob ein Block abgelegt werden soll oder nicht.

3.1.1. Modell-Parameter

Für die Simulation wurde von Bonabeau ein diskreter, dreidimensionaler Raum mit den Ausmaßen 16*16*16 gewählt; die Größe eines Schwarmes bestand aus 10 Agenten. Diese im Vergleich zu anderen Versuchen im Schwarm-Intelligenz Umfeld geringe Anzahl von Partikeln kann einmal durch Hardwarebeschränkungen erklärt werden; außerdem hat die Anzahl der Agenten, wie Bonabeau gezeigt hat, keinen Einfluss auf das Endergebnis, sondern lediglich auf die Laufzeit der gesamten Simulation. Diese wurde durch zwei Kriterien nach oben begrenzt, einmal durch einen Maximalwert platzierter Blöcke (hier 500), bei dessen Überschreitung von einem strukturlosen Gebilde ausgegangen werden kann und außerdem durch einen Maximalwert an Schritten, die jeder einzelne Agent durchführen kann (hier 30.000). Die Zeit selbst wurde in den Versuchen nicht kontinuierlich, sondern diskret simuliert, wobei in jedem Zeitschritt jeder Agent exakt eine Operation ausführt.
Bei Bewegungen der Agenten wurde keinerlei Art von Kollisionscheck durchgeführt, das bedeutet, dass sich zwei Agenten gleichzeitig an der selben Position aufhalten und sich durch platzierte Blöcke hindurch bewegen können. Die Umwelt, die ein Agent an der Position, an der er sich befindet, wahrnehmen kann, beschränkt sich auf die direkt angrenzenden Felder, wobei jedes Feld genau einen von drei möglichen Zuständen (*unbelegt*, *Typ-1-Block*, *Typ-2-Block*) besitzen kann [BGS+00].

3.1.2. Generierung des Agenten-Algorithmus

Der Algorithmus ist, wie oben beschrieben, eine Sammlung von einzelnen Regeln, die jeweils auf die lokal vom Agenten vorgefundene Umwelt reagieren. Für die Bildung solcher Mengen von Regeln wurden von Bonabeau folgende Möglichkeiten untersucht, eine umfassendere Erläuterung hierzu ist unter Abschnitt 3.2 zu finden:

1. **koordinierte Generierung**
   Bei diesem Vorgehen wird eine Menge von Regeln manuell generiert, ausgehend von einem Muster, das erreicht werden soll. Mit diesem Konzept ist es möglich, Strukturen, die denen natürlich entstandener Gebilde sehr nahe kommen, zu erzeugen und damit zu überprüfen, ob das Stigmergy-Konzept überhaupt realistische Resultate erzeugen kann.

2. **automatische Generierung**

   - **zufällige Generierung**
     Bei diesem Vorgehen werden die Regeln, nach denen Strukturen erzeugt werden, vollkommen zufällig generiert, indem jede einzelne Regel durch eine zufällige Kombination von umliegenden Blöcken aktiviert werden kann.
   - **evolutionäre Generierung**
     Der Punkt, der von Bonabeau sehr ausgiebig untersucht wurde, ist die automatische Generierung von Regelmengen mit Hilfe genetischer Algorithmen. Hierbei werden Regeln bzw. Teilmengen von Regeln, die strukturierte Muster erzeugen können, in einem evolutionären Prozess aus der Menge aller möglicher Regeln herausgefiltert und mit anderen Regeln bzw. Regelmengen kombiniert.

3.1.3. Generierte Strukturen

In Abbildung 3.1 sind Strukturen zu sehen, die mit Hilfe koordiniert generierter Regelmengen erzeugt wurden. Nur die Strukturen *(g)* und *(h)* sind durch eine evolutionär automatisch generierte Regelmenge entstanden. Der Unterschied zwischen diesen beiden Arten von Resultaten ist offensichtlich, aber nur schwer formal zu beschreiben.
Die Strukturen, die aus Regelmengen entstehen, sind nicht unbedingt deterministisch bestimmt. Die Strukturen *(d)* und *(e)* beispielsweise sind aus derselben Regelmenge gebildet worden, aber aufgrund der zufälligen Bewegungen der einzelnen Agenten im Simulationsraum und ihrer zufälligen Anfangspositionen können aus identischen Regelmengen vollkommen unterschiedliche Resultate entstehen.
Bonabeau erklärt die Ähnlichkeit zwischen den Strukturen *(d)* und *(e)* und die deutlichen Unterschiede zwischen *(g)* und *(h)*, die ebenfalls beide aus einer Regelmenge entstanden sind, mit dem Fehlen eines eindeutigen Weges zur endgültigen Struktur. Je mehr Möglichkeiten es gibt, eine Struktur weiterzubauen, desto unterschiedlicher fällt das Resultat aus. Die Struktur *(k)* zum Beispiel ist durch ihre Regelmenge und einzelne Regeln, die den Bau koordinieren, eindeutig bestimmt, bei jedem Simulationsdurchlauf entsteht die gleiche Struktur.

3.1.4. Analyse

Theraulaz und Bonabeau haben aus diesen Beobachtungen in zwei Arbeiten ([TB95a, TB95b]) folgende Punkte aufgezeigt:

1. Zwei Architekturen, die ähnlich strukturiert sind, scheinen von zwei ähnlichen koordinierten Algorithmen erzeugt worden zu sein. Dieses kann man sowohl über strukturierte als auch über unstrukturierte Architekturen sagen.

Der Umkehrschluss dieser Aussage ist allerdings nicht gültig, zwei ähnliche Algorithmen produzieren nicht unbedingt ähnliche Strukturen. Beweisen lässt sich diese Aussage einfach dadurch, dass zwei identische Regelmengen R_1 und R_2 betrachtet werden. Wenn aus der Regelmenge R_1 die Regel entfernt wird, die auf den initialen Block reagiert, kann durch sie keine Architektur mehr erzeugt werden, das Resultat unterscheidet sich also deutlich von dem Resultat der Regelmenge R_2.

Über die resultierenden Architekturen von Algorithmen, die nicht aus koordinierten Regelmengen entstanden sind, können keinerlei Aussagen gemacht werden.
Aus diesen Aussagen schließen Theraulaz und Bonabeau, dass die Landschaft, die durch den Algorithmus abgesucht wird, sowohl bei den koordinierten als auch bei den automatisch generierten Regelmengen sehr schroff (rugged) ist, aber ein wenig glatter bei den koordinierten.

2. Der Unterraum der strukturierten Gebilde ist sehr kompakt. Diese Eigenschaft lässt vermuten, dass der Bereich der strukturierten und damit „interessanten“ Muster, die mit dieser Algorithmenfamilie erzeugt werden können, sehr klein ist.

Diese beiden Eigenschaften haben - vom biologischen Standpunkt aus betrachtet - wichtige Konsequenzen, da nach (1) die Morphogenese sehr robust ist und nach (2) die Art der Strukturen, die gebildet werden können, von sehr vielen Nebenbedingungen abhängt [HM].

3.2. Suchmethoden

Der gesamte Raum möglicher Architekturen, die aus dem Platzieren von zwei Blocktypen entstehen können, ist riesig und langwierig zu untersuchen, auch wenn die Regelmenge eingeschränkt wird. Die Möglichkeiten, die von Bonabeau getestet wurden, um Regelmengen zu erzeugen, werden hier kurz vorgestellt und die Ergebnisse jeweils zusammengefasst.

3.2.1. Koordinierte Suche

Mit Hilfe von manuell erzeugten Regelmengen ist nachzuweisen, dass Stigmergy eine realistische Erklärungsmöglichkeit für natürliches Verhalten sozialer Spezies bieten kann. Bonabeau hat an mehreren Strukturen, die in vergleichbarer Weise in der Natur vorkommen, gezeigt, dass diese anhand sehr einfacher Regeln in dem betrachteten Modell durch Simulationen entstehen können. Die Regelmengen wurden dabei bei einigen sehr einfachen Strukturen einzeln aufgestellt, bei komplexeren durch einen Algorithmus, der das zu erzeugende Muster als Eingabe erhält, automatisch generiert, wobei sogenannte BottleNeck-Regeln zur Koordination des Bauvorhabens eingeführt wurden.

3.2.2. Zufällige Suche

Die zufällige Generierung von Regelmengen war der erste Versuch, die Regeln automatisch aufzustellen und somit zu beliebigen strukturierten Mustern zu kommen, ohne ein einzelnes, spezielles Muster als Ziel zu haben. Hierzu wurde eine Menge von Regeln erzeugt, die jeweils auf zufällig gewählte Kombinationen von Blöcken in der Umgebung des Agenten reagierten. Die Resultate waren in keinem Simulationsdurchlauf in irgendeiner Weise interessant oder aussagekräftig, es traten lediglich Muster auf, die vollkommen zufällig waren und/oder schnell den gesamten Simulationsraum ausfüllten. Ein natürliches oder wenigstens naturähnliches Gebilde konnte nicht beobachtet werden.

3.2.3. Evolutionäre Suche

Um realistische Resultate zu erreichen, machte Bonabeau Gebrauch von genetischen Algorithmen, die mit einer heuristischen Fitnessfunktion arbeiten, um den Raum gezielt zu durchsuchen [BTC]. Diese Fitnessfunktion ist schwierig zu definieren, da interessante oder strukturierte Architekturen schwer zu formalisieren sind. Die Fitnessberechnung ist besonders bei frühen Generationen schwierig durchzuführen, da hier nur schwer Aussagen über Strukturiertheit gemacht werden können[1].
Bonabeau hat in einem ersten Versuch eine Heuristik gewählt, der die Anzahl der Regeln zugrunde lagen, die zur Erzeugung der Struktur genutzt wurden. Bei untersuchten Strukturen hat er herausgefunden, dass „interessante“ Strukturen mehr Regeln benutzen als „einfache“, raumfüllende. Der Anteil „interessanter“ Muster war hierdurch bei Einsatz dieses Algorithmus deutlich höher als bei zufällig generierten Regelmengen. Hieraus zog Bonabeau zwei Schlüsse:

1. Algorithmen, die interessante Muster erzeugen, sind sehr selten
2. Die Fitness-Funktion ist nicht ausreichend detailliert

Ausführlichere Informationen zu den einzelnen Kriterien, die Bonabeau in der Fitnessfunktion berücksichtigte, sind im Kapitel VisualSwarm zu finden. In diesem Softwarepaket wurden Bonabeaus Ideen für die Bewertung von Strukturen nachvollzogen.

Bewertung von Struktur vs. Bewertung von Funktion Die Bewertung der erzeugten Architekturen kann auf zwei Arten erfolgen, einmal im Hinblick auf ihre Funktionalität oder aber im Hinblick auf ihre Strukturiertheit. Sinnvoller scheint die Bewertung der Funktion einer Struktur zu sein. Krink und Vollrath testeten 1997 in einer Arbeit [KV97] erfolgreich die Funktionalität von Spinnennetzen, indem sie die gefangenen künstlichen Beutetiere mit den Baukosten für das Netz[2] ins

[1] Ein Beispiel für dieses Problem: Was ist strukturierter, eine einfarbige Reihe aus fünf Blöcken oder ein 2x2 Quader aus 2 unterschiedlichen Blocktypen?
[2] Unterschieden wurde bei den Baukosten zwischen zwei unterschiedlichen Nestbaumaterialien

Verhältnis gesetzt haben. Die Funktionalität zum Beispiel eines Termitennestes wird dagegen durch mehrere Faktoren bestimmt, zum Beispiel Stabilität, thermische Regulierung und Verteidigungsstärke gegen Angreifer. Diese Funktionalität ist aber zu komplex und zu wenig genau spezifizierbar und daher laut Bonabeau kaum sinnvoll und realistisch durch eine Rechenvorschrift bestimmbar.
Aus diesem Grund werden die erzeugten Strukturen bei Bonabeau nur bezüglich ihrer (subjektiven) Strukturiertheit bewertet. Vorteilhaft ist dabei zusätzlich, dass der Bereich der Morphogenese[3] und der Mustergenerierung durch Schwärme stärker betont wird.

3.3. Erweitertes Modell

Bonabeau hat zur Verbesserung der evolutionären Suche einige zusätzliche Funktionen eingebaut. Einmal wurde die Crossover-Operation angepasst, indem die Regeln (Gene) eines Individuums vor der Operation sortiert wurden. Bei dieser Sortierung sind die Regeln, die voneinander abhängig sind, „nahe“ nebeneinander abgelegt worden. Bei Crossover-Operationen wurden dadurch zusammengehörende Regeln nicht getrennt und somit Teilstrukturen vollständig an die Nachfahren weitergegeben. Durchgeführt wurde die Sortierung mit Hilfe eines Bi-Partitionierungs-Algorithmus (Fiduccia und Mattheyeses, 1982).
Ein zweiter Verbesserungsvorschlag hat direkt die Regelmenge eines Schwarmes modifiziert. Wenn ein Schwarm unterdurchschnittlich schlechte Ergebnisse produzierte und nur wenige Regeln während einer Simulation aktiviert wurden, werden diesem einige Regeln hinzugefügt, die in Kombination mit der Umwelt und den übrigen Regeln aktiviert werden. Diese neuen Regeln wurden über mehrere Generationen ohne Anwendung von genetischen Operatoren (wie beispielsweise Mutation) weitergegeben. Insgesamt hat diese Methode jedoch keine signifikanten Verbesserungen hervorgebracht [BGS+00].

[3] Formbildung

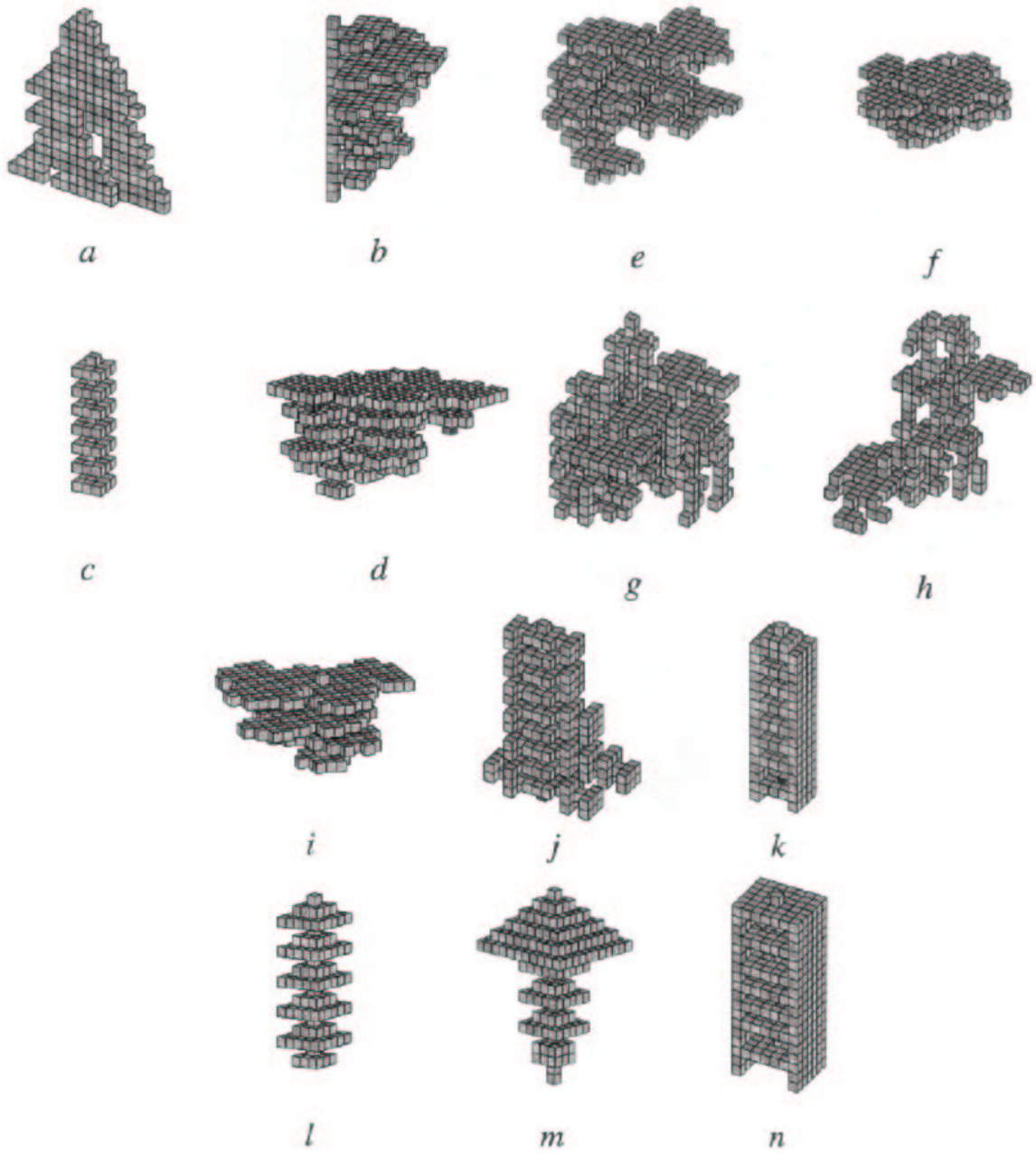

Abbildung 3.1.: Beispiele für Strukturen, die aus manuell generierten Regelmengen erzeugt wurden. Die Strukturen *(g)* und *(h)* wurden automatisch mit Hilfe genetisch generierter Regeln erzeugt.

(c) Bonabeau 2000

4. VisualSwarm

In diesem Kapitel wird das während der Diplomarbeit entwickelte Programm *VisualSwarm* vorgestellt. Es werden die einzelnen Ideen und benutzten Algorithmen beschrieben sowie Schnittstellen und Datenstrukturen erläutert, die eine Weiterentwicklung der Software ermöglichen sollen.
Eine genaue Beschreibung der einzelnen Funktionen und der dazugehörenden Parametrrisierung kann in der zum Programm gehörenden Hilfe (JavaDoc) nachgelesen werden.

4.1. Einleitung

Die Berechnung großer Populationen von Partikelschwärmen kann sehr zeitaufwändig werden, da für effektives Ausnutzen der genetischen Operatoren bei der Evolution einige hundert Generationen einer nicht zu kleinen Population simuliert werden müssen[1]. Daher wurde die Software verteilt entwickelt, um ganze Netzwerke von Rechnern für Simulationen nutzen zu können. Eine Bedingung, die an dieses verteilte System gestellt wurde, war die Ausfallsicherheit, da es bei mehrtägigen Berechnungen auf einem Cluster von Rechnern häufiger zu Ausfällen einzelner Maschinen kommen und eventuell die gesamte Simulation abbrechen kann. Daher wurde ein Server-Worker System entwickelt, welches Simulationsaufgaben erzeugt und diese an beliebig viele Worker verteilt, die durch TimeOut-Zeiten überprüft und kontrolliert werden.
Hierzu existiert ein *Server*, der alle Berechnungen anstößt und die einzelnen Ergebnisse wieder einsammelt und verwaltet. Außerdem werden durch diesen Server die genetischen Operationen durchgeführt, die neue Generationen von Individuen und damit neue Berechnungsaufgaben bilden.
Die *Worker* erhalten vom Server Simulationsparameter in Form eines XML[2]-Datenstroms. Mit Hilfe dieser Parameter wird eine Struktur (Nest) erzeugt und bewertet. Im Anschluss an diese Schritte werden die Daten dieser Struktur an den Server zurückgeschickt.
Während einer Simulation auf einem Worker-Rechner bewegen sich einzelne Agenten zufällig in einer beschränkten Welt. An jeder Stelle, an die sich ein Agent bewegt, überprüft dieser, ob die gegebenenfalls vorhandenen Blöcke eine seiner Regeln (MicroRules, siehe Abschnitt 4.2.2) stimulieren. Wenn dies der Fall ist, legt der Agent

[1] Bonabeau berechnete mehrere hundert Generationen mit jeweils 80 Strukturen und 30.000 Simulationsschritten
[2] e**X**tensible **M**arkup **L**anguage

einen Block an der Position ab, an der er sich befindet. Diese Blockplatzierungen werden in einer Datenstruktur (*ConstructionGraph*) gespeichert, welche auch Rückschlüsse auf die Baureihenfolge und damit die Regelabhängigkeiten untereinander zulässt.
Die Simulation endet, wenn jeder Agent die durch Parameter angegebene Anzahl von Schritten durchgeführt hat oder wenn die Anzahl der insgesamt platzierten Blöcke einen Maximalwert übersteigt[3].

4.2. Simulationsaufbau

4.2.1. Server-Worker Architektur

Die Software ist aufgrund der komplexen und zeitaufwändigen Berechnung verteilt konzipiert worden, um mehrere Rechner / Prozessoren für die gesamte Simulation einer großen Population benutzen zu können. Die Kommunikation zwischen dem Server und den Workern geschieht über einen XML-Datenstrom. Dieser enthält vom Server alle Simulationsparameter, die für eine Simulation benötigt werden, inklusive der Regeln, welche die Agenten zur Erzeugung der Nest-Struktur anwenden. Der XML-Datenstrom der Worker zurück an den Server beschreibt die erzeugte Struktur und die Regeln, die dazu benutzt wurden (siehe Abbildung 4.1).

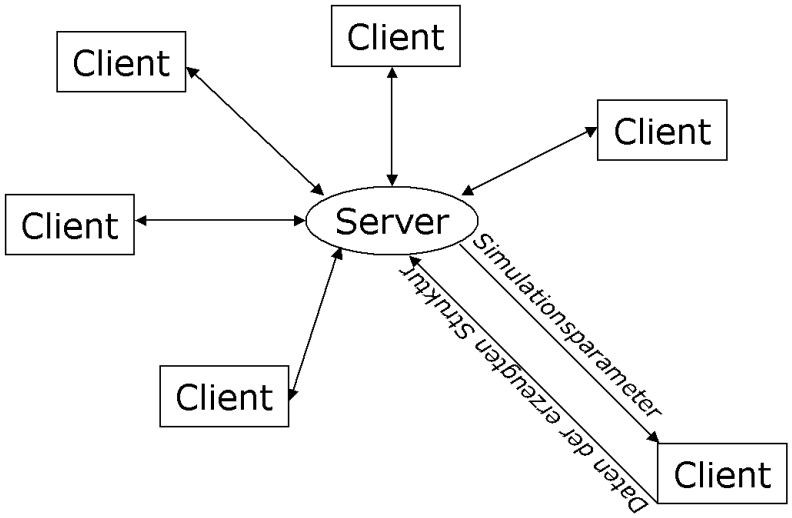

Abbildung 4.1.: Aufbau der verteilten Berechnung: Server-Worker Prinzip

Beim Starten der Simulation werden die einzelnen Regelmengen (Individuen) nacheinander auf alle Worker, die sich anmelden, verteilt. Nach einer erfolgreichen Simulation werden die Ergebnisse zurück an den Server geschickt; wenn die Berechnungsdauer eines einzelnen Workers eine TimeOut-Zeit überschreitet, wartet der Server

[3]Chaotische, strukturlose Gebilde entstehen in diesem Fall. Die Berechnung wird abgebrochen, um die gesamte Simulationszeit zu verkürzen

nicht länger auf ein Ergebnis vom Client, da dieser möglicherweise abgestürzt ist, sondern schickt die gleichen Simulationsdaten zu dem nächsten freien Worker. Außerdem bietet der Server die Möglichkeit, die Simulation an beliebiger Stelle zu unterbrechen und mit modifizierten Konfigurationsdaten wieder aufzunehmen.
Wurde auf diese Art eine gesamte Population berechnet, werden vom Server die einzelnen erzeugten Strukturen mit Hilfe eines genetischen Algorithmus bewertet und aus diesen Ergebnissen neue Individuen für die nächste Generation erzeugt.
Da bei sehr komplexen Berechnungen ein Speicherfehler in der benutzten Simulationsumgebung *SWARM* aufgetreten ist, wurde das Programm so modifiziert, dass nach jeder berechneten Population sowohl der Server als auch die einzelnen Worker mit Hilfe eines Shell-Scriptes neu gestartet werden. Der Server setzt dabei die letzte Berechnung, die gesichert wurde, einfach fort.

4.2.2. MicroRule

Eine MicroRule ist eine Regel, die aufgrund eines speziellen Musters der Blöcke in der (direkten) Nachbarschaft des Agenten aktiviert wird. Bei einer Aktivierung legt der Agent einen Block ab und erweitert damit das gesamte „Nest“. In den durchgeführten Versuchen wurde die direkte Umgebung eines Agenten betrachtet, das heißt an jeder Position wurden alle angrenzenden Positionen in der Umgebung betrachtet, sodass insgesamt 27 Felder ausgelesen worden sind (siehe Abbildung 4.2). Jedes Element einer Population besitzt eine eigene Menge von MicroRules, wodurch

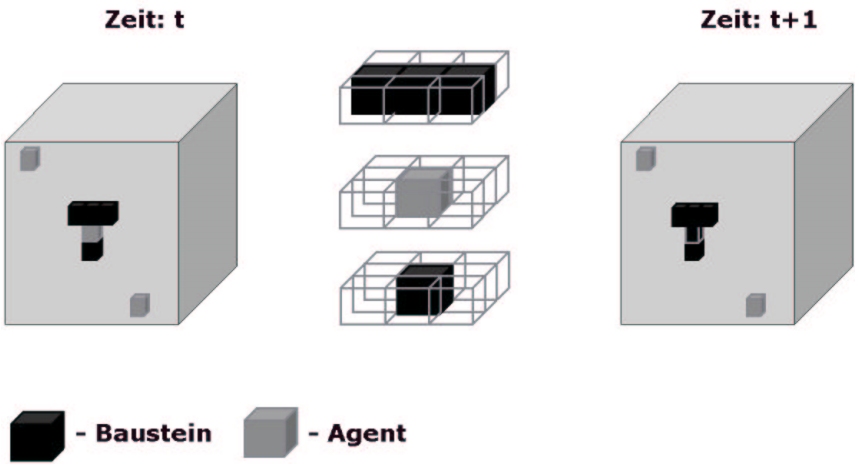

Abbildung 4.2.: Anwendung einer einzelnen MicroRule

deren Bauverhalten gesteuert wird. Innerhalb eines Elementes wendet jeder Agent die selbe Regelmenge an (homogener Partikelschwarm), daher bestimmt die Anzahl der Agenten lediglich die Baugeschwindigkeit und nicht die erzeugte Struktur. Da die Baustellen jedoch vollkommen parallel erweitert werden können und ein einzelner Agent nicht im Raum springen kann, steigt die Baugeschwindigkeit überproportional mit der Anzahl der Agenten, weil der Zeitaufwand, von Baustelle i zu Baustelle

j zu gelangen, nicht benötigt wird, wenn sich Agenten sowohl an der Stelle i als auch an der Stelle j aufhalten.
Die initiale MicroRule-Menge wird zufällig erzeugt, es wird lediglich soweit eingegriffen, dass in der Regelmenge wenigstens eine Regel vorhanden ist, die durch den initial vorhandenen Block aktiviert wird. Durch diesen manuellen Eingriff wird sichergestellt, dass bereits bei Simulationsbeginn Strukturen erzeugt werden können. Dieses Vorgehen hat jedoch keinerlei Auswirkungen auf die erzeugten Strukturen, da die initialen Regeln zufällig aus allen möglichen Regeln ausgewählt werden, die durch einen einzelnen Block aktiviert werden können. Es besteht aber zusätzlich die Möglichkeit, über eine Konfigurationsdatei jeder Population weitere manuell erzeugte Regeln hinzuzufügen, um so die Erzeugung bestimmter Anfangsstrukturen zu fokussieren.
Bei der Erzeugung der initialen MicroRules kann über die Konfigurationsdatei gesteuert werden, mit welcher Wahrscheinlichkeit an jeder Position der 27 Felder einer MicroRule ein Block gesetzt wird. Es hat sich in mehreren Versuchen mit VisualSwarm und auch bei Versuchsreihen von Bonabeau gezeigt, dass diese Wahrscheinlichkeit nicht zu hoch gewählt werden sollte (0.1-0.2), da ansonsten eine Aktivierung dieser MicroRules nur sehr selten stattfindet, weil sie an zu viele Nebenbedingungen (=Blockmuster) geknüpft ist.
Außerdem wird in dieser ersten Phase der MicroRule-Erzeugung zusätzlich festgelegt, wie groß die Anzahl der MicroRules für das jeweilige Element der Population sein darf. Hierzu können Minimal- und Maximalwerte spezifiziert werden oder aber auch für alle Partikelschwärme eine einheitliche Menge von Regeln festgelegt werden.

4.2.3. ConstructionGraph

Der ConstructionGraph dient dazu, ein einzelnes Nest in seiner Struktur zu repräsentieren. Diese Informationen werden von einem Simulations-Client zum Server zurückgeliefert und bei diesem wird aufgrund der Daten im ConstructionGraph die erzeugte Struktur bewertet.
Im Einzelnen enthalten die Knoten im Graphen ein Wertepaar (X/Y), die fortlaufende Nummer jedes abgelegten Blocks (X) und die Nummer der MicroRule, die dieses Ablegen bewirkt hat (Y). Sind beispielsweise bereits 17 Blöcke platziert worden und die MicroRule 43 wird durch einen Teil dieser platzierten Blöcke aktiviert und erzeugt einen neuen Block, erhält dieser Knoten im ConstructionGraph das Wertepaar (18/43). In Abbildung 4.3 ist ein Teil eines ConstructionGraph abgebildet, zusammen mit der Struktur, die während dieser Simulation erzeugt wurde.
Die Kanten sind gerichtet und zeigen an, welche MicroRule von welchen Blöcken und damit Knoten im ConstructionGraph aktiviert wurde. Ein einzelner Knoten kann somit maximal 26 (bei geringstmöglicher Nachbarschaftsgröße von einem Feld) eingehende Kanten haben (alle Positionen um den Agenten herum sind mit Blöcken besetzt und diese Struktur aktiviert eine MicroRule).
Es ist also möglich, die Zusammenhänge von MicroRules aus dem ConstructionGraph wiederherzustellen; allerdings gelingt es nicht, die Struktur als solche aus

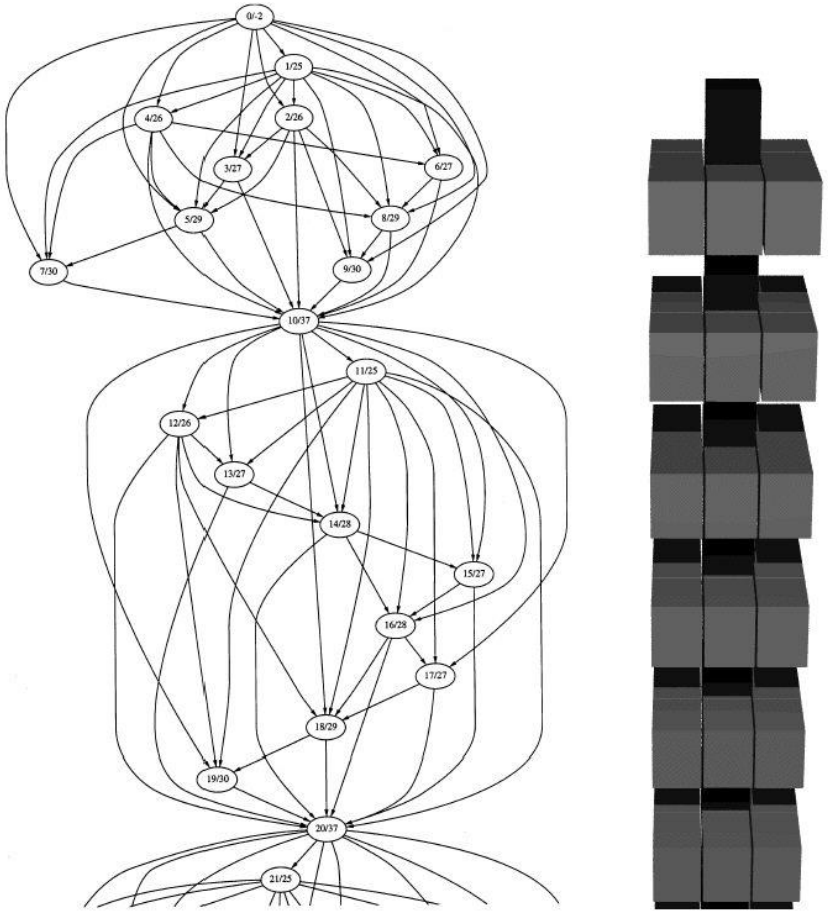

Abbildung 4.3.: Beispiel für einen ConstructionGraph *(c) Bonabeau 2000*

dem ConstructionGraph zu erzeugen, was aber auch für die weiteren Berechnungen nicht notwendig ist.

4.3. Suchmethoden

Um Regelmengen zu finden, die als strukturiert bezeichnete Muster erzeugen können, muss der Lösungsraum, der aus allen vorstellbaren Regelmengen besteht, durchsucht werden. Hierzu gibt es unterschiedliche Möglichkeiten. Einmal der Bereich der koordinierten Suche, bei der ein menschlicher Teil die Regelmengen auswählt und gegebenenfalls manuell anpasst. Auf der anderen Seite gibt es den komplexen Bereich der automatischen Suche, bei der aufgrund von Bewertungsfunktionen Lösungen gefunden werden sollen. Speziell dieser Bereich wird hier näher betrachtet.

4.3.1. Zufällige Suche

Bei dieser Suche wird zufällig eine Regelmenge generiert und das resultierende Muster durch Simulation der Regelmenge erzeugt; hierbei entstehen keine interessanten

Lösungen, da der Lösungsraum zu groß und die Anzahl *interessanter* Regelmengen zu gering ist (siehe Abschnitt 3.2.3 und [BTC]). Die Ergebnisse bei Simulationen mit VisualSwarm decken sich dabei mit den Beobachtungen, die Bonabeau gemacht hat.

4.3.2. Vollständige Suche

Eine theoretische Möglichkeit, den Lösungsraum zu durchsuchen, bietet eine vollständige Suche über alle möglichen Regeln und Regelkombinationen. Realisert werden könnte dieses beispielsweise durch eine Tiefensuche, die alle möglichen Regelmengen generiert und dann simuliert. Die bei der Simulation *beste* Struktur wäre dann das Resultat aus der *besten* Regelmenge, bewertet durch eine Fitnessfunktion. Dieses Vorgehen ist allerdings schon mit den kleinstmöglichen Simulationsparametern praktisch nicht berechenbar. Es existieren insgesamt in einem sehr einfachen Modell mit zwei verschiedenen Blockarten (also 3 Zuständen pro Feld im Simulationsraum), 50 MicroRules pro Regelmenge und der kleinstmöglichen Nachbarschaft von 26 Blöcken (die direkt an das aktuelle Feld anschließenden Positionen) bereits über 10^{650} unterschiedliche Regelmengen[4].

4.3.3. Fokussierte Suche

Um den Lösungsraum einzuengen, besteht die Möglichkeit, die Suche aufgrund von Teilergebnissen in bestimmte Richtungen zu fokussieren. Dazu gibt es viele unterschiedliche Möglichkeiten, das Grundprinzip ist jedoch immer ähnlich. Am Anfang wird zufällig gleichzeitig an verschiedenen Stellen im Lösungsraum gesucht und die einzelnen Ergebnisse werden gesammelt. Die dabei entstandenen Muster werden bewertet und nach ihrer Strukturiertheit sortiert.
Die dann folgenden Suchen werden abhängig von den erfolgreicheren früheren Lösungen durchgeführt. Dabei werden die Regelmengen, die zu guten Ergebnissen geführt haben, als Grundlage für die nachfolgenden Regelmengen in den nächsten Suchanfragen genommen und verändert. Diese Veränderung der Regelmengen können zum Beispiel durch *Simulated Annealing*-Techniken durchgeführt werden. Das bedeutet, dass zu Beginn der Suche die Anpassungen bzw. Veränderungen von bisher erfolgreicheren Regelmengen sehr groß ausfallen (es werden viele Regeln ausgetauscht), um aus lokalen Maxima wieder herauszugelangen, da diese nicht unbedingt ein *globales* Maximum darstellen. Je länger die Suche dauert, desto geringer werden die Anpassungen der Regelmenge, bis sich ein Maximum herausgebildet hat in Form einer Regelmenge, die erfolgreiche Strukturen (in Bezug auf die gewählte Bewertungsfunktion) produzieren kann.
Eine andere Möglichkeit der Umsetzung einer fokussierten Suche ist das Durchsuchen des Lösungsraumes mit Hilfe von evolutionären Algorithmen. Diese Möglichkeit ist in der Diplomarbeit detailliert mit unterschiedlichen Parametern untersucht wor-

[4] Bestimmung der Regelmengengröße: $Anzahl\ Zustaende^{(benachbarte\ Felder * Anzahl\ Regeln)}$

den und wird im Folgenden näher beschrieben.
Zusätzlich zu der Suche *irgendeiner* interessanten Struktur wurde auch der Fall betrachtet, wie eine Regelmenge aus dem Vorhandensein eines Vorgabemusters erzeugt werden kann, damit sie dieses Muster möglichst genau nachbilden kann. Hierdurch kann bestimmt werden, wie schnell und erfolgreich evolutionäre Anpassungen einer Regelmenge funktionieren.

4.3.4. Evolutionäre Suche durch genetische Algorithmen

Die Regelmengen der auf den Workern berechneten Strukturen sollen sich durch Evolution weiterentwickeln und immer strukturiertere Nester erzeugen. Dazu wird ein genetischer Algorithmus benutzt, der nach folgendem Schema arbeitet:

1. Erzeugen zufälliger MicroRule-Mengen (siehe Abschnitt 4.2.2)
2. Berechnung von Nestern durch Simulation eines Schwarmes primitiver Agenten, der jeweils eine Menge von MicroRules benutzt
3. Bewertung der erzeugten Nester mit Hilfe einer Fitnessfunktion
4. Generierung neuer MicroRule-Mengen mit Hilfe von genetischen Operatoren (Crossover, Selektion, Mutation)
5. Wiederholung ab Schritt 2, bis ein Abbruchkriterium erfüllt ist

Diese einzelnen Punkte werden im Folgenden näher erläutert.

4.3.4.1. Bewertungsfunktion Strukturfindung

Die Bewertungsfunktion beurteilt eine erzeugte Struktur aufgrund der Daten, die im ConstructionGraph enthalten sind. Sie besteht aus drei separaten und unabhängigen Teilfunktionen ($F_1 - F_3$), die in einer Linearkombination zusammengerechnet werden. Dazu kann für jede Teilfunktion ein Faktor (f_i) und eine Potenz (p_i) konfiguriert werden. Die gesamte Bewertungsfunktion hat somit die Form:

$$Fitness\ F = \prod_{i=1}^{3} (f_i * F_i^{p_i})$$

Da die Berechnung dieser Fitness, besonders der Teil F_3 (siehe unten), bei komplexen Strukturen einige Zeit in Anspruch nehmen kann, wird sie auf den einzelnen Workern durchgeführt, um den Server nicht zu sehr zu belasten.

Regelmenge (F_1) Die genutzte Regelmenge bestimmt den ersten Teil der Fitnessfunktion (F_1). Dieser Wert beschreibt, wie viele unterschiedliche MicroRules aus

den bei der Simulation vorhandenen Regeln angewandt wurden, um die resultierende Struktur zu erzeugen. Der Wert schwankt zwischen 0 und 1 und wird nach der Formel

$$F_1 = \frac{\#\ angewandte\ MicroRules}{\#\ MicroRules}$$

berechnet. Die Anzahl der genutzten MicroRules wird mit Hilfe des *ConstructionGraphs* bestimmt, da hierin die Nummer jeder MicroRule abgelegt wurde, die die Platzierung eines Blockes bewirkt hat. Mit Hilfe der frei konfigurierbaren Variablen f_1 und p_1 kann dieser Teil in der gesamten Fitnessfunktion beliebig stark gewichtet werden.

Kompaktheit (F_2) Um die erzeugten Strukturen im Hinblick auf ihre Kompaktheit beurteilen zu können, wurde der zweite Teil der Fitnessfunktion (F_2) eingeführt. Je mehr Seiten jedes einzelnen Blocks an einen anderen angrenzen, desto höher ist die gesamte Kompaktheit der Struktur. Die Werte werden ebenfalls mit Hilfe des *ConstructionGraphs* berechnet, indem die Anzahl der Kanten in dem Graphen mit der Anzahl der Knoten in Bezug gesetzt wird:

$$F_2 = \frac{\#\ der\ Kanten}{\#\ der\ Knoten}$$

Jede Kante im *ConstructionGraph* wurde erzeugt, um einen Zusammenhang zwischen dem aktuell platzierten Block mit den umgebenden Blöcken herzustellen, die diese Platzierung bewirkt haben. Da die umgebenden Blöcke auf jeden Fall an den neuen Block angrenzen, kann dieser Wert zur Bestimmung der Kompaktheit herangezogen werden.
Auch für diesen Teil der Fitnessfunktion gilt, dass er durch Modifikation der Variablen f_2 und p_2 in Bezug auf die gesamte Fitnessfunktion beliebig stark oder schwach gewichtet werden kann.

Strukturen (F_3) Um die Strukturiertheit eines komplexen Nestes bewerten zu können, benötigt man eine Aussage darüber, ob das Nest Teilmuster enthält und in welcher Größe und Häufigkeit diese Muster vorkommen. Mit Hilfe des dritten Teils der Fitnessfunktion (F_3) wird versucht, diesen Wert zu bestimmen. Dabei wird ein Muster nicht in dem erzeugten Nest gesucht, sondern im *ConstructionGraph*. Das bedeutet, dass nicht Blockkombinationen verglichen werden, sondern Reihenfolgen bei MicroRule-Ausführungen. Da identische Abfolgen von MicroRule-Ausführungen identische Blockplatzierungen (losgelöst von der genauen Positionierung im Raum) auslösen, kann so die korrekte Anzahl und Größe von Mustern gefunden werden.
Der Algorithmus zur Musterfindung kann vereinfacht folgendermaßen dargestellt werden:

```
FOR ALL Knoten des ConstructionGraph (Breitensuche)
    Muster = findStructure(null, Knoten)
```

```
FUNCTION findStructure(Muster, Startknoten)

    Suche im Graphen unter dem Startknoten einen Knoten mit
    der selben Nummer, die der Startknoten besitzt
    IF zweiter Startknoten gefunden

        Muster += Startknoten
        FOR ALL Nachfolger des ersten Startknotens
            IF Nachfolgerknoten auch NachfolgerKnoten
            des zweiten Startknotens
              Muster += findStructure(Muster, Nach-
              folgerknoten)

    return Muster
```

Abbildung 4.4 verdeutlicht den Prozess der Suche an einer Struktur mit der Größe 4. Der Algorithmus berechnet alle Muster beliebiger Größe, angefangen bei Mustern, die aus nur einem Knoten bestehen. Die Größe der später in der Fitnessfunktion berücksichtigten Muster kann durch Parameter auf eine Mindestgröße festgelegt werden, beispielsweise können so nur Muster berücksichtigt werden, die aus mindestens 5 identischen Regelabfolgen entstanden sind.
Die Berechnung von F_3 geschieht mit folgender Formel:

$$F_3 = \frac{\sum_{i \in Mustermenge} size_i^2 * count_i}{\#\, aller\ Knoten\ im\ Graphen}$$

Hierbei ist $size_i$ die Größe des gefundenen $i$ten Musters (Anzahl der Regeln) und $count_i$ die Häufigkeit, mit der dieses Muster i im gesamten Graphen aufzufinden ist. Die Mustergröße geht quadratisch in die Fitness ein, da die Wahrscheinlichkeit für große Muster deutlich geringer ist als für kleine Muster und große Strukturen somit stärker belohnt werden.

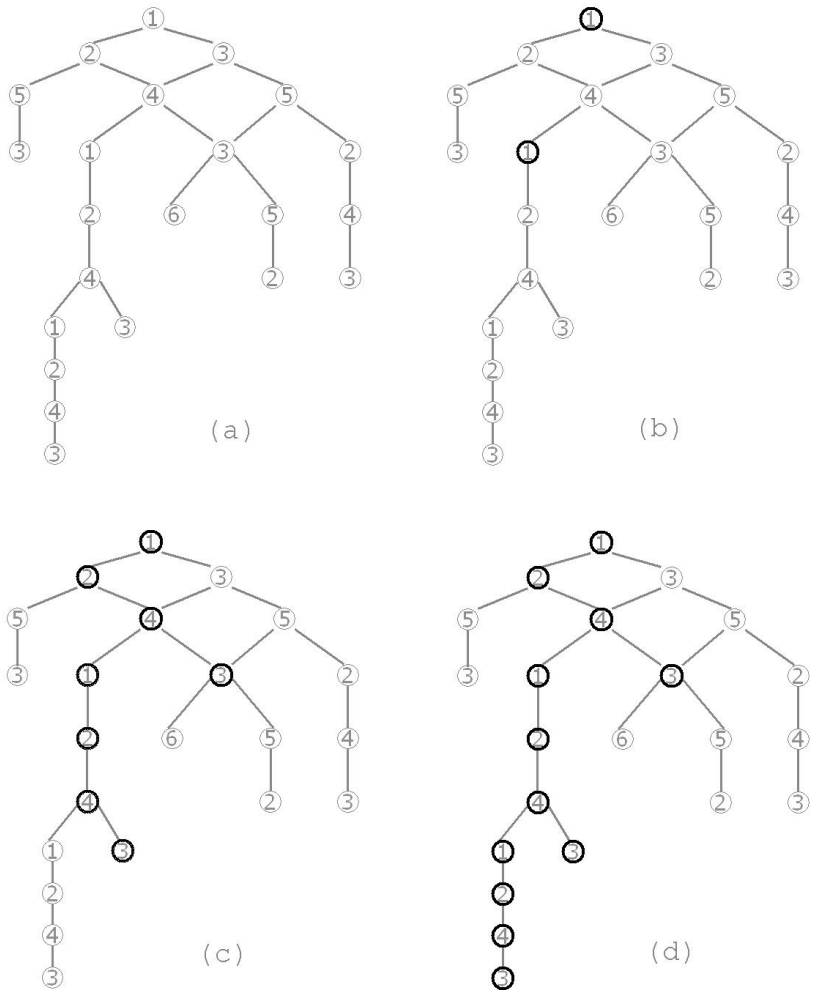

Abbildung 4.4.: **Strukturfindung im ConstructionGraph**. Der *ConstructionGraph* (a) wird, bei der Wurzel beginnend, durchsucht. Tritt eine Knotennummer doppelt auf (b), werden die darunter liegenden Knoten jeweils solange zu einem Muster hinzugefügt, wie sie unter beiden Knoten gleichzeitig vorhanden sind (c). Ist auf diese Art ein einzelnes Muster bestimmt, wird das Vorkommen dieses Musters im gesamten Graphen gezählt (d).

4.3.4.2. Bewertungsfunktion Flaschenhals-Findung

Diese Bewertungsfunktion ist eine Erweiterung der vorhergehenden Strukturfindungs-Funktion (siehe Abschnitt 4.3.4.1). Sie ist aufgrund von Beobachtungen von Bonabeau und während dieser Diplomarbeit durchgeführter Versuche entstanden.
Ein wichtiges Kriterium zur Reproduzierbarkeit von strukturierten Nestern ist das Vorhandensein von sogenannten Bottleneck-Regeln. Das sind Regeln, die eine Art Zwischenstand der Simulation repräsentieren und erst dann aktiviert werden, wenn eine Teilstruktur vollständig erzeugt wurde. Durch diese speziellen Regeln kann der gesamte Bauprozess geordneter ablaufen, weil eine Art Reihenfolge der Bauschritte festgelegt wird. Wenn viele Bottleneck-Regeln existieren, ähneln sich die Strukturen, die aus ein und derselben Regelmenge entstehen stärker, als wenn wenige oder keine Bottleneck-Regeln vorhanden sind.

Vorgehen (F_4) Die Bewertungsfunktion aus Abschnitt 4.3.4.1, die aus den Teilen F_1 - F_3 besteht, wird hier um den Teil F_4 erweitert, der die Ähnlichkeiten zwischen den erzeugten Strukturen bestimmt:

$$F_4 = \frac{\#\ identischer\ Bloecke\ in\ allen\ Strukturen}{\#\ aller\ Bloecke\ in\ allen\ Strukturen}$$

In diesem Teil wird ausgewertet, inwieweit sich die einzelnen Strukturen gleichen, die bei unterschiedlichen Durchläufen einer einzelnen Regelmenge erzeugt wurden, indem die Anzahl der Blöcke gezählt wird, die in *allen* Strukturen vorkommen und diese ins Verhältnis gesetzt werden zu der Gesamtanzahl von Blöcken, die in allen Durchläufen insgesamt platziert wurden.

Fitness Die Gesamtfitness berechnet sich demnach mit Hilfe der Formel

$$Fitness\ F = \prod_{i=1}^{4}(f_i * F_i^{p_i})$$

wobei die Teile F_1 - F_3 aus Abschnitt 4.3.4.1 übernommen werden. Die Werte, die der Teil F_4 annehmen kann, bewegen sich im Bereich $[0, 1]$.

4.3.4.3. Bewertungsfunktion PatternMatching

Im ersten Abschnitt 4.3.4.1 wurden zufällig generierte Nester aufgrund ihrer Strukturiertheit bewertet. Mit der hier beschriebenen Bewertungsfunktion *PatternMatching* wird der umgekehrte Weg eingeschlagen. Es wird versucht, von Partikelschwärmen ein vorgegebenes Muster nachbilden zu lassen.
Ziel dieser Untersuchungen ist es, herauszufinden, wie und ob Partikelschwärme auf die Erfüllung eines bestimmten Planes trainiert werden können. Im Idealfall existieren nach Berechnung mehrerer Generationen Mengen kleinstmöglicher MicroRule-Kombinationen, denen es gelingt, ein gewünschtes Muster schnell und effektiv zu erzeugen.

Vorgehen Als Eingabe für die Simulation wird zusätzlich zu den Simulationsparametern ein Muster angegeben, auf welches die einzelnen Partikelschwärme trainiert werden sollen. Ein Muster ist eine Menge von Blockpositionen, die erzeugt werden sollen. Es ist dabei nicht von Bedeutung, ob die Blockarten von gewünschtem Muster (Plan) und erzeugter Struktur übereinstimmen, wichtig ist, dass die erzeugte Struktur möglichst das gewünschte Muster abdeckt.
Grundsätzlich kann zwischen zwei Abdeckungen unterschieden werden. Die Auswahl der gewünschten Methode geschieht durch Simulationsparameter:

Exakte Abdeckung Hier muss das Muster exakt abgebildet werden. Jeder in der erzeugten Struktur fehlende Block führt zu Abstrichen in der Fitnessfunktion genauso wie jeder zusätzliche Block. Die Fitness wird nach folgender Formel berechnet

$$F = \frac{\#\, korrekt\, platzierte\, Bloecke - \#\, falsch\, platzierte\, Bloecke}{\#\, Bloecke\, im\, gewuenschten\, Muster}$$

wobei die minimal erreichbare Fitness 0 beträgt, das heißt, negative Werte werden auf 0 erhöht.

Überlappende Abdeckung Diese Fitnessfunktion ist weniger restriktiv, die Bedingung ist hier, dass wenigstens das gewünschte Muster abgedeckt wird, zusätzliche Blöcke verringern die Fitness nicht:

$$F = \frac{\#\, korrekt\, platzierte\, Bloecke}{\#\, Bloecke\, im\, gesuchten\, Muster}$$

Durch diese Berechnungsvorschrift bewegt sich der Fitnesswert in den Grenzen $[0; 1]$.

Die Abdeckungsart liefert je nach Plan unterschiedlich gute Ergebnisse. Wenn Muster erzeugt werden sollen, die zeitabhängig sind, das heißt es sind endliche Strukturen, die nicht bis zu einem Rand des Simulationsraumes reichen, kann das Platzieren von Blöcken nicht immer durch Regeln gestoppt werden, da die Partikelschwärme aufgrund der fehlenden globalen Sicht und des fehlenden internen Speichers keine Zählmöglichkeit besitzen. Aus diesem Grund kann in Simulationen nicht unbedingt eine exakte Abdeckung erreicht werden. In solchen Fällen bietet sich die *Überlappende Abdeckung* an. Wenn allerdings Muster generiert werden sollen, die durch die Simulationswelt begrenzt sind, besteht keine Zähl-Notwendigkeit, hier kann die *Exakte Abrenzung* gegebenenfalls bessere Ergebnisse liefern.

Fitness Die Fitness wird abhängig von der Abdeckungsart mit den oben angegebenen Formeln berechnet. Ein korrekt platzierte Block ist dabei ein während der Simulation abgelegter Block beliebiger Art (Typ), der sich an einer Stelle befindet, die in dem gesuchten Muster spezifiziert wurde. Um in der Notation der unter Abschnitt 4.3.4.1 eingeführten Fitness zu bleiben, wird dieser Fitnesswert mit F_1

bezeichnet und kann durch einen beliebigen Faktor und Exponenten noch weiter skaliert werden. Die Gesamt-Fitness wird also auch hier nach der Formel

$$Fitness\ F = \prod_{i=1}^{3}(f_i * F_i^{p_i})$$

berechnet mit $F_2 = F_3 = 1$.

4.3.4.4. Genetische Operatoren

Mit Hilfe von genetischen Operatoren (Crossover, Selektion, Mutation) sollen die einzelnen Regelmengen in Bezug auf die aufgestellte Fitnessfunktion von Generation zu Generation verbessert werden, das heißt, der Fitnesswert der Nester soll steigen. Im Folgenden ist ein Individuum im Sinne des genetischen Algorithmus eine Menge von MicroRules, die bei der Erzeugung eines Nestes genutzt werden; ein Genom des Individuums ist eine einzelne MicroRule. Die gesamte Population ist die Menge von MicroRules-Mengen, die simuliert werden, also die Anzahl von Strukturen, die mit Hilfe der Client-Rechner erzeugt werden.
Um eine neue Generation aus einer alten zu erzeugen, werden dabei nachstehend angegebene Schritte durchgeführt:

1. Erzeugen von Nachkommen durch Crossover zweier erfolgreicher Individuen aus der alten Generation und Einfügen dieser beiden Nachkommen in die neue Generation.

2. Selektion von Individuen aus der alten Generation und Einfügen in die neue Generation, bis die gewünschte Populationsgröße erreicht ist.

3. Veränderung einiger Individuen der neuen Generation durch Mutation.

Die Durchführung der Schritte 1 und 3 wird dabei durch Wahrscheinlichkeitswerte gesteuert.

4.3.4.5. Selektion

Die Selektion wird an zwei Stellen genutzt. Einmal werden für eine Crossover-Operation jeweils zwei Individuen selektiert, des Weiteren werden Individuen aus der alten Generation selektiert, um die neue Generation aufzufüllen und so die Anzahl der Individuen, das heißt die Größe der Population auf einem konstanten Wert zu belassen.
Die Selektion wird bei *VisualSwarm* nach dem *roulette-wheel* Verfahren [Gol89] durchgeführt, das heißt, die Wahrscheinlichkeit, dass ein Individuum selektiert wird, ist proportional zu seiner Fitness. Dabei können besonders fitte Individuen mehrmals selektiert werden und damit zum Beispiel mehr Nachkommen produzieren als weniger fitte.

Realisiert wurde die fitnessabhängige Selektion durch eine Skalierung der Summe aller Fitnesswerte der Individuen einer Population zu 1. Eine Zufallszahl p bestimmt dann exakt das zu selektierende Individuum i_0, für das gilt:

$$\sum_{i=0}^{i_0}(Fitness_i) \geq p \geq \sum_{i=0}^{i_0-1}(Fitness_i)$$

4.3.4.6. Crossover

Die Crossover-Operation eines genetischen Algorithmus ist die Genom-Kombinierung zweier Eltern-Individuen zu neuen Kinder-Individuen. Auf welche Art und Weise diese Vereinigung erfolgt, ist durch eine Anzahl von Crossover-Punkten festgelegt. Eine Anzahl von 2 bedeutet zum Beispiel, dass die Bitmuster beider Eltern-Individuen an zwei identischen Stellen aufgeteilt und für die Kinder-Individuen vermischt werden. Dieses Vermischen geschieht nach folgendem Prinzip:

- Festlegung von Crossover-Punkten, die gültige Bitpositionen für *beide* Elternteile darstellen. Diese Einschränkung ist notwendig, da die Menge von MicroRules der beiden jeweiligen Elternteile unterschiedlich groß sein kann.
- Kopieren des Bitmusters vom ersten Elternteil bis zum ersten Crossover-Punkt in das erste Kind-Individuum, gleichzeitig Kopieren des Bitmusters vom zweiten Elternteil bis zum ersten Crossover-Punkt in das zweite Kind-Individuum. An dieser Stelle werden die beiden Kinder vertauscht und mit dem Kopieren des Bitmusters bis zum nächsten Crossover-Punkt fortgefahren. Dort werden die Kinder wieder vertauscht und so weiter.

Die Bestimmung der Crossover-Punkte geschieht bei jeder Crossover-Operation neu, sodass die selben Elternteile unterschiedliche Nachkommen produzieren können.

Beispiel:

Elternteil I: 10101 0101 01010101010
Elternteil II: 11111 1111 00000000111

Crossover-Punkte: nach Bit 5 und nach Bit 9

Kind I: 10101 1111 01010101010
Kind II: 11111 0101 00000000111

Die Anzahl der gewünschten Crossover-Punkte kann beliebig gewählt werden, allerdings hat sich bei unterschiedlichen Versuchsreihen gezeigt, dass sie sehr gering (1 - 2 Crossover-Punkte) gehalten werden sollte, da die in einem Individuum enthaltenen MicroRules gegebenenfalls aufeinander aufbauen bzw. voneinander abhängen können, um erfolgreiche Strukturen zu kreieren; durch Crossover-Operationen können diese Zusammenhänge schnell auseinander gerissen werden.

Die Wahrscheinlichkeit, dass eine Crossover-Operation durchgeführt wird, kann ebenfalls durch Parameter spezifiziert werden. Ein Wert von 0.2 bedeutet in diesem Zusammenhang beispielsweise, dass etwa jedes 5. Individuum in eine Crossover-Operation verwickelt ist, oder, anders ausgedrückt, dass etwa jedes 5. Individuum der neuen Generation durch Crossover erzeugt wird. Eine Crossover-Operation generiert dabei immer zwei Nachkommen, die direkt in die neue Generation von Individuen eingefügt werden.
Individuen, die in eine Crossover-Operation verwickelt waren, können bei der späteren Selektion zusätzlich auch noch in die neue Generation gelangen. Dadurch wird ermöglicht, dass durch eine Crossover-Operation eventuell zerstörte MicroRule-Zusammenhänge auch in der neuen Generation noch weiter existieren können.

Sortierung der Gene Die erwähnten Probleme, dass bei Crossover-Operationen MicroRules, die aufeinander aufbauen, auf zwei unterschiedliche Nachfahren verteilt werden und damit eine erfolgreiche Struktur, die die Eltern-Individuen erzeugt haben, von keinem der beiden Kinder-Individuen erzeugt werden kann, können durch eine Sortierung der MicroRules innerhalb der jeweiligen Individuen vermindert werden. Dazu kann in der Konfigurationsdatei eine Ordnung für die Gene eingestellt werden. Diese bewirkt, dass beide Elternteile, die an einer Crossover-Operation beteiligt sind, vor der Kombination auf unterschiedliche Art und Weise sortiert werden. Die Regeln im Vater werden so angeordnet, dass alle in der letzten Simulation aktivierten MicroRules an den Anfang gesetzt werden. Dem entsprechend werden bei der Mutter alle aktivierten Regeln am Ende der Regelmenge zusammengefasst. Die beiden Vorfahren haben damit die Form:

```
Vor der Sortierung:
* Vater:  NAANA.....NAANN
* Mutter: AANNN.....NANAN


Nach der Sortierung:
* Vater:  AAAAA.....NNNNN
* Mutter: NNNNN.....AAAAA


A: aktivierte Regel
N: nicht aktivierte Regel
```

Dieses Vorgehen hat den Vorteil, dass bei Crossover-Operationen mit einer ungeraden Anzahl von Crossover-Punkten die aktivierten Regeln *beider* Elternteile vorwiegend im ersten Nachfahren kombiniert werden. Der zweite Nachfahre erhält dagegen eine deutlich geringere Anzahl aktivierter Regeln.
Wenn eine Sortierung der Gene in einer Simulation eingeschaltet wird, sollte niemals eine gerade Anzahl von Crossover-Punkten gewählt werden, da in diesem Fall die

Nachfahren nur mit geringer Wahrscheinlichkeit eine Kombination der Fähigkeiten der Vorfahren repräsentieren sondern eher die Elternteile exakt abbilden, weil mehr nicht-aktivierte Regeln vermischt werden als aktivierte[5].

4.3.4.7. Mutation

Die Mutation verändert jedes Individuum in der neu zusammengestellten Generation mit einer gewissen Wahrscheinlichkeit. Bei der Mutation der einzelnen Bits (= MicroRules) eines Individuums wird dabei unterschieden, ob die entsprechenden MicroRules bei der letzten Simulation angewandt wurden oder nicht. Genutzte MicroRules werden mit einer deutlich geringeren Wahrscheinlichkeit mutiert und damit durch neue MicroRules ersetzt als ungenutzte MicroRules. Dadurch bleiben erfolgreiche Regeln mit größerer Wahrscheinlichkeit erhalten. In den durchgeführten Versuchen betrug die Wahrscheinlichkeit, ungenutzte MicroRules auszutauschen, 0.9, die Wahrscheinlichkeit, genutzte auszutauschen, 0.01.
Die Mutation tauscht bei *VisualSwarm* vollständige Regeln aus, diese Vorgehensweise ist an Bonabeau angelehnt. Vorstellbar wäre auch eine Veränderung der existierenden Regeln, indem durch Anpassung der Nachbarschaftsblockmuster eine Aktivierung in der nächsten Generation unter anderen Bedingungen stattfinden kann.

4.3.5. „Simulated Annealing“ Suche

Im vorhergehenden Abschnitt wurde eine Durchsuchung des Lösungsraumes mit Hilfe von evolutionären Algorithmen beschrieben. Derselbe Algorithmus, der dort angewandt wurde, kann auch genutzt werden, um eine Suche, die sich ähnlich verhält wie *Simulated Annealing*, durchzuführen.
Ziel dieses Algorithmus ist es, am Beginn einer Simulation durch sehr starke Veränderungen der Regelmenge eines Individuums große und weit auseinanderliegende Teile des Lösungsraumes zu besuchen. Diese Veränderungen können durch eine hohe Mutationsrate sowohl bei den genutzten als auch bei den ungenutzten MicroRules erreicht werden. Durch die Selektion werden die besser bewerteten Regelmengen statistisch gesehen häufiger in die nächste Generation übernommen als weniger erfolgreiche. Von einer Generation zur nächsten wird die Mutationswahrscheinlichkeit immer geringer, das gesamte System kühlt aus und stabilisiert sich letztendlich auf einzelnen lokalen Extremstellen, da irgendwann die Wahrscheinlichkeit für die Mutation von Regeln gegen *null* geht.
Die Mutationsrate in der Generation i wird bei dem hier eingesetzten Algorithmus aufgrund der folgenden Formel berechnet:

$$Mutationsrate_i = Mutationsrate_{i-1} * Auskuehlungsfaktor$$

[5]Das erste Individuum erhält mit großer Wahrscheinlichkeit die aktivierten Regeln des Vaters und nur mit geringer Wahrscheinlichkeit die aktivierten Regeln der Mutter, beim zweiten Nachfahren ist es genau umgekehrt

Der $Auskuehlungsfaktor$ ist dabei eine reelle Zahl aus $[0;1]$ und die initiale Mutationsrate wird aus der XML-Konfigurationsdatei ausgelesen.
Um bei diesem Vorgehen das „Simulated Annealing“-Prinzip annähernd nachzubilden, ist es notwendig, dass zur Bildung einer neuen Generation der bei der evolutionären Suche angewandte Crossover-Operator nicht eingesetzt wird. Dazu können die Wahrscheinlichkeit für den Einsatz dieses genetischen Operators in der Konfigurationsdatei oder die Anzahl der Crossover-Punkte auf *null* gesetzt werden.
Die für diesen Algorithmus wichtigen Konfigurationsdatei-Einträge sind:

used_annealing Der Abkühlungsfaktor für die Wahrscheinlichkeit, die den Austausch (Mutation) genutzter MicroRules steuert

unused_annealing Der Abkühlungsfaktor für die Wahrscheinlichkeit, die den Austausch (Mutation) ungenutzter MicroRules steuert

4.4. Programmsteuerung und Hilfsmittel

Das Softwarepaket *VisualSwarm* besteht grundsätzlich aus zwei Teilen, einmal dem *Server*, der Steuerungs- und Verteilungsaufgaben übernimmt, und dem *Client*, der die eigentliche Berechnung eines einzelnen Nestes durchführt. Client-Instanzen können in beliebiger Anzahl gestartet werden, der Server versucht, alle Aufgaben gleichmässig an diese zu vergeben[6].
Sowohl für den Server als auch für den Client existieren Konfigurationsdateien, die alle Simulationsparameter enthalten. Beispiele für solche Parameter sind Faktoren für die Fitnessfunktion oder Angaben über den Kommunikationsport zwischen Server und Client. Eine detaillierte Auflistung aller möglicher Parameter ist im Anhang A zu finden.
Die Konfigurationsdatei ist ein XML-Dokument, welches beim Programmstart interpretiert wird. XML bietet den Vorteil, dass es plattformunabhängig ist und leicht manuell verändert werden kann. Zusätzlich zu den Simulationsparametern bietet die Konfigurationsdatei die Möglichkeit, MicroRules manuell in die initiale Population einzusetzen. Durch diese Regeln kann die Berechnung von Strukturen durch spezielle Vorgabemuster fokussiert werden. Bonabeau hat hiervon in seinen Arbeiten ausgiebig Gebrauch gemacht und teilweise sogar vollständig auf evolutionäre Anpassungen der Regeln verzichtet [BGS+00].

4.4.1. Graphical User Interface - ThreadObserver

Zur Visualisierung der Zwischenergebnisse und der verteilten Berechnungen existiert ein Graphical User Interface (*ThreadObserver*), dieses ermöglicht schon während der Simulation, einzelne Strukturen zu betrachten. Außerdem gibt es Auskunft darüber,

[6] Bei großen Berechnungen bietet es sich sogar an, mehr Clients zu starten als die Populationsgröße beträgt, da so flexibel auf den Ausfall einzelner Clients reagiert werden kann

auf welchen Clients und damit auf welchen Rechnern die einzelnen Berechnungen durchgeführt und welche Zeiten dazu benötigt wurden.
Um die Entwicklung der durchschnittlichen Fitness der einzelnen Generationen zu verfolgen, wurde ein Diagramm entwickelt, das diese zu jedem beliebigen Zeitpunkt während der Gesamtsimulation anzeigt. Hierzu ist das Softwarepaket *jFreeChart* genutzt worden, das flexible Schnittstellen für die Visualisierung von Daten zur Verfügung stellt. Abbildung 4.5 zeigt die GUI während der Laufzeit einer komplexen Simulation von 80 Individuen.

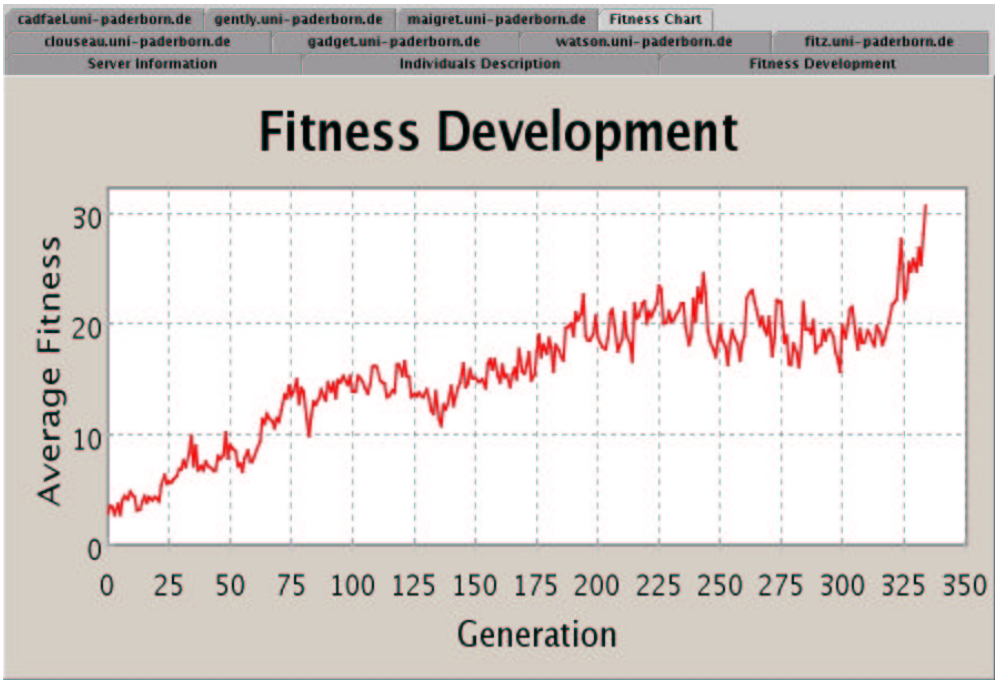

Abbildung 4.5.: **Die GUI des Servers** Dargestellt wird das User Interface während einer komplexen Berechnung, die einzelnen Panels geben Auskunft über die Worker-Prozesse und die darauf ausgeführten Berechnungen. Zusätzlich existieren noch Panels, die die durchschnittliche Entwicklung der Fitness der einzelnen Generationen aufzeigen und eine Liste aller berechneten Nester, sortiert nach Fitness, die Möglichkeiten zur Visualisierung der Strukturen bietet.

4.4.2. Hilfsmittel

Die Software *VisualSwarm* nutzt einige Hilfsprogramme / Tools. Im Folgenden werden diese kurz beschrieben:

JFreeChart JFreeChart [G+02] ist eine Java Klassenbibliothek, die es ermöglicht, beliebige Datensätze in unterschiedlichsten Diagrammaufbereitungen darzustellen. Die Bibliothek ist unter der Lesser General Public License (LGPL)

freigegeben und daher kostenlos nutzbar. Sie wird hier eingesetzt, um die Entwicklung der durchschnittlichen Fitness jeder Generation über die gesamte Simulationsdauer visuell in einem Liniendiagramm darzustellen.

POV-Ray Die während der Simulation erzeugten Strukturen können als POV-Ray [C+99] Beschreibungsdateien abgelegt werden. Sie ermöglichen so eine übersichtliche Betrachtung der erzeugten Strukturen. POV-Ray ist ein mächtiger RayTracer, dessen Beschreibungsdateien einfach generiert und jederzeit manuell angepasst werden können. Zusätzlich bietet POV-Ray noch die Möglichkeit, Animationen zu erstellen. Dadurch können Strukturen von allen Seiten betrachtet und so noch besser beurteilt werden. Für diese zu berechnende Animation werden zusätzlich zu den Szenenbeschreibungsdateien noch Sequenzbeschreibungen speziell für POV-Ray in einer separaten Datei angelegt.
Ausschnitt aus einer POV-Ray Beschreibungsdatei (eine komplette Datei ist im Anhang C zu finden):

```
background
{
color White
}

camera
{
location <8+anim, -16+anim, -6+anim>
look_at <8, 8, 8>
}

light_source
{
<8, -16, -8> color White
}

box
{
<8, 8, 1>,
<8.96, 8.96, 1.96>
texture
{
pigment {color LightGray}
}
}
```

SWARM Eine am Santa Fe Institute entwickelte Multi-Agenten Simulationsumgebung (siehe Abschnitt 4.4.3).

Xerces Sowohl die Kommunikation zwischen Server und Worker-Prozessen als auch die Parameterisierungen von Server und Worker geschehen mit Hilfe von XML-Datenströmen. Diese Datenströme werden mit Hilfe des Xerces-Parsers von Apache [Apa02] geparst und damit interpretiert.

4.4.3. Die SWARM Simulationsumgebung

Am Santa Fe Institute wurde zur Simulation von Multiagentensystemen ein Softwarepaket[7] entwickelt, welches im Verlauf dieser Diplomarbeit als Hilfsmittel eingesetzt worden ist. SWARM ist kostenlos erhältlich und wird seit 1994 unter der GNU-Lizenz weiterentwickelt. Es hat bereits Einsatz in den unterschiedlichsten Gebieten gefunden, zum Beispiel bei ökonomischen Simulationen, Simulationen im Bereich der Chemie, der Ökologie und der Soziologie.
Dieser Abschnitt gibt einen Einblick in die Ideen, die zur Entwicklung dieses Paketes geführt haben und untersucht, ob es für zukünftige Aufgaben auf diesem Gebiet geeignet ist.
Für die Durchführung eines Experimentes im Bereich der Agenten basierten Modellierung sind zwei Dinge notwendig [Ter98]:

1. Die Definition des experimentellen Vorgehens des Computers

2. Die Software-Implementierung des Problems

Ein Großteil des Zeitaufwandes bei der Simulation von Multiagentensystemen wird für Punkt *(1)* - die Entwicklung einer Simulationsumgebung, die speziell auf das zu untersuchende Problemfeld zugeschnitten ist - aufgewandt. Dadurch ist es in anderen Experimenten häufig nicht möglich, die Simulationsumgebung oder größere Teile daraus wiederzuverwenden. Das SWARM Simulationspaket ermöglicht es Wissenschaftlern, sich bei experimentellen Untersuchungen von Schwärmen ganz auf die Schwärme beziehungsweise Agenten selbst zu konzentrieren, da die Verwaltung und eigentliche parallele Simulation von der Simulationsumgebung durchgeführt wird. Die Aufgabe zur Durchführung eines Experimentes besteht bei Einsatz von SWARM hauptsächlich darin, das benötigte logische Modell auf das SWARM-Modell abzubilden.
Der Einsatz einer einheitlichen Simulationsumgebung wie zum Beispiel SWARM hat zusätzlich den Vorteil, dass durchgeführte Experimente jederzeit nachvollzogen und Ergebnisse dadurch von unterschiedlichen Parteien verifiziert werden können. Ohne eine einheitliche Simulationsumgebung ist es kaum möglich und sinnvoll, alle Angaben zur Simulationsumgebung in das zu einem Experiment veröffentlichte Papier aufzunehmen [Ter98], [MBLA96], [Swa00].
R. Hoar beschreibt in [Hoa] weitere Simulationsumgebungen und stellt Unterschiede und Probleme der einzelnen Softwarepakete einander gegenüber.

[7] http://www.santafe.edu/projects/swarm/

4.4.3.1. Elemente von SWARM

Die kleinste Einheit in einer SWARM Simulation ist ein einzelner Agent. Eine Menge von Agenten kann zu Schwärmen zusammengefasst werden, wobei auch Schwärme wieder als einzelne Elemente (Agenten) von übergeordneten Schwärmen aufgefasst werden können; dadurch lassen sich beliebig komplexe Hierarchien bilden (siehe Abbildung 4.6). Ein Schwarm ist also eine lockere Zusammenfassung von Objekten zu

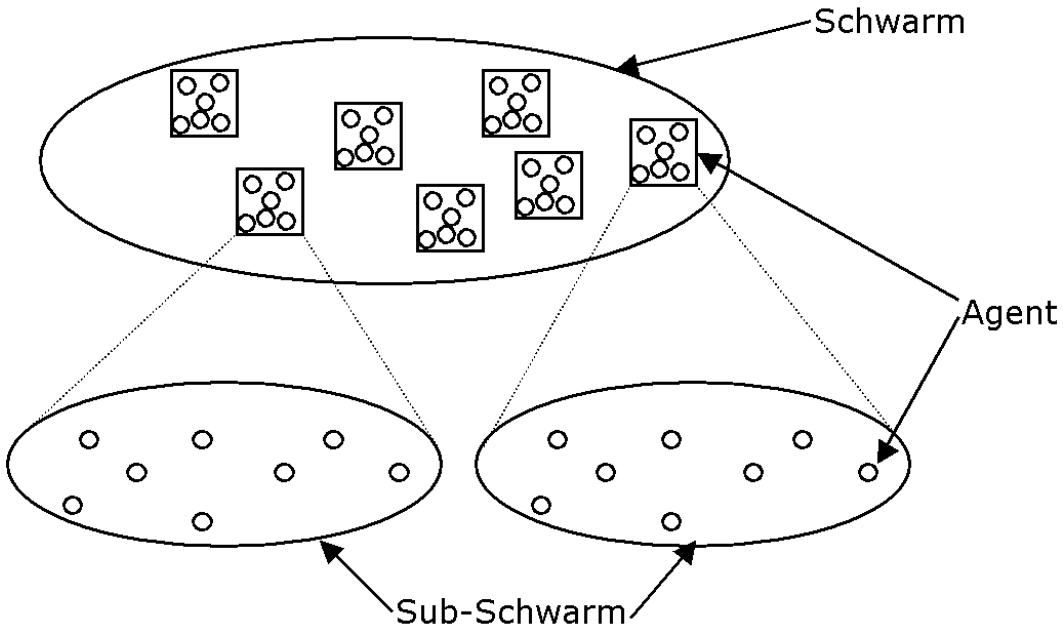

Abbildung 4.6.: Schwarm-Hierarchien

einer logischen Einheit. Sowohl für einzelne Agenten als auch für gesamte Schwärme können Befehlsketten (Actions) angegeben werden, die zu speziellen Zeitpunkten oder in speziellen Zeitintervallen mit Hilfe eines Schedulers ausgeführt werden. Diese Aktionen sind jeweils diskrete Ereignisse, die in einer vorher festgelegten Reihenfolge abgearbeitet werden.
Durch diese Möglichkeit der Komposition und damit Abstraktion lassen sich sehr komplexe Systeme modellieren, die trotzdem noch überschaubar bleiben, da bei Untersuchungen eines Schwarms nicht alle einzelnen Abstraktionsebenen benutzt werden müssen. So ließe sich zum Beispiel ein Mensch als eine Menge von Körperteilen modellieren, wobei die Körperteile selbst wiederum als eine Menge von Zellen gesehen werden können, diese wiederum als eine Menge von Atomen und so weiter. Die Umwelt selbst, in der sich die Agenten bzw. Schwärme bewegen, ist nicht beschränkt, sie ist vielmehr selbst ein Agent und kann beliebige Aktionen zu beliebigen Zeitpunkten ausführen. Dadurch hat man auch an dieser Stelle eine größtmögliche Freiheit und ist nicht auf vorgefertigte Objekte beschränkt.

4.4.3.2. Simulationen mit SWARM

Die Simulation einer Menge von Agenten geschieht in diskreten Zeitschritten. Zu jedem Zeitpunkt wird von SWARM für jeden Agenten untersucht, ob dieser eine Aktion oder eine Aktionskette ausführt. Repräsentiert ein Agent einen gesamten Unterschwarm, wird die Aktion für jeden einzelnen Agenten in diesem Unterschwarm ausgeführt.
Um die Entwicklung der Agenten beobachten zu können, existieren so genannte *Observer Swarms*. Diese ermöglichen eine genaue Überwachung untergeordneter Agenten, ohne sie in ihren Handlungen zu beeinflussen („*a simulated world under glass*“ [MBLA96]).
Damit ergibt sich das folgende Gerüst für das experimentelle Vorgehen bei Simulationen mit SWARM nach [Ter98] und [Swa00]:

1. Erzeugung einer künstlichen Umgebung, der darin enthaltenen Agenten und eines Zeit-Objektes. Die Agenten haben die Möglichkeiten, abhängig von ihren eigenen Regeln und den Vorgaben der Umwelt zu agieren.

2. Erzeugung von Objekten, welche die Daten der in Schritt *(1)* erzeugten Objekte beobachten, speichern und analysieren.

3. Simulation der Umgebung, indem sich sowohl die Agenten als auch die beobachtenden Objekte gleichzeitig in der Zeit vorwärts bewegen.

4. Interaktion mit der Simulation über die produzierten Daten.

Da SWARM vollständig objektorientiert aufgebaut ist, können bei der Entwicklung alle Vorteile der *OO*-Programmierung (Vererbung, Kapselung von Variablen, Mehrfachverwendung von Code etc.) gegenüber der traditionellen prozeduralen Programmierung genutzt werden. Außerdem können durch so genannte *Probes* eine beliebige Auswahl der vorhandenen Daten jedes einzelnen Objektes während der Simulation angezeigt und verändert oder einzelne Funktionen auf den Objekten manuell ausgeführt werden. Die graphischen Auswertungshilfsmittel, die SWARM zur Verfügung stellt, machen ebenfalls Gebrauch von diesen *Probes* und erlauben somit eine verständliche und übersichtliche Repräsentation des Zustandes der Simulation zu jedem beliebigen Zeitpunkt[8].
SWARM stellt somit Werkzeuge bzw. Hilfsmittel für das Design, die Implementierung, den Simulationsablauf und die Analyse zur Verfügung, um nebenläufige, verteilte künstliche Welten zu berechnen [LMB95].

4.4.3.3. Probleme beim Einsatz von SWARM

SWARM wurde in der Implementierung zu dieser Diplomarbeit erfolgreich eingesetzt, um die einzelnen Struktur-erzeugenden Agenten in einer dreidimensionalen

[8] SWARM nutzt zur graphischen Visualisierung TCL/TK und ist sowohl unter Unix als auch auf Windows Systemen lauffähig

Umwelt zu bewegen. Allerdings mussten bereits hierbei umfassende Ergänzungen zum SWARM-Paket vorgenommen werden, da dieses nur eine 2-dimensionale Umwelt unterstützt. Auch wurden die graphischen Hilfsmittel, die das Fortschreiten der Simulation anzeigen und die diversen Analysemöglichkeiten zur Verfügung stellen, nicht benötigt, sodass sich der von *VisualSwarm* genutzte Funktionsumfang auf ein Minimum beschränkt. Diese Funktionalitäten sind in einer einzelnen Java-Klasse gekapselt, sodass leicht auf eine andere - gegebenenfalls neu zu entwickelnde und auf die unterschiedlichen Bedürfnisse angepasste - Simulationsumgebung zurückgegriffen werden kann.
Ein weiteres Problem stellte die Installation von SWARM dar, da sie nur unter Schwierigkeiten durchgeführt werden konnte, was auch an der spärlichen Dokumentation lag, die im Großen und Ganzen nur aus einer JavaDoc-Hilfe bestand mit teilweise fehlender Dokumentation von einzelnen Funktionen.
Da bei den zukünftigen Weiterentwicklungen der Software *VisualSwarm* der dreidimensionale Aspekt stärker berücksichtigt werden wird und kaum zusätzliche von SWARM angebotene Funktionen benötigt werden, ist die Neuentwicklung einer schmaleren Simulationsumgebung anzuraten, auch deshalb, weil beim Einsatz des SWARM-Pakets bei größeren Simulationen immer wieder Speicherfehler auftraten, die nicht behoben werden konnten und teilweise fatale Ausfälle produzierten. Außerdem ist man beim Einsatz von SWARM auf diskrete Zeitmodelle beschränkt, was gerade bei einer Umsetzung von Simulationsergebnissen in der realen Welt zu Problemen führen kann.

5. Simulationsergebnisse

In diesem Kapitel werden sowohl die Ergebnisse der Arbeiten von Eric Bonabeau als auch die mit der entwickelten Software *VisualSwarm* erhaltenen Erkenntnisse zusammengefasst und miteinander verglichen.

5.1. Ergebnisse von Bonabeau

5.1.1. Koordinierte Suche

Ein Hauptaspekt von Bonabeaus Arbeit war der Nachweis, dass mit dem Kommunikationsprinzip *Stigmergy* eine Erklärungsmöglichkeit für das effektive Zusammenarbeiten von einfachen, sozialen Individuen zur Erzeugung komplexer Strukturen gegeben werden kann. Dazu hat er eine Vielzahl von Strukturen, die in ähnlicher Form in der Natur vorkommen, mit einem sehr einfachen Modell und manuell erzeugten Regelmengen nachbilden können; beim Einsatz von hexagonalen Blöcken konnten beispielsweise Strukturen in kleinem Maßstab erzeugt werden, die in ähnli-

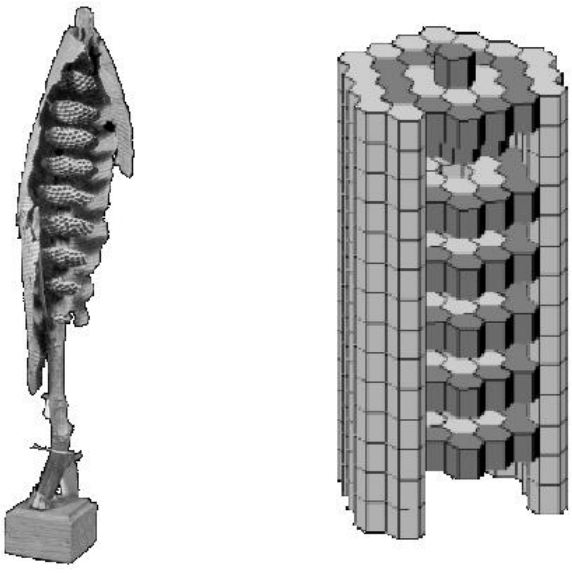

Abbildung 5.1.: Vergleich von einem natürlichem Nest und einer Struktur, die in einem sehr einfachen Modell durch eine Simulation erzeugt wurde.

cher Form von Wespen gebaut werden (siehe Abbildung 5.1), wobei die Eigenschaften und räumlichen Gegebenheiten der natürlichen Nester abstrahiert auch in den

vereinfachten Modellen vorgefunden werden konnten.
Eine wichtige Bedingung, um auch bei mehreren Simulationsläufen immer ein eindeutiges Ergebnis zu bekommen, war dabei nach Bonabeau das Vorhandensein von so genannten *Bottleneck MicroRules* [BGS+00]. Diese Regeln sorgen dafür, den Bauprozess zu koordinieren, indem sie Zwischenstände repräsentieren. Um einen solchen Zwischenstand zu erreichen, müssen erst durch eine beliebige Teilmenge von Agenten die Blöcke platziert worden sein, die die entsprechende Regel des Zwischenstandes aktivieren. Es ist dabei nicht notwendig, dass der Bauprozess immer vollständig identisch abläuft, vielmehr ist es wichtig, eindeutige und auch eindeutig geordnete Zwischenstände zu haben, um bei jedem Durchlauf identische Strukturen zu erhalten.

5.1.2. Evolutionäre Suche

Die Erzeugung von Regelmengen, die strukturierte Muster erzeugen können, realisierte Bonabeau, nachdem er anhand manuell generierter Regelmengen nachgewiesen hat, dass das Stigmergy-Konzept mächtig genug ist, mit Hilfe von evolutionären Algorithmen. Zur Bewertung der Strukturen, die von den einzelnen Regelmengen erzeugt wurden, benutzte er eine Fitnessfunktion, die der Strukturiertheit eines Musters eine reelle Zahl zuordnet. Bei dieser Bewertung wurden sowohl Kompaktheit der Struktur, Komplexität der beteiligten Regelmenge und Vorhandensein von wiederholten Teilstrukturen berücksichtigt.
Eine Schwierigkeit bei diesem Vorgehen war das abstrakte Kriterium der Strukturiertheit. In Abbildung 5.2 sind vier Strukturen zu sehen, die beiden oberen sind aus einer einzelnen, manuell erzeugten Regelmenge in zwei unterschiedlichen Simulationsdurchläufen entstanden, die beiden unteren wurden aus einer automatisch mit Hilfe von evolutionären Methoden gefundenen Regelmenge erzeugt. Auf den ersten Blick fällt auf, dass sich die beiden oberen Strukturen sehr ähnlich sind und dass sie insgesamt strukturiert / geordnet erscheinen. Die beiden unteren sind sich deutlich weniger ähnlich und sehen auch für einen menschlichen Betrachter weniger strukturiert aus. Bonabeaus Schwierigkeit bestand darin, diese für menschliche Bewerter offensichtlichen Unterschiede durch Algorithmen formal zu spezifizieren; durch seine hier vorgestellte Bewertungsfunktion, die auch als Grundlage für *VisualSwarm* genommen wurde, konnte das nur teilweise erreicht werden.
Bonabeau versuchte zusätzlich, die Zwischenergebnisse durch unterschiedliche *menschliche* Bewertungen zu ordnen und dadurch eine effektivere Fitnessfunktion zu entwickeln, dieses schlug aber fehl und führte zu keiner Verbesserung der Endergebnisse. Die menschlichen Bewertungen waren zu wenig eindeutig: Teilweise wurden besonders komplexe Formen als gut bewertet, teilweise ingenieurstechnisch einfach umsetzbare etc., sodass die Fitnessfunktion sich in keine spezielle Richtung entwickeln konnte [BDT99].
Ein Resultat von Bonabeau war die Einordnung der Gene (Regeln) bezüglich ihrer Funktion in den Regelmengen für die gesamte Struktur. Ob ein spezielles Gen (MicroRule) als *major gene*, *modifier gene* oder *neutral gene* wahrgenommen wird,

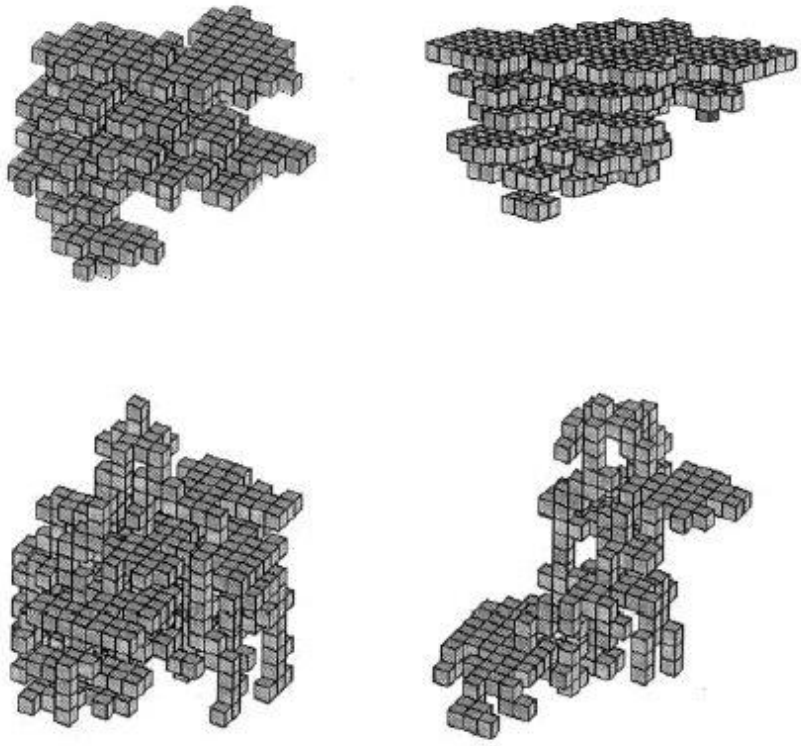

Abbildung 5.2.: Die oberen beiden Strukturen wurden durch eine manuell erzeugte Regelmenge in zwei unterschiedlichen Simulationsläufen gebaut. Die unteren beiden Muster sind durch eine Regelmenge erzeugt worden, die mit evolutionären Anpassungen entwickelt wurde.

hängt dabei vollkommen vom genetischen Umfeld ab, in dem es auftritt; diese Funktion kann sich im Verlauf der Evolution aber auch sehr schnell, das heißt von einer Generation zur nächsten, wieder verändern [BGS+00].

5.2. Ergebnisse mit VisualSwarm

5.2.1. Koordinierte Suche

Die koordinierte Suche mit manuell erstellten Regelmengen hat zusammengefasst dieselben Ergebnisse geliefert, die auch von Bonabeau herausgefunden wurden. Im Laufe dieser Arbeit wurden mehrere Strukturen, die identisch zu denen von Bonabeau sind, mit *VisualSwarm* nachgebildet. In Abbildung 5.3 sind einige dieser Muster zu sehen. Erzeugt wurden die Strukturen in dem beschriebenen einfachen Modell aus zwei unterschiedlichen Blockarten. Die Berechnungszeit bewegt sich für diese Muster auf einem einzelnen Personal Computer im kleineren Minutenbereich.

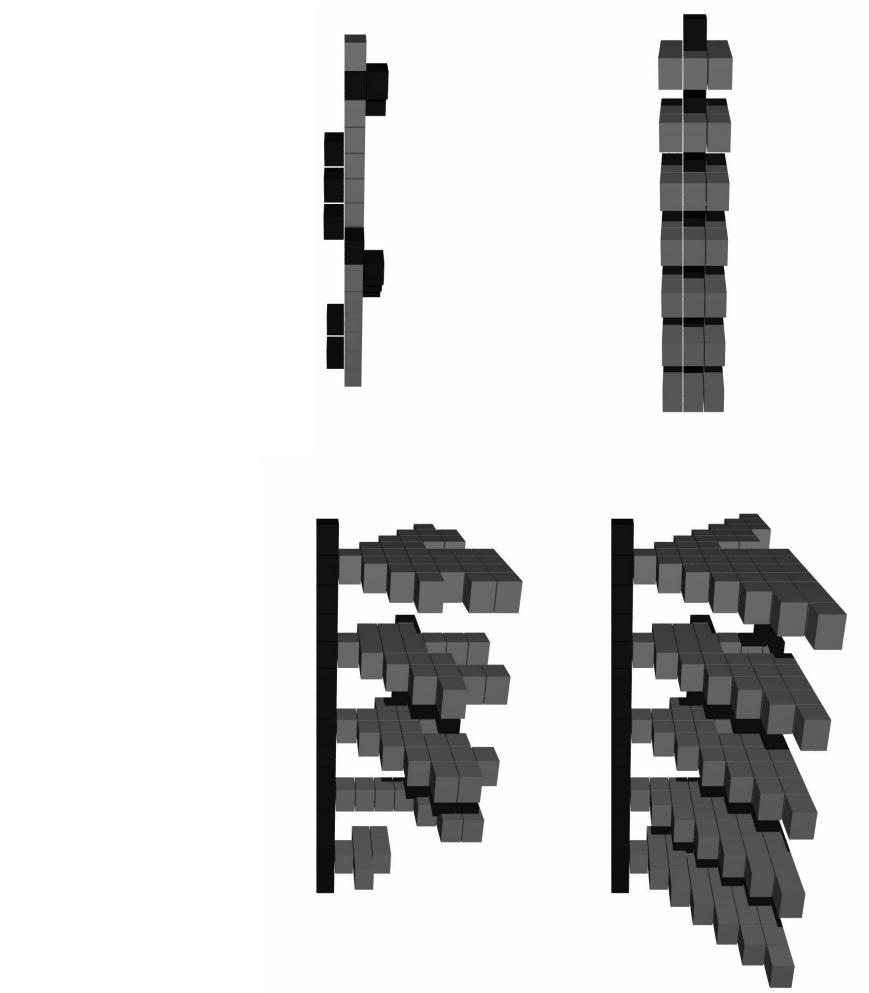

Abbildung 5.3.: Beispielstrukturen, die mit Hilfe von *VisualSwarm* durch manuell erzeugte Regelmengen (koordinierte Suche) generiert wurden

5.2.2. Evolutionäre Suche

Auch bei dieser Suchmethode wurde versucht, mit *VisualSwarm* die Ergebnisse von Bonabeau nachzuvollziehen. Da einige der erzeugten Strukturen signifikante Ähnlichkeiten zu Bonabeaus Resultaten aufweisen, kann dieser Versuch als erfolgreich bezeichnet werden, da eine exakte Reproduktion der Ergebnisse schon aufgrund des riesigen Lösungsraumes nicht möglich ist. Allerdings war es auch mit *VisualSwarm* nicht möglich, evolutionär Strukturen zu erzeugen, die Parallelen zu Strukturen in der Natur besitzen. Durch einige Veränderungen an Bonabeaus Fitnessfunktion konnten allerdings Architekturen erzeugt werden, die vom menschlichen Standpunkt aus recht strukturiert erscheinen, wobei dieses natürlich wieder sehr schwer zu formalisieren ist.
Die einzelnen Regeln in einer Regelmenge besitzen sehr starke Abhängigkeiten untereinander, schon der Wegfall einer einzelnen Regel kann eine vollkommene Zerstörung

der ursprünglich erzeugten Struktur mit sich führen (Beweis siehe Abschnitt 3.1.4). Daher ist es schwierig, Strukturen mit Hilfe von evolutionären Algorithmen an die nächste Generation weiterzugeben, da Crossover- und Mutations-Operationen diese Regelzusammenhänge schnell zerstören können. Ein Hilfsmittel, um dieses teilweise zu umgehen, besteht in der Sortierung der einzelnen Gene aufgrund der Tatsache, ob sie zur Erzeugung einer Struktur aktiviert wurden oder nicht (siehe Abschnitt 4.3.4.6).

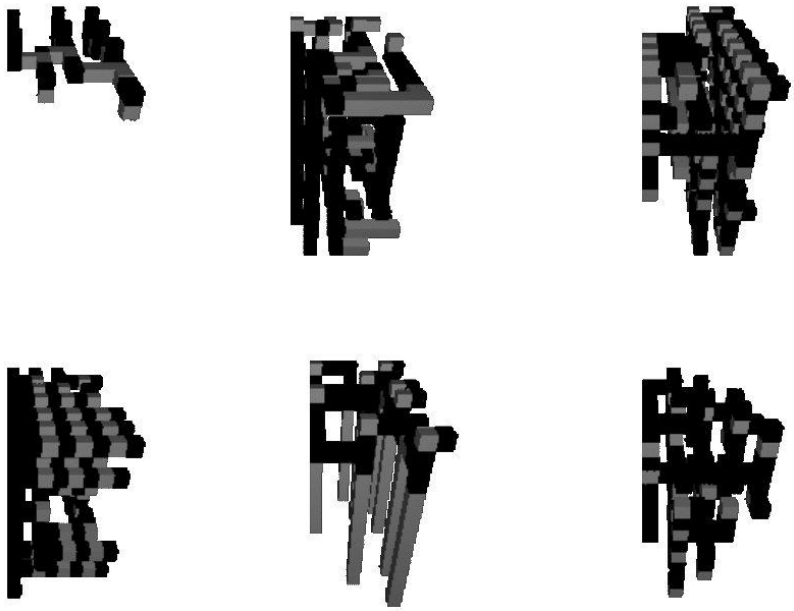

Abbildung 5.4.: Beispielstrukturen, die mit Hilfe von *VisualSwarm* durch evolutionär erzeugte Regelmengen generiert wurden

5.2.3. Zielgerichtete Suche

Bei der zielgerichteten Suche wird nicht eine beliebige *hochbewertete*, sondern eine speziell *vorgegebene* Struktur gesucht. Ziel dieser Berechnungen ist die automatische Generierung von Regelmengen zur Erzeugung gewünschter Strukturen, die durch eine Art Plan beschrieben werden.
Um die gewünschten Strukturen zu erhalten, wurden die gebildeten Muster bezüglich ihrer Ähnlichkeit zur endgültigen Struktur bewertet. Je mehr Blöcke übereinstimmen, desto höher ist die Fitness der Regelmenge, die dieses Muster erzeugt hat (näheres siehe Abschnitt 4.3.4.3).
Beim Test dieser Bewertungsfunktion hat sich schnell herausgestellt, dass komplexe Muster, das heißt Muster, die aus vielen Regeln erzeugt wurden, mit der zielgerichteten Suche nicht gefunden werden. Bei Mustern aus wenigen Regeln und einem sehr kleinen Raum möglicher Regeln (minimal benötigte Regelanzahl * 1.3) konnten aber durchaus Regelmengen gefunden werden, welche die gewünschten Blockplatzierun-

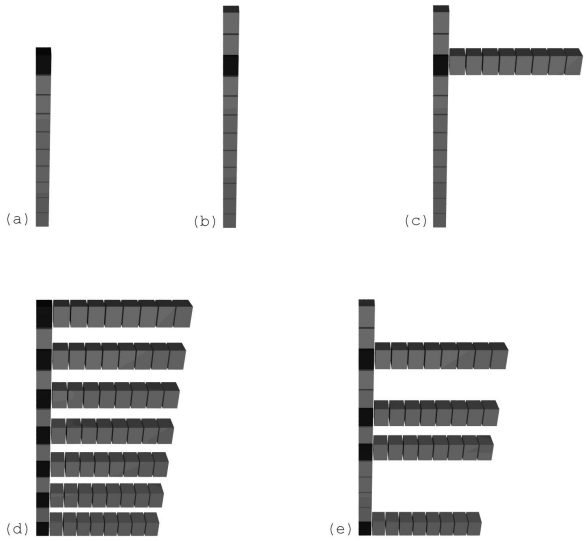

Abbildung 5.5.: Beispielstrukturen, die mit Hilfe von *VisualSwarm* durch zielgerichtete Suche generiert wurden *((a) - (d))*. Die Struktur *(e)* oder ähnliche Abwandlungen traten bei der Regelmengensuche zur Erzeugung von Struktur *(d)* sehr häufig auf, da Regelmengen, die *(d)* erzeugen können, nicht in jedem Simulationslauf identische Resultate erzeugen.

gen durchführen. In Abbildung 5.5 sind Strukturen abgebildet, deren dazugehörige Regelmenge durch eine zielgerichtete Suche gefunden wurde. Die exakte Abdeckung hat sich dabei für die Bewertung als erfolgreicher herausgestellt als die überlappende Abdeckung; letztere produziert sehr große Strukturen, die das gewünschte Muster zwar abdecken, aber keine Ähnlichkeit mit dem Zielgebilde mehr erkennen lassen.
Die zielgerichtete Suche liefert bessere Resultate, wenn die Wahrscheinlichkeit für den Austausch aktivierter Regeln höher gewählt wird ($p = 0.1$) als bei der evolutionären Suche. Dieses resultiert daraus, dass Regeln, die mehr Blöcke produzieren, als im gewünschten Muster vorhanden sind, trotzdem noch die Fitness erhöhen und sich daher als erfolgreiche Regeln durchsetzen können. Zum Austausch dieser nur lokal erfolgreichen Regeln, welche die Findung der Regel verhindern, die ein korrektes Teilmuster erzeugen kann, muss die Wahrscheinlichkeit für eine Ersetzung derselben vergrößert werden.

5.2.4. Simulated Annealing Suche

Die „Simulated Annealing“-Suche wurde mit *VisualSwarm* als ein Spezialfall der evolutionären Suche realisiert, indem vollkommen auf Crossover-Operationen verzichtet wird und die Mutationswahrscheinlichkeit zum Ersetzen genutzter MicroRules mit der Zeit abnimmt. Näheres Angaben hierzu sind im Kapitel 4.3.5 zu finden. Bei den durchgeführten Versuchsreihen wurde die Mutationswahrscheinlichkeit, beginnend bei 0.5, in jeder Generation mit 0.99 multipliziert, sodass sie stetig kleiner

wurde mit dem Grenzwert 0. Auffällig bei den Auswertungen der Fitnessentwicklung (siehe Abbildung 5.6) ist, dass erst bei sehr kleinen Mutationswahrscheinlichkeiten (etwa ab 0.06) eine signifikante Steigerung der durchschnittlichen Fitness einer Population zu erkennen ist. Bei größeren Mutationsraten können keine komplexeren und damit fitteren Strukturen entstehen, da von einer Generation zur nächsten einzelne für den Bauprozess wichtige Regeln zu häufig ausgetauscht werden.
Durch diese Ergebnisse kann Bonabeaus empirisch erzielter Parameter für die Mutationswahrscheinlichkeit, der sich in einer ähnlichen Größenordnung bewegt, belegt werden.

Abbildung 5.6.: Durchschnittliche Fitnessentwicklung der Population bei Durchsuchung des Lösungsraumes mit Hilfe von „Simulated Annealing“-ähnlichen Techniken.

6. Fazit und Ausblick

Stigmergy als indirektes Kommunikationsmittel zwischen sozialen Insekten ist ein sehr einfacher Ansatz, um komplexes Verhalten, das die Fähigkeiten eines einzelnen Individuums bei weitem übersteigt, zu erzeugen. Bonabeau hat in seinen Arbeiten durch manuell erzeugte Regelmengen gezeigt, dass Stigmergy im Prinzip ausreicht, um direkte Kommunikation und Befehlsstrukturen - beispielsweise durch eine übergeordnete, allwissende Instanz - zu ersetzen. Bei Simulationen mit dem entwickelten Softwarepaket VisualSwarm unter Zuhilfenahme der Simulationsumgebung *SWARM* konnten Bonabeaus Schlussfolgerungen in diesem Bereich vollständig bestätigt werden, es gelang die Nachbildung seiner Ergebnisse; selbst durch ein sehr simples und abstraktes Modell sind damit Ergebnisse erzielt worden, die natürlichen Strukturen in Form und Aufteilung signifikant ähneln. Eine Umsetzung dieses Modells in einer kontinuierlichen Umgebung mit einer unbegrenzten Anzahl von Zuständen hat somit durchaus das Potential, natürliches Verhalten nachbilden zu können. Dabei ist es wichtig, herauszustellen, dass Stigmergy nur eine von vielen Möglichkeiten ist, das Verhalten sozialer Spezies zu erklären, allerdings aufgrund von u.a. den Ergebnissen dieser Arbeit einen sehr erfolgreichen Erklärungsversuch darstellt.
Mit Hilfe von einer wie hier aufgebauten Gen-Menge und Kommunikation lediglich durch Stigmergy ist es damit möglich, präzise und wiederholt einzelne Architekturen zu erzeugen, ohne die Architekturen selbst in ihrer Gesamtheit kodieren zu müssen.

Der zweite Aspekt von Bonabeaus Arbeiten, die automatische Generierung von „erfolgreichen“ Regelmengen, muss dagegen etwas differenzierter betrachtet werden. Die von ihm gewählte Suchmethode, das evolutionäre Durchsuchen des Lösungsraumes mit Hilfe von genetischen Algorithmen, scheint ein sinnvoller Ansatz zu sein, da eine vollständige Suche aufgrund der Größe des Raumes entfällt und daher die Notwendigkeit zum Einsatz von Heuristiken besteht. Bonabeau hat größtenteils biologische, evolutionsbezogene Ansätze betrachtet, spezielle Algorithmen aus der Informatik wurden kaum eingesetzt; daher bleibt zu untersuchen, ob andere Suchmethoden, wie zum Beispiel Simulated Annealing, gegebenenfalls bessere Ergebnisse liefern oder schneller zu vergleichbar guten Ergebnissen gelangen können. Erste Simulationen mit einer Technik, die mit Simulated Annealing vergleichbar ist, brachten ähnliche Ergebnisse wie die evolutionäre Suche, zusätzlich konnten aber noch weitere Erkenntnisse, zum Beispiel über den Wertebereich einzelner Simulationsparameter, gewonnen werden.
Die größte Schwierigkeit bei der Bewertung von Zwischenergebnissen, die für die evolutionäre Herangehensweise der Regelmengengenerierung essenziell ist, ist Bonabeaus Formulierung der Fitnessfunktion. Er untersuchte in seinen Arbeiten die

Strukturiertheit der Muster, ohne auf den funktionalen Aspekt einzugehen; dieses ist genau gegensätzlich zu natürlich entstehenden Bauwerken, deren Struktur erst und ausschliesslich durch die Funktion entsteht. Hierin liegt auch der Grund, dass ein Großteil der von Bonabeau erzielten Strukturen erst durch manuelle Eingriffe verbessert werden musste.
Die Ergebnisse, die in Simulationsdurchläufen mit der entwickelten Software *VisualSwarm* erzielt wurden, ähneln stark den Ergebnissen von Bonabeau. Die Unterschiede zwischen den jeweiligen resultierenden Strukturen lassen sich durch den riesigen Lösungsraum erklären, insgesamt sind aber durchaus Ähnlichkeiten zu seinen Resultaten zu entdecken; eine Bewertung, ob die erzeugten Muster eher strukturiert oder weniger strukturiert sind, ist an dieser Stelle nicht möglich, da in Bonabeaus Arbeiten keine exakten Versuchsparameter und keine detaillierte Beschreibung aller Parameter der Fitnessfunktion aufgeführt werden. Durch die Weiterentwicklung und Ergänzung der Bewertungsfunktion (zum Beispiel um den Teil F_4 [Bottlenecks]) konnten theoretische Überlegungen Bonabeaus in *VisualSwarm* berücksichtigt werden, aber aufgrund der Schwächen der Bewertungsfunktion an sich konnten keine signifikanten Veränderungen bzw. Verbesserungen der Ergebnisse festgestellt werden.
Durch die verteilte Konzeption der Software und verbesserte Rechnerleistungen können bereits mit kleineren Netzwerken (< 10 Rechner) mehrere hundert Generationen von Partikelschwärmen innerhalb weniger Tage berechnet werden, wozu Bonabeau noch mehrere Wochen benötigte.

Insgesamt wurden in dieser Diplomarbeit die Arbeiten von Bonabeau und anderen auf dem Gebiet der Strukturfindung durch Stigmergy nachvollzogen und die beschriebenen Konzepte und Erfahrungen in der Software *VisualSwarm* realisiert. Dabei konnten einmal Bonabeaus Ergebnisse reproduziert, aber auch Weiterentwicklungen dieses Themas durch modifizierte Bewertungsfunktionen erreicht werden. Es stellte sich heraus, dass die Bewertungsfunktion Bonabeaus nicht ausreicht, um durch evolutionäre Herangehensweise Regelmengen automatisch zu generieren. Ob durch ein grundlegendes Umschwenken auf eine Bewertung von Bauwerken aufgrund ihrer Funktionalität und nicht ihrer Strukturiertheit die Resultate der indirekten Kommunikation in evolutionär angepassten Partikelschwärmen durch Stigmergy noch weiter verbessert werden können und damit stärker an natürliche Nester heranreichen, bleibt noch zu untersuchen. Andere Arbeiten (siehe [KV97]) haben hier bereits erfolgreiche Ergebnisse erzielen können, allerdings war dort die Funktionalität deutlich klarer und exakter spezifizierbar als bei den hier betrachteten sozialen Spezies.

Die Untersuchungen im Bereich Stigmergy und der evolutionären Entwicklung von Regelmengen durch einfache Partikelschwärme liefern sowohl für Biologen als auch für Ingenieure interessante Ergebnisse. Biologen gewinnen Einblicke in die grundlegenden Prozesse und die minimal benötigten Fähigkeiten von „einfachen“ Lebewesen und können somit deren Verhalten fundierter beurteilen und erklären. Gordon

schrieb hierzu:

> *„If we knew how an ant colony works, we might understand more about how all such systems work, from brains to ecosystems.“* [Gordon, 1999].

Ingenieure können die Ergebnisse beispielsweise bei der Entwicklung von Programmen oder Maschinen benutzen, die Probleme durch kollektive Zusammenarbeit von einfachen Teilelementen / Agenten angehen. In beiden Bereichen werden Wissenschaftler mit demselben „inversen“ Problem konfrontiert: Automatische Generierung eines einfachen Algorithmus, der ein benötigtes Muster oder einen benötigten Zustand erzeugen kann.

A. XML-Steuerdatei

Die Simulationen werden jeweils vollständig parameterisiert mit Hilfe externer Konfigurationsdateien. Als Dateiformat wurde hierbei XML gewählt, da es plattformunabhängig eingesetzt und komfortabel abgefragt werden kann.
Im Folgenden sind die einzelnen XML-Tags aufgelistet, die mit Werten belegt werden können. Alle Eingaben sind optional, wenn keine spezifischen Werte festgesetzt werden, wird auf Default-Werte zurückgegriffen.

1. **Server Parameter**

   result_file_name Der Dateiname, in den die Simulationsergebnisse abgelegt werden. Diese Datei wird während der Simulation genutzt, muss also eine gültige Pfadangabe mit Schreibberechtigung enthalten.
   Werte: gültiger Dateiname mit vollständiger Pfadangabe
   Vorgabe: „result.xml“

   fitness_file_name Eine Datei, die die einzelnen Fitnesswerte jeder einzelnen Population der Gesamtsimulation enthält.
   Werte: gültiger Dateiname mit vollständiger Pfadangabe
   Vorgabe: „fitness“

   min_no_initial_microrules Die minimale Anzahl von MicroRules, die jeder Agent während einer Simulation besitzt.
   Werte: [1...MAX_INT]
   Vorgabe: 40

   max_no_initial_microrules Die maximale Anzahl von MicroRules, die jeder Agent während einer Simulation besitzt.
   Werte: [min_no_initial_microrules..MAX_INT]
   Vorgabe: 60

   diagonal_positioning Blöcke müssen immer an anderen Blöcken anschliessend abgelegt werden. Wird eine diagonale Positionierung zugelassen, reicht es aus, dass sich zwei Blöcke lediglich an einer einzelnen Kante oder einer Ecke berühren. Wird dieser Parameter auf *FALSE* gesetzt, werden nur Regeln erzeugt, die Blöcke positionieren, die mit einer gesamten Fläche an einen bereits platzierten Block anschließen.
   Werte: TRUE / FALSE
   Vorgabe: FALSE

   probability_for_block_in_microrule Ein Parameter, der festlegt, mit welcher Wahrscheinlichkeit eine einzelne Position in einer MicroRule mit einem

Block belegt wird. Ein Wert von 0.2 bedeutet beispielsweise, dass etwa jedes fünfte Feld einen Block besitzt.
Werte: [0...1]
Vorgabe: 0.2

probability_for_used_gene_mutation Die Wahrscheinlichkeit, dass eine während der Simulation genutzte MicroRule durch den Mutations-Operator durch eine neue MicroRule ersetzt wird.
Werte: [0...1]
Vorgabe: 0.01

probability_for_unused_gene_mutation Die Wahrscheinlichkeit, dass eine während der Simulation nicht genutzte MicroRule durch den Mutations-Operator durch eine neue MicroRule ersetzt wird.
Werte: [0...1]
Vorgabe: 0.9

probability_for_crossover Die Wahrscheinlichkeit, dass für ein einzelnes Individuum eine Crossover-Operation durchgeführt wird.
Werte: [0...1]
Vorgabe: 0.2

crossover_points Die Anzahl der Crossover-Punkte, die bei einer Crossover-Operation zur Erzeugung der Nachfahren benutzt werden.
Werte: [1...MAX_INT]
Vorgabe: 1

used_annealing Der Abkühlungsfaktor für die Wahrscheinlichkeit, die den Austausch (Mutation) genutzter MicroRules steuert. Ist der hier angegebene Wert ≥ 1, wird der Wahrscheinlichkeitswert zwischen den Generationen nicht angepasst (siehe auch Abschnitt 4.3.5).
Werte: [0...1]
Vorgabe: 1

unused_annealing Der Abkühlungsfaktor für die Wahrscheinlichkeit, die den Austausch (Mutation) ungenutzter MicroRules steuert. Ist der hier angegebene Wert ≥ 1, wird der Wahrscheinlichkeitswert zwischen den Generationen nicht angepasst (siehe auch Abschnitt 4.3.5).
Werte: [0...1]
Vorgabe: 1

order_genes Dieser Parameter definiert, ob die MicroRules-Menge eines Partikelschwarms nach Aktivierung sortiert werden soll, bevor Crossover-Operationen durchgeführt werden. Wenn dieser Parameter auf *TRUE* gesetzt wird, werden die einzelnen MicroRules des ersten an der Crossover-Operation beteiligten Individuums so sortiert, dass die aktivierten MicroRules am Anfang eingeordnet werden, die aktivierten Regeln des zweiten Individuums werden am Ende angeordnet. Dadurch steigt die Wahrscheinlichkeit, dass Regeln, die aufeinander aufbauen, nicht durch die

Crossover-Operation getrennt werden.
Werte: TRUE / FALSE
Vorgabe: FALSE

timeout_seconds Der Timeout-Wert hilft, abgestürzte Clients zu identifizieren. Die hier angegebene Zeit (in Sekunden) wartet der Server maximal auf das Ergebnis einer Simulation, danach werden die Simulationsparameter an einen anderen Client geschickt. Sollte vom ersten Client das Ergebnis dann doch noch kommen, wird es verworfen.
Werte: [0...MAX_INT]
Vorgabe: 1200

number_of_generations Die Anzahl von Generationen, die der Server berechnen soll. Da jede Generation gesichert wird und diese gesicherten Zwischenstände als Startpunkt einer neuen Simulation benutzt werden können, kann dieser Wert auch noch nachträglich nach oben korrigiert werden.
Werte: [1...MAX_INT]
Vorgabe: 100

size_of_population Mit diesem Parameter wird festgelegt, wieviele Srukturen berechnet bzw. wieviele Schwärme simuliert werden sollen im Laufe einer einzelnen Generation. Die Populationsgröße bleibt in jeder Generation konstant.
Werte: [1...MAX_INT]
Vorgabe: 60

visualization_script Dieser Parameter definiert den Namen eines Skriptes, welches POVRay-Beschreibungsdateien visualisieren kann. Dieses könnte beispielsweise folgendermaßen aussehen (abhängig von installierter Software):

```
#!/bin/tcsh
/usr/bin/povray +I $1.pov +o $1.png; /usr/X11R6/bin/xv $1.png
```

Werte: vollständiger Dateiname
Vorgabe: „./display_structure“

2. **Client Parameter**

initial_brick_position_n Diese drei Parameter (n = x, y, z) legen fest, an welcher Position der initiale Block in der Simulationsumgebung abgelegt wird.
Werte: [0..world_size_n]
Vorgabe: (7, 7, 3)

agentcolor Die Farbe des visualisierten Agenten wird durch diesen Wert festgelegt.

Werte: [2...255]
Vorgabe: 64

brickcolor Die Farbe eines abgelegten Blockes kann mit diesem Konfigurationswert festgelegt werden.
Werte: [2...255]
Vorgabe: 65

individuals_file_name Die Daten zu jedem Individuum in der Population werden von jedem Client an dieser Stelle angelegt. Daher muss diese Pfadangabe entweder ein global adressierbares Netzwerkverzeichnis sein oder auf jedem einzelnen Client-Rechner existieren.
Werte: gültiger Dateiname mit vollständiger Pfadangabe
Vorgabe: „individuals“

output_file_steps Durch diesen Wert wird definiert, nach wievielen Simulationsschritten ein Zwischenergebnis in einer POVRay-Beschreibungsdatei abgelegt werden soll. Umfasst die Simulation beispielsweise 30.000 Simulationsschritte und dieser Wert wurde auf 5.000 gesetzt, wird die erste Datei nach dem Simulationsschritt 5.000 erzeugt, die zweite nach Schritt 10.000 und so weiter. Die einzelnen Dateien werden durch einen zusätzlichen Suffix fortlaufend durchnummeriert.
Werte: [1...MAX_INT]
Vorgabe: 100

use_display Um den Fortschritt einzelner Simulationen visuell mitverfolgen zu können, kann dieser Wert auf *true* gesetzt werden. Dann erscheint für jeden Level ein Fenster, in dem die platzierten Blöcke direkt angezeigt werden.
Werte: TRUE / FALSE
Vorgabe: FALSE

show_agents Dieser Parameter kann nur in Kombination mit dem *use_display* benutzt werden.
Werte: TRUE / FALSE
Vorgabe: FALSE

min_size_of_structure Zur Bestimmung der Strukturiertheit eines Musters können wiederholte Teilstrukturen gesucht werden. Die minimale Grösse, die eine solche Struktur besitzen muss, kann hier festgelegt werden.
Werte: [1...MAX_INT]
Vorgabe: 1

number_of_empty_away_moves Um Agenten an einer Struktur festzuhalten und damit den Bauprozess zu beschleunigen, da ein Abwandern in den leeren Simulationsraum nach Vorgabe keine Regel aktivieren kann, soll durch diesen Parameter festgelegt werden, wie oft ein Agent von seiner aktuellen Position aus ein benachbartes Feld wieder verlassen darf, bis er ein Feld erreicht, das wenigstens einen Block in der Nachbarschaft besitzt,

wo also eine Möglichkeit besteht, eine Regel anzuwenden.
Werte: [0...MAX_INT]
Vorgabe: 2

factor_fn Die Faktoren f_i für die einzelnen Teile der Fitnessfunktion

$$Fitness\ F = \prod_{i=1}^{4}(f_i * F_i^{p_i})$$

Werte für n: [1...4]
Werte: [- MAX_DOUBLE...MAX_DOUBLE]
Vorgabe: 1

exponent_fn Die Exponenten p_i für die einzelnen Teile der Fitnessfunktion

$$Fitness\ F = \prod_{i=1}^{4}(f_i * F_i^{p_i})$$

Werte für n: [1...4]
Werte: [- MAX_DOUBLE...MAX_DOUBLE]
Vorgabe: 1

server_name Der DNS Name des Servers.
Werte: Server-Name
Vorgabe: „localhost“

client_name Der Name des Clients. Dieser Wert ist nur zu Debug-Zwecken gedacht und kann beliebig gewählt werden.
Werte: Client-Name
Vorgabe: „Standard Client“

3. **Server und Client Parameter**

world_size_n Die Grösse des Simulationsraumes.
Werte: [1...MAX_INT]
Vorgabe: (16, 16, 16)

cells_per_level Dieser Wert legt fest, wie viele Felder in der Ebene, in der sich der Agent befindet, betrachtet werden, inklusiv des Feldes, in dem sich der Agent selbst befindet.
Werte: [9...MAX_INT]
Vorgabe: 9

number_of_levels Dieser Wert definiert, wie viele Ebenen betrachtet werden.
Werte: [3...MAX_INT]
Vorgabe: 3

number_of_agents Durch diesen Parameter wird definiert, wieviele Agenten gleichzeitig in der Simulation vorkommen.
Werte: [1...MAX_INT]
Vorgabe: 1

output_file_prefix Der Name der erzeugten POVRay-Beschreibungsdatei mit vollständigem Pfad. Dieser Name wird ergänzt durch Anhängen der Generation und der Simulationsnummer und erhält die Dateiendung „.pov“.
Werte: gültiger Dateiname mit vollständiger Pfadangabe
Vorgabe: „output_“

simulation_steps Die Anzahl von Schritten, die jeder einzelne Agent durchführen muss, bevor die Simulation beendet wird. Nur wenn die Anzahl maximaler Blöcke (*max_bricks_steps*) überschritten wird, wird die Simulation vor Erreichen dieses Wertes abgebrochen.
Werte: [1...MAX_INT]
Vorgabe: 1000

max_bricks_number Durch diesen Wert wird festgelegt, ab wann eine erzeugte Struktur zu groß geworden ist und daher nicht weiter berechnet wird.
Werte: [2...MAX_INT]
Vorgabe: 500

bottleneck_runs Dieser Parameter legt fest, wieviele Durchläufe auf einem Client mit derselben Regelmenge durchgeführt werden sollen. Im Anschluss an alle Durchläufe werden die einzelnen Resultate miteinander verglichen.
Werte: [0...MAX_INT]
Vorgabe: 0

port_number Die Portnummer, auf der Client und Server miteinander kommunizieren.
Werte: [1.000...32.000]
Vorgabe: 8990

4. **Vordefinierte Regeln**
Dieser Parametereintrag beschreibt die Regeln (MicroRules), die bei der Simulation als Vorgabe-Regeln verwendet werden. Alle hier definierten Regeln sind in jedem Individuum vorhanden und werden gegebenenfalls durch zufällige, weitere Regeln ergänzt, wenn die maximale Anzahl von MicroRules entsprechend groß gewählt wurde. Eingeleitet wird die Regelmenge durch das $< rules >$ - Tag. Als Untereinträge sind eine beliebige Menge von MicroRule-Tags erlaubt

microrule Eine MicroRule ist eine Regel, die dann aktiviert wird, wenn die Umgebung des Agenten mit der in der Regel definierten Umgebung übereinstimmt.

environment Die Umgebung beschreibt die Existenz von Blöcken in den Nachbarzellen des Agenten. Wieviele Nachbarzellen betrachtet werden, ist dabei von den oben gemachten Parametern abhängig.
Eine Umgebung der Form

```
0#0#0#
0#0#0#    // Ebene i+1
0#0#0#

0#0#0#
1#0#2#    // Ebene i, Agent befindet sich in der Mitte
0#0#0#

0#0#0#
0#0#0#    // Ebene i-1
0#0#0#
```

kann dabei folgendermaßen interpretiert werden:
In den Ebenen über und unter dem Agenten darf kein Block positioniert sein, links vom Agenten muss ein Block der Art 1 liegen, rechts ein Block der Art 2.

reaction Dieser Parameter beschreibt, welche Art von Block an die Stelle, an der sich der Agent befindet, gelegt werden soll.

Beispiel Zwei Regeln, die in jedem Individuum vorhanden sind:
$< rules >$
$< microrule >$
$< environment >$
0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#1#0#0#0#0#0#0#0#0#
$< /environment >$
$< reaction > 2 < /reaction >$
$< /microrule >$
$< microrule >$
$< environment >$
0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#1#0#0#
$< /environment >$
$< reaction > 2 < /reaction >$
$< /microrule >$
$< /rules >$

B. Konfigurationsdatei-Beispiel

```
<ruleset>
<parameters>
<cells_per_level>9</cells_per_level>
<number_of_levels>3</number_of_levels>
<number_of_agents>50</number_of_agents>
<agentcolor>64</agentcolor>
<brickcolor>65</brickcolor>
<initial_brick_position_x>8</initial_brick_position_x>
<initial_brick_position_y>8</initial_brick_position_y>
<initial_brick_position_z>2</initial_brick_position_z>

<world_size_x>16</world_size_x>
<world_size_y>16</world_size_y>
<world_size_z>16</world_size_z>
<output_file_prefix>Output_</output_file_prefix>
<result_file_name>Result</result_file_name>
<output_file_steps>5000</output_file_steps>
<simulation_steps>10000</simulation_steps>
<use_display>true</use_display>
<show_agents>false</show_agents>
</parameters>

<rules>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  0#0#0# 0#0#0# 0#0#0#  1#0#0# 0#0#0# 0#0#0#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  0#0#0# 0#0#0# 0#0#0#  0#0#0# 0#0#0# 1#0#0#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
```

```
0#0#0# 0#0#0# 0#0#0#  0#0#0# 0#0#0# 0#0#0#  0#0#0# 0#0#0# 0#0#1#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  0#0#0# 0#0#0# 0#0#0#  0#0#1# 0#0#0# 0#0#0#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  0#2#0# 0#0#0# 0#2#0#  0#0#0# 1#0#0# 0#0#0#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  0#0#0# 2#0#2# 0#0#2#  0#0#0# 0#0#0# 0#1#0#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  0#2#2# 0#0#0# 0#2#2#  0#0#0# 0#0#1# 0#0#0#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  0#0#2# 2#0#2# 0#0#0#  0#1#0# 0#0#0# 0#0#0#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  2#2#2# 2#0#2# 2#2#2#  0#0#0# 0#1#0# 0#0#0#
</environment>
<reaction>2</reaction>
</microrule>
<microrule>
<environment>
0#0#0# 0#0#0# 0#0#0#  0#0#0# 0#0#0# 0#0#0#  2#2#2# 2#2#2# 2#2#2#
</environment>
```

```
<reaction>1</reaction>
</microrule>
</rules>
</ruleset>
```

C. POV-Ray Beschreibungsdateien

In diesem Abschnitt wird eine beispielhafte Szenenbeschreibungsdatei für POVRay vollständig aufgelistet. Als Kommentar werden in jede während der Simulation generierte Datei Hinweiszeilen eingefügt, die einige wichtige Simulationsparameter enthalten. Im zweiten Teil ist eine Sequenzbeschreibung abgebildet, die es ermöglicht, mit Hilfe von POVRay eine Animation zu erstellen, die die Struktur aus unterschiedlichen Perspektiven abbildet.

C.1. Szenenbeschreibung

```
// standard include files

#include "colors.inc"
#include "shapes.inc"
#include "finish.inc"
#include "glass.inc"
#include "metals.inc"
#include "stones.inc"
#include "woods.inc"


// ====================================================
// Simulation Information:
//
//    Number of Agents: 50
//    Number of simulations per generation: 4
//    Number of simulation steps: 50000
//
// Probabilities:
//
//    Block in microrule: 0.11
//    Crossover: 0.2 (1-point crossover)
//    Unused gene mutation: 0.9
//    Used gene mutation: 0.01
// ====================================================

#declare anim = clock;
```

```
background
{
color White
}

camera
{
location <8+anim, -16+anim, -6+anim>
look_at <8, 8, 8>
}

light_source
{
<8, -16, -8> color White
}

box
{
<8, 8, 1>,
<8.96, 8.96, 1.96>
texture
{
pigment {color LightGray}
}
}
box
{
<8, 8, 2>,
<8.96, 8.96, 2.96>
texture
{
pigment {color Blue}
}
}
box
{
<9, 8, 2>,
<9.96, 8.96, 2.96>
texture
{
pigment {color Blue}
}
}
```

C.2. Sequenzbeschreibung

```
Antialias=off
Antialias_Threshold=0.1
Antialias_Depth=2
Initial_Frame=1
Final_Frame=30
Initial_Clock=-5
Final_Clock=5
Cyclic_Animation=on
Pause_when_done=off
Input_File_Name="output_gen_0076_sim_0040.pov"
```

D. Kommandozeilen-Parameter

Die folgenden Parameter können beim Start von Server- bzw. Client-Instanzen angegeben werden; die einzelnen Argumente sind beliebig kombinierbar.

-configuration, -c: Der Name einer XML Konfigurationsdatei, die Simulationsparameter enthält. Wird keine Konfigurationsdatei angegeben, werden die Vorgabewerte für die einzelnen Parameter benutzt.
Beispiel: Server -configuration config.xml

-debug, -d: Steuerung der Konsolenausgaben des Programmes. Hierbei sind Werte von 0 bis 9 gültig, wobei 0 keinerlei Ausgaben produziert (nicht einmal Fehlermeldungen) und 9 annähernd jeden Programmschritt dokumentiert. Der Vorgabewert, der benutzt wird, wenn dieses Argument fehlt, ist Level 3.
Beispiel: Server -debug 7

-help, -h: Auflistung aller möglicher Parameter und Abbruch des Programmes
Beispiel: Server -help

-resume, -r: Der Name einer Result-Datei, deren Ergebnisse als Grundlage für weitere Berechnungen genommen werden. Es ist möglich, durch Angabe einer neuen Konfigurationsdatei die Simulation mit früheren Ergebnissen und neuen Parametern weiterzuführen. Dieser Parameter ist nur bei Server-Instanzen gültig.
Beispiel: Server -resume results/result.xml

-verbose, -v: dasselbe wie -debug 5

Literaturverzeichnis

[AC01] Agrafiotis, Dimitris K. ; Cedeño, Walter. *Feature Selection for Structure-Activity Correlation Using Binary Particle Swarms.* J. Med. Chem. 2002, 45, 1098-1107. October 2001

[All94] Allman, William F. *Mammutjäger in der Metro - Wie das Erbe der Evolution unser Denken und Verhalten prägt.* Spektrum Akademischer Verlag. 1994

[Apa02] Apache Software Foundation. *The Apache XML Project, Xerces API.* http://www.apache.org. 2002

[BDT99] Bonabeau, Eric ; Dorigo, Marco ; Theraulaz, Guy. *Swarm Intelligence - From natural to artificial Systems.* Oxford University Press. 1999

[BGS+00] Bonabeau, Eric ; Guérin, Sylvain ; Snyers, Dominique ; Kuntz, Pascale ; Theraulaz, Guy. *Three-dimensional architectures grown by simple 'stigmergic' agents.* http://www.elsevier.com/locate/biosystems. January 2000

[BM01] Bonabeau, Eric ; Meyer, Christopher. *Swarm Intelligence: A Whole New Way to Think About Business.* Harvard Business Review. 2001

[Brü00] Brückner, Sven. *Return from the Ant - Synthetic Ecosystems for Manufacturing Control.* 2000

[BTC] Bonabeau, Eric ; Theraulaz, Guy ; Cogne, Francois. *The Design of complex Architectures by simple Agents*

[BTD+] Bonabeau, Eric ; Theraulaz, Guy ; Deneubourg, Jean-Louis ; Aron, Serge ; Camazine, Scott. *Self-organization in social insects*

[BTD+98] Bonabeau, Eric ; Theraulaz, Guy ; Deneubourg, Jean-Louis ; Franks, Nigel R. ; Rafelsberger, Oliver ; Joly, Jean-Louis ; Blanco, Stéphane. *A model for the emergence of pillars, walls and royal chambers in termite nests.* The Royal Society. 1998

[C+99] Carson, Chirstopher [u. a.]. *POV-Ray User's Documentation 3.0.10.* http://www.povray.org. 1996-1999

[Dec00] Decker, Ethan H. *Biology 576: Landscape Ecology & Macroscopic Dynamics: Self-Organizing Systems*. 2000

[Fer01] Ferber, Jacques. *Multiagentensysteme, eine Einführung in die verteilte künstliche Intelligenz*. Addison-Wesley. 2001

[G+02] Gilbert, David [u. a.]. *The JFreeChart Class Library*. http://www.object-refinery.com/jfreechart/index.html. October 2002

[Gol89] Goldberg, David E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley. 1989

[Gra59] Grassé, P.-P. *La reconstruction du nid et les coordinations interindividuelles chez Bellicositerms natalensis et Cubitermes sp. La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs*. Insectes Sociaux. 1959

[HM] Holland, Owen ; Melhuish, Chris. *Stigmergy, self-organisation, and sorting in collective robotics*. Artificial Life

[Hoa] Hoar, Ricardo. *Swarm Development Tools*

[Jaca] Jacob, Christian. *Swarm Intelligence Systems*. Department of Computer Science University of Calgary

[Jacb] Jacob, Christian. *Toward a Better Understanding of Complexity*. Department of Computer Science University of Calgary

[JL00] Johnson, Paul ; Lancaster, Alex. *Swarm User Guide*. www.swarm.org. April 2000

[KE01] Kennedy, James ; Eberhart, Russel C. *Swarm Intelligence*. MK - Morgan Kaufmann. 2001

[Ken97] Kennedy, J. *The particle swarm: social adaptation of knowledge*. IEEE International Conference on Evolutionary Computation. 1997

[KV97] Krink, Thiemo ; Vollrath, Fritz. *Analysing spider web-building behaviour with rule-based simulations, and genetic algorithms*. 1997

[Lan90] Langton, Chris G. *Computation at the edge of chaos: Phase transitions and emergent computation*. Physica D 32:12-37. 1990

[LMB95] Langton, Chris G. ; Minar, Nelson ; Burkhart, Roger. *The Swarm Simulation System: A tool for studying complex systems (Draft version)*. http://www.santafe.edu/projects/swarm/. April 1995

[LS00] Lettmann, Theodor ; Schulz, André. *Intelligente Agenten*. 2000

[Mas] Mason, Zachary. *Programming with Stigmergy: Using Swarms for Construction*

[MBLA96] Minar, Nelson ; Burkhart, Roger ; Langton, Chris G. ; Askenazi, Manor. *The Swarm Simulation System: A Toolkit for building Multi-Agent Simulations.* http://www.santafe.edu/projects/swarm/. June 1996

[MGH01] Martinoli, Alcherio ; Goodman, Rodney ; Holland, Owen. *Swarm Intelligence and Self-Organization.* California Institute of Technology, Lecture Sheets. Winter 2001

[Mit] Mitchell, Melanie. *Life and Evolution in Computers*

[Sta00] Staelin, Charles P. *jSIMPLEBUG - a Swarm tutorial for Java.* www.swarm.org. April 2000

[Swa94] Swarm Team. *An overview of the Swarm simulation system.* http://www.santafe.edu/projects/swarm/. August 1994

[Swa00] Swarm Development Group. *Brief Overview of Swarm.* www.swarm.org. March 2000

[TB95a] Theraulaz, Guy ; Bonabeau, Eric. *Coordination in distributed building.* Science 269, 686-688. 1995

[TB95b] Theraulaz, Guy ; Bonabeau, Eric. *Modelling the collective building of complex architectures in social insects with lattice swarms.* Jounal of Theoretical Biology 177, 381. 1995

[Ter98] Terna, Pietro. *Simulation Tools for Social Scientists: Building Agent based Models with SWARM.* http://www.soc.surrey.ac.uk/JASSS/1/2/4.html. March 1998