Ein agentenbasierter evolutionärer Adaptions- und Optimierungsansatz für verteilte Systeme

Der Technischen Fakultät der
Universität Erlangen–Nürnberg

zur Erlangung des Grades

D O K T O R – I N G E N I E U R

vorgelegt von

Stephan Otto

Erlangen 2009

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	20.04.2009
Tag der Promotion:	18.09.2009
Dekan:	Prof. Dr.-Ing. habil. Johannes Huber
Berichterstatter:	Prof. Dr.-Ing. Gabriella Kókai
	Prof. Dr.-Ing. Günther Görz

I think the next century
will be the century of complexity.
Stephen Hawking

It's not the strongest
of the species that survives,
not the most intelligent,
but the one, most responsive to change.
Charles Darwin

Für Larysa und Delia

Kurzfassung

Verteilte Systeme sind in den letzten Jahrzehnten durch den Erfolg des World Wide Web allgegenwärtig geworden. Das Internet, Peer-to-Peer Netzwerke, Grid Computing und mobile Endgeräte bilden zusammen eine verteilte, heterogene, dynamische und komplexe Infrastruktur auf deren Basis verteilte Informationssysteme globale Dienste zur Verfügung stellen. Die ständig steigende Komplexität lässt zentralisierte Top-Down Ansätze zur Steuerung zunehmend an ihre Grenzen stoßen. Die Verteilung von Ressourcen und Informationen, Heterogenität und Dynamik sind Belege für die Ähnlichkeit technischer Systeme mit natürlichen komplexen adaptiven Systemen (CAS), wie z.B. Zellen, Lebewesen und Ökosystemen. Diese weisen ein hohes Maß an Selbstadaptivität und -optimierung auf. In den letzten Jahren hat daher das Interesse an CAS und deren Untersuchung sowie die Übertragung ihrer Mechanismen zur Selbstorganisation auf technische Systeme stark zugenommen. Der Beitrag dieser Arbeit besteht in der Untersuchung eines Ansatzes zur Bereitstellung adaptiver und selbst optimierender Systemfunktionalität.

Hierzu wurden evolutionäre Agenten als ein CAS Mechanismus aus der Kombination von evolutionären, ökonomischen und agentenbasierten Ansätzen eingeführt. Die jeweiligen Vorteile der Ansätze wurden vereinigt. Hierzu zählen einerseits implizite Parallelität, Robustheit und Optimierung in komplexen Suchräumen und andererseits Flexibilität, Autonomie und Interaktionsfähigkeit von Agenten. Evolutionäre Agenten sind als elementare Bestandteile von Informationssystemen in zweierlei Hinsicht konzipiert. Sowohl die eigentliche Systemfunktionalität als auch das verteilte Systemmanagement im Hinblick auf Adaptivität und Optimierung wird gleichzeitig durchgeführt. Damit stellen Agenten kollaborativ die Systemdienste bereit und ein dezentraler evolutionärer Prozess etabliert parallel die systemweite Optimierung der Dienste. Die ökonomische Dimension erlaubt dezentrale marktbasierte Koordinationsmechanismen zwischen den Agenten. Die Modellierung erfolgt Bottom-Up aus Agentensicht und unterstützt damit das Verständnis und die Formalisierung verteilter Probleme. Auf Basis des formalen Optimierungsansatzes wurden wichtige adaptive Eigenschaften abgeleitet. Hierzu gehört die Anpassung des Gesamtsystems sowohl in seiner Dimension als auch in der Funktionalität. Weiterhin wurde durch die Einführung der Erweiterten Takeover Time (ETT) gezeigt, dass dynamische Netzwerkstrukturen, wie das Internet, Peer-to-Peer Systeme und Grid Computing für den Einsatz evolutionärer Agenten geeignet sind. Die Einführung eines Dezentralitätsmaßes erlaubt die Einordnung und den Vergleich mit bestehenden parallelen evolutionären Verfahren.

Mit Hilfe von Prototypen wurden die abgeleiteten Eigenschaften an zwei Problemstellungen in unterschiedlichen Domänen validiert. Dies unterstreicht die Vorteile und Tragweite der Kombination evolutionärer, ökonomischer und agentenbasierter Verfahren. Ebenso ist die Umsetzung der Referenzarchitektur als Erweiterung bestehender Agentenframeworks gedacht und kann integriert werden. Auch dies wurde an drei unterschiedlichen Systemen gezeigt.

Abstract

Distributed systems have become ubiquitous in recent decades due to the success of the World Wide Web. The Internet, peer-to-peer networks, grid computing and mobile devices together form a distributed, heterogeneous, dynamic and complex infrastructure as a base of distributed information systems to provide global services. Traditional top-down approaches have shown to be insufficient in controlling such complexity. The distribution of heterogeneous resources, information and dynamics are evidence of the similarity of modern distributed systems with complex adaptive systems (CAS), such as cells, organisms and ecosystems. These have a high degree of self-adaptation and self-optimization. In recent years the interest in CAS and the transfer of their self-organization mechanisms in technical systems increased. The contribution of this work is the investigation of an approach providing distributed self-adaptive and self-optimizing system functionality.

Evolutionary Agents have been introduced as a combination of evolutionary, economic and agent-based approaches. The particular advantages of the approaches were combined. This includes implicit parallelism, robustness and optimization in complex search spaces and flexibility, autonomy and social skills of agents. Evolutionary agents are conceived as elementary components of information systems in two dimensions. Both the system functionality, as well as the distributed system management in terms of adaptation and optimization is performed simultaneously. Agents provide system services in a collaborative manner. In parallel, a decentralized evolutionary process establishes the system-wide optimization of services provided by agents. The economic dimension allows decentralized market-based coordination mechanisms between the agents. The modeling is bottom-up perspective of agents and thus supports the understanding and formalization of distributed implicit problems. On the basis of the formal optimization approach important adaptive characteristics were derived. To this end, the adjustment is part of the overall system, both in its dimension as well as in the functionality. Furthermore, the introduction of the Enhanced Takeover Time (ETT) has shown that dynamic network structures, like the Intenet, peer-to-peer systems and grid computing are suitable for the application of evolutionary agents. The introduction of a measurement of distribution allows the classification and comparison with existing parallel evolutionary processes.

By means of prototypes derived properties have been validated on two problems in different domains. This highlights the advantages and scope of the combination of evolutionary, economic and agent-based methods. Similarly, the implementation of the reference architecture is designed as an extension of existing agent frameworks and can easily be integrated. This has also been shown in three different systems.

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Wirtschaftsinformatik II (Prof. Kirn) der Universität Hohenheim und am Lehrstuhl für Informatik 2 (Prof. Philippsen) der Universität Erlangen-Nürnberg. Zu besonderem Dank bin ich Prof. Gabriella Kókai verpflichtet, die es mir ermöglichte, diese Dissertation anzufertigen. Sie hat mich bereits während des Studiums für evolutionäre Verfahren begeistert und während der Promotion unterstützt und immer wieder motiviert. Nicht zuletzt hat Sie durch ihre Anmerkungen und Reviews erheblich zur Qualität der vorgelegten Dissertation beigetragen und das Erstgutachten erstellt.

Weiterhin gilt mein Dank Prof. Görz für die Erstellung des Zweitgutachtens. Zu Dank verpflichtet bin ich auch Prof. Kirn für die Diskussionen auf den Gebieten Multiagentensysteme und wissenschaftlichen Arbeiten, die zum Inhalt und zur Strukturierung beigetragen haben. Ebenso möchte ich der Firma Methodpark Software AG (Prof. Hindel) für den gewährten Freiraum und die großzügige Unterstützung während der Anfertigung dieser Dissertation danken. Prof. Philippsen und seinen Mitarbeitern danke ich für die kollegiale Aufnahme am Lehrstuhl für Informatik 2.

Folgende Personen haben durch Diskussionen, gemeinsame Arbeiten und Reviews zum Gelingen dieser Arbeit beigetragen: Christian Anhalt, Thomas Bieser, Dr. Andreas Dietrich, Christian Loos, Xiaoxi Luo, Dirk Mahne, Prof. Ingo Timm und Serge Tsafak.

Vielen Dank auch an meine Eltern, meinen Bruder und meine Freunde, die mich immer wieder motiviert und für den nötigen Ausgleich gesorgt haben. Ein besonderer Dank gilt Larysa, die mich die gesamte Zeit 'ertragen' hat und mir den Rücken freihielt.

Stephan Otto
Erlangen 2009

Inhaltsverzeichnis

Kapitel 1

Einleitung und Problemstellung

1.1 Einleitung

Die Vision des *Ubiquitous Computing*[1] ist seit ihrer Beschreibung Anfang der 90er Jahre faktisch wahr geworden. Das Internet, Peer-to-Peer Netzwerke, Grid Computing und mobile Geräte bilden zusammen eine globale und dynamische Infrastruktur. Prozessoren mit der Fähigkeit zur Kommunikation finden sich in einer Vielzahl von technischen Geräten, die gemeinsam in Netzwerken interagieren. Durch die allgegenwärtige Vernetzung sind Millionen von Rechenressourcen zusammengeschaltet und bilden die Grundlage vom Computergrid des CERN[2] bis hin zu Peer-to-Peer Kommunikationssystemen.

Durch die Interaktionen vieler Komponenten ist diesen Systemen die Veränderung in Raum und Zeit und damit einhergehende Dynamik gemeinsam. Neue Anforderungen entstehen, wie der Umgang mit Topologieveränderungen, extreme Lastschwankungen, Mobilität, Ausfälle von Knoten und Teilsystemen und Interoperabilität von unzuverlässiger und fehlerhafter Software. Dienste kapseln Funktionalität von Ressourcen, Organisationen und Teilsystemen und bilden Zugriffspunkte darauf ab. Durch zeitlich begrenzten virtuellen Zusammenschluss von Diensten und Organisationen entstehen *virtuelle Organisationen* (VO [107]). Die Kollaboration virtueller Organisationen erschwert den Austausch benötigter Informationen und erfordert gleichzeitig standardisierte Kommunikations- und Interaktionsmuster. Kooperationen können für die Dauer einer einzigen Transaktion bestehen oder über die gesamte Lebensspanne von Diensten hinweg. Gleichzeitig stehen Dienste, Organisationen und virtuelle Organisationen im Wettbewerb um Speicherplatz, Bandbreite und Rechenleistung. Diese Dualität kooperativer Konkurrenz wird auch als *Coopetition* [31] bezeichnet.

Verteilung, Heterogenität und Dynamik führen zu enormen Zuwachs an Komplexität auf allen Ebenen verteilter Informations- und Kommunikationssysteme (kurz: *Informationssysteme*[3]). Traditionelle zentralisierte Top-Down Managementansätze stoßen zunehmend an ihre Grenzen. Immer mehr Parallelen finden sich zwischen modernen Informationssystemen und sogenannten *Komplexen Adaptiven Systemen* (*CAS*) zum Beispiel in der Physik, Biologie, Ökonomie und Soziologie. In den letzten Jahrzehnten hat daher das Interesse an Softcomputing und Organic Computing mit biologischem Vorbild und deren Übertragung auf technische

[1] The Computer for the 21st Century, Mark Weiser [246]

[2] Europäische Organisation für Nuklearforschung, siehe `http://www.cern.ch/`

[3] Im Rahmen dieser Arbeit wird auch einfach der Begriff System verwendet, sofern kein anderer Kontext benannt wurde oder ersichtlich ist.

Systeme stark zugenommen. *Self-x Eigenschaften*[4], Robustheit, Adaptivität und Autonomie sind Merkmale natürlicher komplexer adaptiver Systeme und zunehmend auch moderner Informationssysteme. Eine große Herausforderung stellen der Entwurf und die Entwicklung von Informationssystemen nach den CAS Prinzipien dar. Eine zentrale Frage umfasst daher die Gestaltung lokaler Verhaltensmuster, um global das gewünschte und möglichst optimale Systemverhalten innerhalb vorgegebener Grenzen zu gewährleisten.

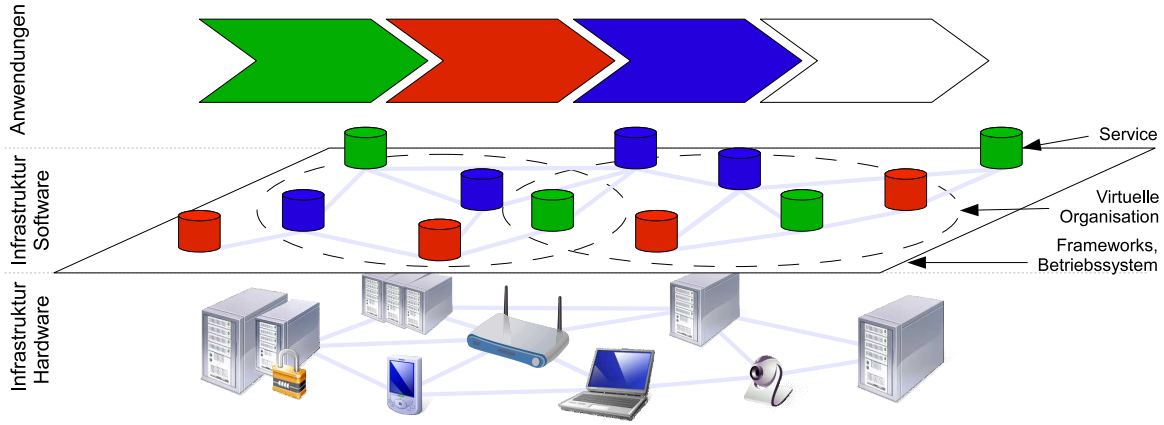


Abbildung 1.1: Komplexe Informationssysteme zur Bearbeitung von Aufgaben mittels kollaborativen verteilten Diensten in virtuellen Organisationen auf verteilter und vernetzter Infrastruktur

Durch die globale Vernetzung von Ressourcen, Diensten, Organisationen und deren Informationssystemen, umfasst deren Management weit mehr als nur interne Anpassung und Optimierung von Prozessen. Vielmehr liegt der Fokus auf schneller und flexibler interorganisationaler Prozess- und Systemoptimierung, um in dynamischen und globalisierten Märkten zu bestehen. Abbildung 1.1 zeigt Dienste, die sich über Rechner und Organisationsgrenzen vernetzen, um autonom unternehmensübergreifende Prozesse auszuführen. Hierzu zählt neben der Betrachtung interner Abläufe auch das Berücksichtigen der Marktsituation, um ressourcenschonend, kostengünstig und effektiv auf Nachfrageveränderungen zu reagieren. Es ist daher ein zentraler Beitrag unternehmensübergreifender Informationssysteme, Wettbewerbsfähigkeit selbstadaptiv und selbstoptimierend zu gewährleisten. In diesem Kontext ist das Erreichen eines optimalen Systemzustandes im Vergleich zur kontinuierlichen Anpassung weit weniger relevant. Ein wichtiger Faktor hierbei ist die Dynamik der Systeme selbst und ihrer Umwelt. Während der zentralen Erstellung optimaler Lösungen haben sich System und Umwelt bereits verändert. Informationssysteme können damit selbst als dynamisches Problem aufgefasst werden, dem das Management in Form ständiger Anpassung und Optimierung begegnet. Die Komplexität dieser Systeme verhindert eine Steuerung durch zentrale und vordefinierte Kontrollmechanismen [53]. Ebenso ist externe Ursache-Wirkungsanalyse schwierig [114, 153]. Daher nimmt Selbstadaptivität in Form der Erreichung 'zufriedenstellender' Systemzustände [88] einen bedeutenden Stellenwert ein.

Ein Versprechen der *Multiagententechnologie* ist Flexibilität in zweierlei Hinsicht: erleichterte externe Wartbarkeit und erweiterte Fähigkeiten, um notwendige Anpassungen

[4]Hierzu zählen unter anderem *selbst organisierend*, *selbst konfigurierend*, *selbst optimierend*, *selbst heilend* [173, 174, 210, 211]

selbständig durchführen zu können [130]. Ihre Fähigkeit, autonom und zielorientiert in dynamischen Umgebungen kollaborativ Entscheidungen zu treffen, ist ein vielversprechender Ansatz für die Anwendung innerhalb verteilter Informationssysteme. Als Repräsentanten von Diensten, Unternehmen, Organisationen oder physischen Ressourcen können *Softwareagenten* das Management von Informationssystemen unterstützen [237]. Der Einsatz intelligenter Softwareagenten steht in verteilten Systemen vor der Herausforderung des Umgangs mit Skalierbarkeit der verwendeten Konzepte, Methoden und Algorithmen [74, 247].

Vor dem Hintergrund der Optimierung stellen evolutionäre Verfahren einen erfolgversprechenden Ansatz dar. Implizite Parallelität, Robustheit und der Umgang mit komplexen Suchräumen bei gleichzeitig einfacher Umsetzung machen evolutionäres Computing zu einer echten Alternative im verteilten Management. Umsetzungen evolutionärer Verfahren nutzen trotz impliziter Parallelität weitgehend zentrale Instanzen zur Verwaltung von (Teil)Populationen. Dabei werden Potentiale insbesondere bei der Umsetzung in heterogenen verteilten Umgebungen, wie Grid oder Peer-to-Peer Netzwerken, nicht optimal genutzt. Qualitäts- und Lösungsbewertungen (sogenannte Fitnessfunktion) werden extern durchgeführt und sind problemspezifisch. In unternehmensübergreifenden betrieblichen Informationssystemen ist Zentralisierung und Externalisierung häufig nicht möglich oder nicht gewünscht. Daher sollten Entscheidungen und Lösungen lokal angestrebt werden, um in verteilten Umgebungen vor Ort zeitnah und effizient deren Umsetzung zu realisieren. Ebenso sind problemspezifische Bewertungen für veränderliche Probleme zu unflexibel.

1.2 Problemstellung

Software im betrieblichen Umfeld soll möglichst flexibel und zeitnah auf Nachfrageveränderungen reagieren. Im Allgemeinen erfolgt die Steuerung und Optimierung verteilter Informationssysteme im Top-Down Verfahren und nutzt dabei zentralisierte Infrastruktur. Beim Management verteilter und komplexer Software finden sich wesentliche Probleme, die zugleich durch bisherige Ansätze noch nicht ausreichend gelöst wurden. Die nachfolgenden Abschnitte gehen detailliert darauf ein.

1.2.1 Verteilung

Betriebswirtschaftliche und wissenschaftliche Optimierungsprobleme beinhalten eine ganze Reihe von Anforderungen an die zur Bearbeitung notwendige technische Umgebung. Idealerweise offerieren zentrale Lösungsansätze beträchtliche Vorteile: alle Daten und Informationen werden an einer Stelle gesammelt, gespeichert und verarbeitet. Die Lösung kann so erstellt werden, dass alle Fakten berücksichtigt werden. In der Praxis hingegen existieren Restriktionen, die Zentralisierung verhindern. Beispiele finden sich im Management der Netzwerkinfrastruktur von Telekommunikationsunternehmen [126] oder auch in logistischen Fragestellungen [97]. Weiss führt dazu an ([247], S. 112), dass Informationssysteme zu groß und komplex werden können, um diese noch zentral steuern oder optimieren zu können:

> "Planned information environments are too large, complex, dynamic, and open, to be managed centrally or via predefined techniques - the only feasible alternative is for computational intelligence to be embedded at many and sundry places in such environments to provide distributed control."

Die nachfolgenden Absätze geben einen Überblick zu unterschiedlichen Aspekten der Verteilung.

Problemverteilung

Verteilte Probleme sind nach Petcu [190] Probleme bei denen jede Variable durch einen eigenen Repräsentanten kontrolliert wird. Hierzu zählen räumlich und damit inhärent verteilte Probleme. Im logistischen Kontext [97] sind dies z.B. Repräsentanten verschiedener Unternehmen, die gemeinsam entlang einer logistischen Wertschöpfungskette Dienstleistungen gegenüber dem Endkunden erbringen. Die Problemstellung ergibt sich hierbei aus dem flexiblen und optimalen Zusammenspiel der einzelnen Akteure als Gesamtsystem. Ein anderes Beispiel für räumliche Problemverteilung sind sogenannte Sensor Netzwerke. Hierbei interagieren die Sensoren gemeinsam, um Aufgaben wie Monitoring eines Gebietes sicherzustellen. Neben räumlicher Verteilung ist Virtualisierung ein weiterer Punkt. Im Kontext von Grid-Systemen schließen sich unterschiedliche Akteure zeitlich begrenzt zu sogenannten *virtuellen Organisation* (*VO*) [107] zusammen, um gemeinsam Aufgaben zu bearbeiten. Die einzelnen Teilnehmer können sich weder gegenseitig kontrollieren, noch auf der Herausgabe wichtiger Informationen bestehen, die zur optimalen Zusammenarbeit benötigt werden. Daher existiert in diesen Szenarien keine explizite zentrale Kontrolle. Einer der aktuellen Trends ist die Verwendung und Nutzung verteilter Berechnungsressourcen, sogenannter Cluster [80]. Beispiele hierfür sind zum Einen die aktuelle Zusammensetzung der Top500 Supercomputer[5] und zum Anderen sind aktuelle Entwicklungen, wie das Internet, Grid-Systeme oder Peer to Peer (P2P) [180] substantieller Bestandteil der Infrastruktur geworden. Die Nutzung dieser verteilten und heterogenen Infrastruktur bietet enormes Potential und ist gleichzeitig eine der Herausforderungen an die nächste Generation entstehender Frameworks und Informationssysteme. Damit ergibt sich die Problematik der effizienten Gestaltung verteilter Systeme. Vom generellen Standpunkt sind damit verteilte Systeme als inhärent verteilte Probleme aufzufassen.

Operational Research (OR) stellt hierzu zentralistische Lösungsmethoden bereit, um optimale Ergebnisse zu erzielen. Es ist nicht entscheidend, woher die Verteilung resultiert, sondern ob eine Zusammenführung der Informationen im Kontext der Verteilung sinnvoll und vor allem auch möglich ist. Durfee et al. führt dazu an, dass einzelne Knoten in ihren Fähigkeiten limitiert sind [57]. Dazu zählen Beschränkungen der Bandbreite von Netzwerkverbindungen, Speicherplatzrestriktionen und auch begrenzte Berechnungskapazitäten. Obwohl technische Grenzen beständig erweitert werden, steigen die Anforderungen nach physikalischen Ressourcen mit zunehmender Größe und Komplexität verteilter Probleme ungleich stärker. Ein anderer Aspekt ist das Fehlen geeigneter zentraler Kontrollinstanzen wie in virtuellen Organisationen. Damit stehen der Zusammenführung und zentralen Bearbeitung in der Praxis die oben genannten Gründe im Weg.

Gesucht sind aus diesem Grunde Methoden und Verfahren zur Einbettung verteilter Steuerungs- und damit Optimierungsmöglichkeiten innerhalb des Systems. Damit werden zum Einen die Lösungsmöglichkeiten im System dezentral selbst durch die beteiligten Akteure erstellt und es erfolgt die Nutzung der bereits bestehenden Systemstrukturen. Zum Anderen wird ein ineffizienter Transfer und die Bearbeitung an zentraler Stelle vermieden. Die Herausforderung liegt hiermit in der Erstellung eines generellen dezentralen Ansatzes der an unterschiedliche Kontexte angepasst werden kann. Alba identifiziert Forschungsbedarf für

[5] siehe www.top500.org

verteilte evolutionäre Verfahren in lose gekoppelten Computerclustern, wie etwa Grid und Peer-to-Peer Systeme [2].

Informationsweitergabe

In mehrstufigen *Wertschöpfungsketten* (eng. *Supply Chains*) erstreckt sich das Optimierungspotential üblicherweise über die gesamte Länge der Kette. In den vergangenen Jahren sind die Einzelunternehmen immer effizienter geworden und haben das Ihnen zur Verfügung stehende Potential bereits weitgehend ausgeschöpft [115]. Dies entspricht der Erreichung lokaler Optimum für jedes einzelne Unternehmen. Ohne Zusammenarbeit mit seinen Zulieferern kann ein Unternehmen seine Wettbewerbsfähigkeit in einer solchen Situation alleine nicht mehr steigern. Dies kann durch Offenlegung der internen Kosten, z.B. dem sogenannten *Open Book Accounting* [62, 79, 165], geschehen. Kaum ein Unternehmen möchte allerdings über Open Book Accounting reden, aus gutem Grund: Zulieferer machen sich verwundbar, denn der Abnehmer bekommt Informationen, die seine Verhandlungsposition stärken. Er kann dann dessen Marge ausrechnen und entsprechend den Preis drücken. Weiterhin können mehrere Zulieferer gegeneinander ausgespielt werden, sollten diese ihre Kosten offenlegen.

Ein Austausch von Informationen wäre für das Gesamtsystem sicherlich die perfekte Lösung, allein die Praxis zeigt hier, dass ökonomisches und strategisches Verhalten Unternehmen dazu zwingt, sensible Informationen zurückzuhalten, um sich Wettbewerbsvorteile zu verschaffen. Virtuelle Organisationen sehen sich den gleichen Problemen ausgesetzt, da hier die zeitliche Volatilität erschwerend hinzukommt. Bei kurzzeitiger Zusammenarbeit ist Vertrauenswürdigkeit sehr schwer herzustellen. Die Weitergabe vertraulicher Informationen an Dritte fördert Abhängigkeiten und führt zu Wettbewerbsnachteilen. Ebenso liegen von legislativer Seite datenschutzrechtliche Beschränkungen in Form von Gesetzen vor. Luck et al. führt in [145] an, dass unterschiedliche Eigentümer von Diensten zu verschiedenen Zielen und damit zu konfligierenden Interessen führen. Die Rolle von Zielkonflikten wird von Timm [237] detailliert betrachtet.

Die durch Datenschutz verursachten Restriktionen systemweiter Optimierung sind von ihren Auswirkungen ähnlich wie bereits im Abschnitt 'Problemverteilung' dargelegt. Die Ursachen der Informationszurückhaltung sind hier allerdings nicht in fehlenden technischen Möglichkeiten zu suchen, sondern werden durch strategische Überlegungen geleitet. Die Herausforderung ist daher ein Mechanismus, der Anreize schafft, benötigte Informationen auszutauschen und gleichzeitig zwischen den Partnern eine Win-Win Situation herbeizuführen. Diese Natur solcher Anreize kann im betrieblichen Umfeld hauptsächlich durch ökonomischen Nutzen manifestiert werden.

1.2.2 Heterogenität

Begünstigt durch geringe Betriebskosten und breite Verfügbarkeit kommunikationsfähiger Hardware übernehmen zunehmend vernetzte handelsübliche Systeme die Rolle von Computergrids [33]. Verteilte Systeme setzen sich aus einer Vielzahl unterschiedlicher Teile zusammen, die sich in ihrer Hardware, Betriebssystem, Funktionalität, Zugehörigkeit, Umgebung und dem jeweiligen Kontext unterscheiden. Im Positionspapier Organic Computing [242] schreibt die Gesellschaft für Informatik (GI):

> "[...] wird keine fest konfigurierte Standard-Software-Basis mehr geben, die identisch auf allen Knoten des organischen Computers residiert. "

Damit kommt technischer und semantischer Interoperabilität der einzelnen Knoten entsprechend hohe Bedeutung zu. Es sind nicht allein technische Aspekte entscheidend, sondern auch Nutzervorgaben, gesetzliche Rahmenbedingungen und Organisationsziele zu berücksichtigen. Ebenso bedarf die Abfolge von Nachrichten in Form semantischer Kommunikationsprotokolle, wie etwa in der *Agent Communication Language ACL* [24], zusätzlicher Berücksichtigung. Hierbei ist Kommunikationsfähigkeit noch vor der Rechenleistung wichtigstes Kriterium [242]. Michael Luck bemerkt dazu in [145]:

> "[...] computing is something that happens by and through communication between computational entities. "

Software sollte Kommunikationsstandards einhalten und Optimierungsfunktionalität sollte mit unterschiedlichem Kommunikationsverhalten und Standards umgehen können. Weiterhin besteht das Problem, Steuerungs- und Optimierungsfunktionen innerhalb heterogener Umgebungen zu etablieren. Hierbei ist einfacher Integration- und Anpassung in bestehende Software der Vorzug vor Neuentwicklungen zu geben. Auch das Problemfeld der Bearbeitung veteilter evolutionärer Optimierung im heterogenen Umfeld ist noch nicht ausreichend gelöst [2, 33].

1.2.3 Dynamik

Bei der Betrachtung dynamischer Aspekte ergeben sich verschiedene Ebenen von Veränderungen. Verteilte und dynamische Probleme manifestieren sich zum Beispiel bei Nachfrageveränderungen in Versorgungsnetzwerken. Bei steigender Nachfrage ist die Struktur der Netze entsprechend anzupassen. Ein weiterer Aspekt sind verteilte und dynamische Systeme zur Bearbeitung von Problemen. Hier erfordert die Dynamik des Systems selbst entsprechende Maßnahmen um Problemlösungen zeit- und qualitätsgerecht zu erbringen.

Probleme sind veränderlich und erfordern Reaktionen in Form modifizierter Lösungen. Verteilte Probleme sind in dieser Hinsicht besonders zu beachten, da hierbei die Zeiten zum Einsammeln benötigter Daten und der Verteilung (engl. *Deployment*) der Lösung mit zu kalkulieren sind. Petcu [189] merkt dazu an:

> "Dynamic systems are another reason: by the time we manage to centralize the problem, it has already changed. "

Je mehr Zeit für die Problemlösung mit all den erforderlichen Schritten verwendet wird, desto wichtiger wird die Aktualität der Daten. In diesem Zusammenhang weisen Jäger und Boucke auf die Aktualität von Informationen als Bestandteil der Informationsbeschaffung hin [118]. Die Betrachtung von Informationslogistik führt zu einem weiteren Faktor: das problemlösende System selbst. Hierbei sind die Schaffung und das Management entsprechender Strukturen entscheidend. Im Gridkontext sind sogenannte virtuelle Organisationen (VO) mit der Problembearbeitung beauftragt. Hierbei ist die VO ein dynamischer Zusammenschluss zunächst unabhängiger Elemente. Das Management solcher Organisationen beinhaltet die Auswahl und Rollenzuweisung einzelner Teilnehmer ebenso wie dezentrale Verhandlungen über gegenseitige Vereinbarungen der Teilnehmer [74]. Selbst nach erfolgreicher Bildung kann jederzeit ein Knoten ausfallen oder aus der Organisation ausscheiden. Gefordert ist die Kompensation solcher Ausfälle und flexibles Bereitstellen alternativer Ressourcen.

Dynamische Umgebungen und die induzierten Probleme verlangen nach Adaptivität der verarbeitenden Systeme selbst. Ein Aspekt von Adaptivität ist der Erhalt erfolgreicher Strategien aus der Vergangenheit. Im Kontext von Grid Computing bedeutet dies den Erhalt einer in der Vergangenheit erfolgreichen virtuellen Organisation sowie deren Verbesserung. Dies führt bei wiederholt auftretenden ähnlichen Aufgaben zur Erhöhung der Effizienz durch verringerte Setup Kosten. Die Balance zwischen Exploration neuer Möglichkeiten und der Verbesserung bestehender Strategien wird im Kontext des maschinellen Lernens als *Exploration-Exploitation Dilemma* bezeichnet [228, 236]. Im betrieblichen Umfeld ist dies die Anpassung langfristiger Unternehmensstrategien bei gleichzeitiger Flexibiltiät gegenüber kurzfristigen Änderungen. Im Agentenkontext bedeutet dies eine Balance zwischen deliberativem und reaktivem Verhalten [237].

In der Agententechnologie existieren hierfür erste Ansätze [237, 260], die jedoch nicht ausreichend für den Einsatz in großen parallelen Systemen sind. Das Gesamtsystem soll in der Lage sein, auf langfristige Umgebungsänderungen und kurzfristige Veränderungen möglichst adaptiv und optimal zu reagieren.

1.2.4 Komplexität

Die Systemwissenschaft [16, 153, 181] verwendet den Begriff *komplexes System*[6] für eine Menge miteinander in nichtlinearer Wechselwirkung stehender Elemente. Die betrachteten Problemstellungen der Verteilung, Dezentralität und Dynamik werden durch das Konzept komplexer Systeme übergreifend vereinigt. Beispiele hierfür sind Proteine, Nervengewebe, Ökosysteme und Organisationen, deren Systemelemente Atome, Nervenzellen, Lebewesen und Individuen bilden. In den letzten Jahren hat das Interesse an komplexen System im Hinblick auf deren Analyse und Steuerung stark zugenommen, da immer mehr Parallelen zu aktuellen Informationssystemen heraustreten. Beispiele hierfür sind das Management verteilter Systeme [126, 187], das Finanzsystem [231] und *Supply-Chains*[7] [94, 97, 195].

Gemeinsam ist diesen Systemen, das Ursache-Wirkungsbeziehungen bereits dann nicht mehr erkennbar sind, wenn Interaktionen zwischen ihren Elementen als nichtlineare Rückkopplungen erfolgen (siehe hierzu Matthies [153]). Aus Sicht einer externen Kontroll- und Optimierungsinstanz erweist sich die Analyse einzelner Systemelemente und deren Beitrag zur Gesamtleistung damit als schwierig bis unmöglich. Ebenso problematisch sind Top-Down Ansätze als Modellierungs- und Steuerungsverfahren, da zum Teil keine übergeordnete Instanz existiert und Abhängigkeiten nicht baumartig sondern netzwerkartig verlaufen. Die Dekomposition und Analyse isolierter Elemente birgt die Gefahr der Übersimplifizierung. Wichtige Systemeigenschaften sind nicht mehr erkennbar und lassen sich nicht unmittelbar aus dem Verhalten einzelner Elemente ableiten. Solche *emergenten* Eigenschaften lassen sich beim Aufschaukeln der Bestellmengen in Supply-Chains als sogenannter *Bullwhip-Effekt* [140, 166] beobachten. Ein weiteres Beispiel ist entfernungsabhängiger Abbau von Futterquellen durch Ameisen [71, 251]. Allgemein ist emergentes Systemverhalten nicht direkt kodiert, sondern ergibt sich aus dem Zusammenspiel der Elemente und ihrer Interaktionen. Während natürliche Systeme ein hohes Maß an Adaptivität, Robustheit und Fehlertoleranz aufweisen, sind technische Systeme bislang nicht in der Lage diese Eigenschaften adäquat umzusetzen.

In den vorangegangenen Abschnitten wurden die Problembereiche Verteilung, Heterogenität, Dynamik und Komplexität verteilter Informationssysteme betrachtet. Damit entsteht

[6] Der Unterschied und die Abgrenzung zur Komplexitätstheorie der Informatik gibt Abschnitt 2.1.3

[7] auch Wertschöpfungsnetzwerk

die Aufgabe, einen Adaptions- und Optimierungsansatz für verteilte Systeme so zu entwickeln, dass die Bereitstellung verteilter Systemfunktionalität bei gleichzeitiger Ressourceneffizienz und Einhaltung vorgegebener Nebenbedingungen erreicht wird. Dabei ist ein effizienter Ansatz zu entwickeln, der für die oben genannten Problembereiche geeignet ist und unter den jeweiligen Bedingungen funktioniert. Die Steuerungs- und Optimierungsfunktionalität soll als Teil des Systems integriert und dadurch dezentralisiert werden [200, 247].

1.3 Zielsetzung

Um den in den vorangegangenen Abschnitten betrachteten Problemen gerecht zu werden, wird eine Kombination von Agenten und evolutionären Verfahren eingeführt. Dieser mittels sogenannter *evolutionärer Agenten* realisierter Adaptions- und Optimierungsmechanismus vereinigt die jeweiligen Vorteile beider Ansätze. Dazu zählen einerseits implizite Parallelität, Robustheit und Optimierung in komplexen Suchräumen und andererseits Flexibilität und Interaktionsfähigkeit von Agenten. Die Einbettung evolutionärer Agenten in verteilte Systeme stellt einen vielversprechenden Ansatz zur Selbstadaption und Selbstoptimierung dar. Hierdurch wird eine in Abbildung 1.2 skizzierte systemübergreifende Lösungsmöglichkeit dargestellt.

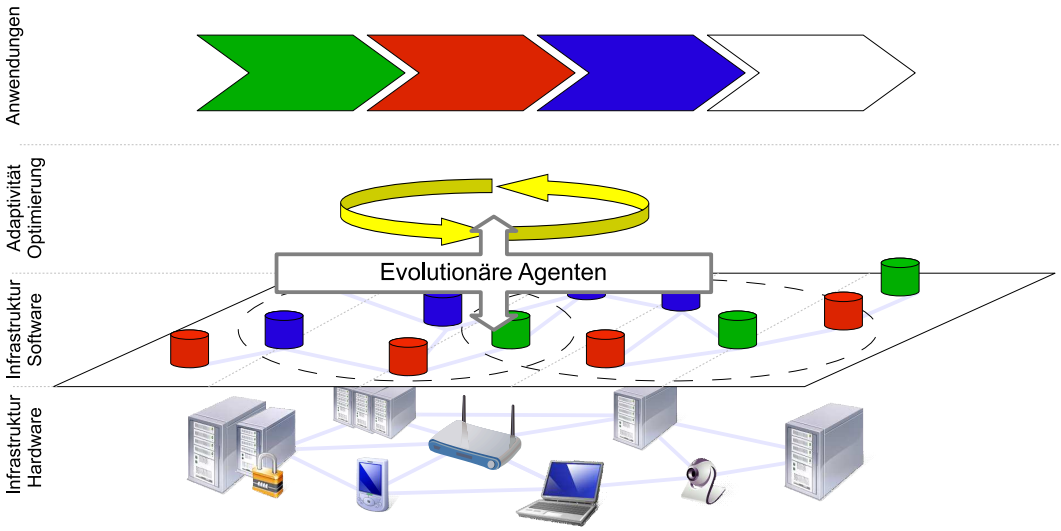


Abbildung 1.2: Einbettung evolutionärer Agenten als Adaptivitäts- und Optimierungsfunktionalität innerhalb verteilter Informationssysteme

Die Zielsetzung ist weiterhin die Entwicklung eines generellen und damit problem- und domänenunabhängigen Ansatzes. Hierzu wird zunächst ein formales Modell verteilter Optimierung benötigt, das sowohl agentenspezifische und evolutionäre Konzepte berücksichtigt. Aus dem Modell sind Aspekte der Adaptivität abzuleiten und schließlich durch geeignete Experimente zu belegen. Die Zielstellung umfasst im Einzelnen:

1. **Entwicklung formales Modell**: Ursache-Wirkungsanalysen lassen sich in komplexen Systemen nicht ableiten [153] und stellen damit neue Anforderungen an die Modellierung im Agentenkontext. Die Bottom-up Modellierung von Agent und evolutionären

Aspekten ermöglicht den Fokus auf lokaler Ebene und erhöht das Verständnis für die Repräsentation verteilter Probleme. Der die Anwendung in unterschiedlichen Domänen soll durch problemunabhängige Repräsentation Rechnung getragen werden. Das verteilte Optimierungsproblem wird aus Systemsicht formal definiert und mit der Definitionen des Multiagentensystems verknüpft. Ein weiterer Punkt ist die Verbindung von Systemfunktionalität mit Aspekten des Systemmanagements, um Managementansätze direkt innerhalb des Systems zu realisieren. Zur möglichst breiten Anwendbarkeit erfolgt die Verwendung bestehender Ansätze, sofern vorhanden.

2. **Dezentrales Adaptions- und Optimierungsverfahren**: Auf Basis des formalen Modells wird ein dezentraler evolutionärer Algorithmus erstellt. Die Herausforderung besteht in der Verknüpfung evolutionärer und agentenbasierter Problemlösung in einer Marktökonomie zum dezentralen Systemmanagement (Adaptivität und Optimierung auf Systemebene). Aus Sicht der künstlichen Intelligenz wird hierbei das Exploration-Exploitation Dilemma adressiert.

3. **Ableitung von Eigenschaften**: Bottom-up Modelle offenbaren nicht automatisch emergente Eigenschaften. Vielmehr lassen sich hierdurch komplexe Systeme durch vereinfachte Modelle der Elemente repräsentieren. Die Herausforderung für das Modell besteht darin, geforderte Eigenschaften auf Systemebene vorherzusagen. Auf der Basis des Modells evolutionärer Agenten wird das Systemverhalten abgeleitet.

4. **Evaluation**: Mit der Anwendung an zwei unterschiedlichen Problemen (Adaptive Logistiknetzwerke und Facility Location in Mixed Mode Environments) wird der Nachweis zur Realisierbarkeit erbracht. Dabei ist primär der Nachweis der vorausgesagten Eigenschaften auf lokaler (Agent) und globaler (System) Ebene relevant. Darüber hinaus wird die generelle Umsetzbarkeit in unterschiedlichen Agentenframeworks gezeigt.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in fünf Kapitel. Abbildung 1.3 stellt deren Struktur und ihre Abhängigkeiten dar. Die einleitende Darstellung von Ausgangssituation, Problemstellung und Zielsetzung inklusive Vorgehensweise erfolgte in **Kapitel 1**.

Kapitel 2 beschäftigt sich mit dem aktuellen Stand der Forschung. Hierzu werden zunächst aktuelle relevante Informationssysteme und deren Eigenschaften analysiert. Hierbei wird gezeigt, dass technische Systeme immer mehr Eigenschaften komplexer Systeme aufweisen. Aus dieser Analyse ergeben sich Schlussfolgerungen für den Aufbau und die Funktionsweise von Strukturen zur Anpassung und Optimierung moderner Informationssysteme. Darüber hinaus werden die informatischen Grundlagen von Multiagentensystemen, evolutionären Verfahren und deren Kombination erläutert, sofern diese für evolutionäre Agenten relevant sind. Leser können dieses Kapitel teilweise oder ganz überspringen, sofern sie mit den Grundlagen vertraut sind.

Kapitel 3 stellt das Konzept und die Funktionsweise evolutionärer Agenten vor. Hierzu erfolgt zunächst die formale Erstellung eines Modells. Anschließend wird das dezentrale Optimierungsverfahren vorgestellt und schließlich ausgehend von lokalen Eigenschaften eine Analyse resultierender Systemeigenschaften durchgeführt.

Die Anwendung an den beiden Problemstellungen und deren Evaluation beschreibt **Kapitel 4**. Zunächst wird eine Referenzarchitektur aus dem formalen Modell abgeleitet. In den

nächsten beiden Schritten erfolgt jeweils die Übertragung des generischen Ansatzes auf die konkreten Problemstellungen der Adaptivität von Logistiknetzwerken und Facility Location.

Abschließend gibt **Kapitel 5** einen Gesamtüberblick über die Arbeit und deren Ergebnisse. Es werden mögliche Weiterentwicklungen und Anwendungsgebiete thematisiert.

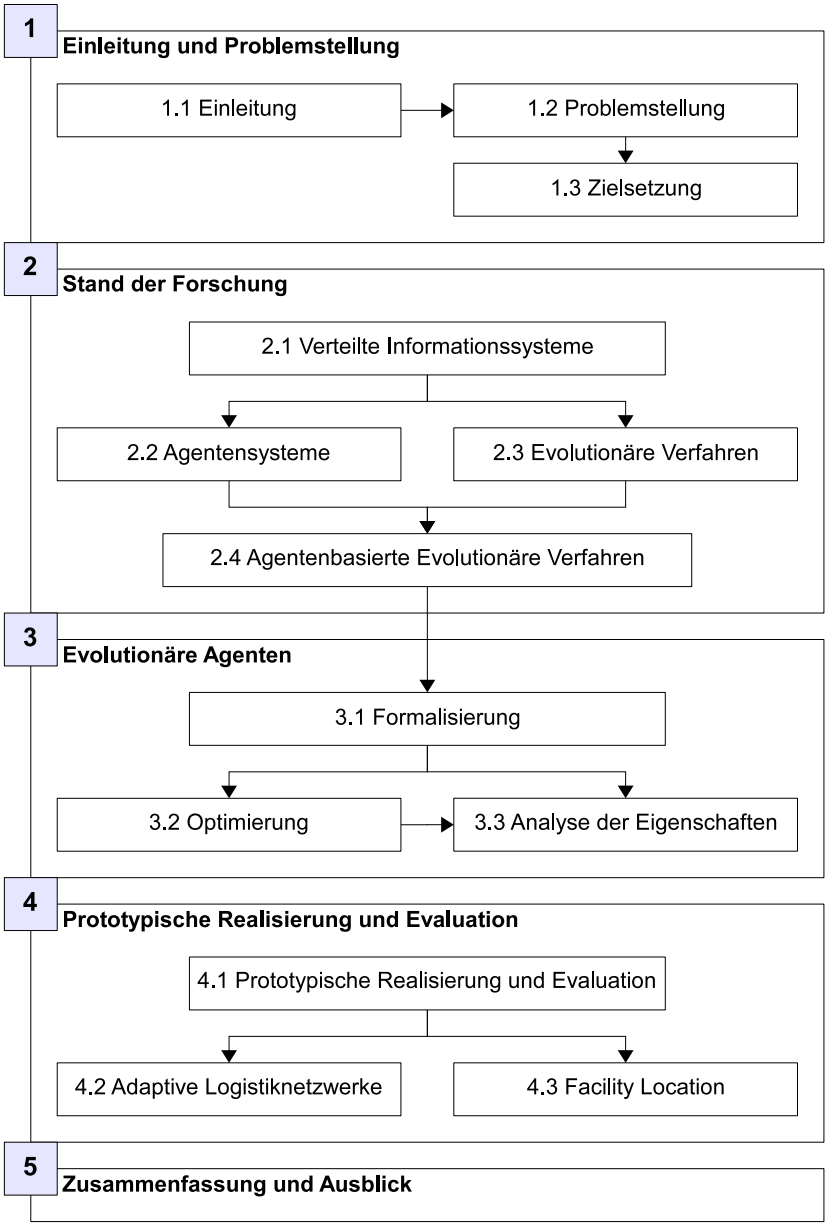


Abbildung 1.3: Aufbau der Arbeit

Kapitel 2

Stand der Forschung

Diese Arbeit vereint unterschiedliche Ansätze in einem interdisziplinären Kontext. Daher ist das Ziel dieses Kapitels ein Überblick der relevanten Forschungsbereiche hinsichtlich der gestellten Zielsetzung (siehe Kapitel 1.3). Die Vorgehensweise ist daher zunächst die Identifikation allgemeiner Eigenschaften aktueller und möglicher zukünftiger verteilter Informationssysteme (*Komplexe Adaptive Systeme*) in Kapitel 2.1. Auf dieser Basis werden die beiden verwendeten Forschungsansätze in den Kapiteln 2.2 und 2.3 ausführlich beschrieben. Es erfolgt eine Abgrenzung hinsichtlich weiterer relevanter Verfahren. Nachfolgend geht Kapitel 2.4 näher auf das Forschungsgebiet im Schnittbereich evolutionäre Verfahren und Agenten ein. Abschließend erfolgt eine Zusammenfassung in Kapitel 2.5.

2.1 Verteilte Informationssysteme

Zur Beschreibung von Systemen im Sinne der Systemtheorie existiert eine ganze Reihe von Definitionen. Im besonderen Interesse dieser Arbeit liegt es, grundlegende Gemeinsamkeiten *verteilter Informationssysteme* zu nennen. Der Begriff System soll daher hier im technischen Sinne für Informations- und Kommunikationssysteme verstanden werden und wird im Rahmen dieser Arbeit auf Software-Systeme und deren Einbettung in einen Umgebungskontext beschränkt. Die folgenden Kapitel beschreiben zunächst grundlegende Eigenschaften und Begriffe aus der Systemtheorie. Darauf aufbauend wird der Begriff Informationssystem definiert und anschließend folgen Ausprägungen aktueller verteilter Informationssysteme.

2.1.1 Systembegriff

Ein *System* ist eine Menge von Elementen die in Beziehung zueinander stehen. Daraus folgt, dass es auch Elemente geben muss, die sich außerhalb des Systems befinden [153]. Diese gehören zur Systemumgebung (oder kurz Umgebung), in die ein System eingebettet ist. Die Berührungspunkte von System und *Systemumgebung* (oder auch kurz *Umgebung*) werden als *Systemgrenze* bezeichnet. Ein *isoliertes* System ist ein System ohne Input oder Output zur Umgebung. Im Sinne von physischen Systemen wird die Unterscheidung *abgeschlossen* bzw. *geschlossen* für Systeme verwendet, die nur Energie aber keine Materie mit der Umgebung austauschen können. Geschlossene Systeme besitzen mindestens einen Input und einen Output. *Offene* Systeme können sowohl Energie als auch Materie mit der Systemumgebung austauschen. Diese Eigenschaft macht eine Berechnung offener Systeme nicht möglich [146].

Auf jedes *dynamische* System gibt es eine *Struktursicht* und eine *Verhaltenssicht*. Die Struktursicht betrachtet die Elmente des Systems, deren Eigenschaften und ihre Wechselwirkungen. Die Verhaltenssicht beschreibt die im Zeitverlauf beobachtbare Entwicklung.

2.1.2 Emergenz

Um verteilte Systeme besser zu verstehen, bedarf es der Betrachtung von Eigenschaften auf lokaler (Elemente bzw. Agenten) und globaler Ebene (System). Mit der Betrachtung des Begriffes *Agent* eng verknüpft ist der Begriff *Emergenz*. Eine Voraussetzung für das Entstehen von Emergenz ist das Vorhandensein mehrerer miteinander interagierener Elemente - auch als Agenten bezeichnet. Emergenz ist das Entstehen eines nicht explizit definierten Systemverhaltens (Makroebene) auf Basis von Interaktionen zwischen Agenten und deren Verhaltensmustern (Mikroebene). Nachfolgend sind Beispiele für emergentes Verhalten aufgeführt.

Emergente Eigenschaften lassen sich beim Aufschaukeln der Bestellmengen in Supply-Chains als sogenannter *Bullwhip-Effekt* [140, 166] beobachten. Hierbei führen kleine Nachfrageschwankungen auf Kundenseite zu immer stärkerem Aufschaukeln der Bestellmengen entlang der *Wertschöpfungskette*. Obwohl die Verhaltensweisen der einzelnen Teilnehmer völlig transparent sind, kommt es zu den Aufschaukelungen. Eine bekannte Anwendung in diesem Zusammenhang ist das sogenannte *Beer Game* [225]. Hier spielen menschliche Teilnehmer nach einfachen Regeln eine Wertschöpfungskette.

Ein weiteres Beispiel findet sich in Ameisenkolonien [71]. Hier werden Futterquellen mit sehr hoher Wahrscheinlichkeit anhand ihrer Entfernung zum Nest abgebaut. Zuerst werden die Nächsten und erst dann die weiter Entfernten erschlossen [251]. Dieses auf Makroebene (globales Verhalten der Kolonie) in der Natur und Simulation beobachtbare Verhalten existiert auf der Mikroebene der Ameisen (lokales Verhalten) nicht. Diese Verhaltensmuster (Abbau von Futterquellen anhan ihrer Entfernung vom Nest) entstehen im Zusammenspiel einer großen Anzahl von Ameisen und Duftstoffe (sog. *Pheromone*).

Auch evolutionäre Verfahren [22, 88, 223] führen zur Entstehung emergenter Eigenschaften in Form von Problemlösungen. Die evolutionären Prozessschritte (Fitnessberechnung, Selektion, Rekombination, Mutation) enthalten keine konkreten Anweisungen zur Konstruktion optimaler Lösungen. Vielmehr sind sie abstrakt und problemunabhängig zur Verarbeitung von Lösungen formuliert.

Nach Kubik [138] sind die Quellen für das Entstehen von Emergenz die Eigenschaften von Agenten, Kommunikation zwischen den Agenten, Einflüsse der Umgebung und die Evolution von Agenten und Umgebung. Basierend auf diesen Eigenschaften schlägt Kubik [138] fünf Kategorien von Multiagentensystemen mit steigender Emergenz vor:

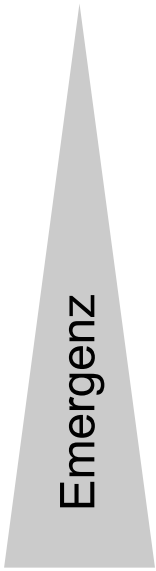
1. Agenten interagieren in einer statischen Umgebung. Diese liefert kein Feedback auf Aktionen an die Agenten. Das Verhalten der Agenten verändert sich nicht.

2. Die Umgebung gibt Feedback und beeinflusst damit die Aktionen der Agenten. Agenten und Umgebung verändern sich nicht weiter.

3. Die Agenten entwickeln sich in einer statischen Umgebung. Die Eigenschaften und das Verhalten der Agenten ändern sich über die Zeit.

4. Die Agenten entwickeln sich weiter und die Umgebung gibt Feedback.

5. Sowohl Agenten als auch die Umgebung entwickeln sich über die Zeit. Neue Eigenschaften und Verhaltensweisen entstehen.



Als Arbeitsdefinition wird das Auftreten von Effekten auf Makroebene bezeichnet, die auf Mikroebene nicht definiert sind. Als Teil dieser Arbeit wird Emergenz von Verhaltensmustern auf Systemebene (Makroebene) untersucht, die auf Agentenebene (Mikroebene) nicht definiert sind. Es ist jedoch nicht Ziel dieser Arbeit, neue Definitionen oder Forschungen auf dem Gebiet der Emergenz zu liefern.

2.1.3 Komplexe Systeme

Struktur(elemente) und Verhalten von Systemen stehen in Wechselwirkung zueinander. Das Verhalten eines Systems kann Änderungen der Struktur bewirken und ebenso hat die Struktur Einfluss auf das Verhalten. Als *komplexes Verhalten* wird nach Kratky [137] das Vorhandensein von mindestens einem Rückkopplungskreis und eine nichtlineare Wechselwirkung zwischen den Systemelementen bezeichnet. Bar-Yam [16] charakterisiert *komplexe Systeme*, als Systeme die eine nichttriviale interne Struktur haben, die sich im zeitlichen Verlauf ändert. Eine Ursache-Wirkungs Analyse ist bereits dann nicht mehr möglich [153].

Ein wesentliches Merkmal komplexer Systeme ist die Herausbildung emergenter Eigenschaften. Diese ergeben sich aus der Interaktion der Systemelemente. Tesfatsion beschreibt komplexe Systeme als Systeme mit den folgenden Eigenschaften [234]:

- Das System besteht aus miteinander interagierenden Elementen
- Das System zeigt emergente Eigenschaften, die nur aus der Interaktion der Elemente bestehen. Emergente Eigenschaften sind nicht Bestandteile der Elemente.

Es sind damit zur Beschreibung eines komplexen Systems zusätzliche Informationen auf Systemebene notwendig. Die Summe der Beschreibungen der Elemente reicht dazu nicht aus. Ein Maß zur Beschreibung der Komplexität von Daten ist die *Kolmogorov Komplexität* [142, 218]. Die zentrale Idee hierbei ist die Programmlänge zur Beschreibung der Daten im Vergleich zur Länge der Daten selbst. Sei dazu x eine Menge von Daten und $|x|$ die Länge

der Daten. $K(x)$ ist die Länge des kürzesten Programms, das x ausgibt und dann stoppt. Es existiert immer ein Programm, das die Daten einfach ausgibt, in etwa folgender Form '`print(x); end;`'. Damit gilt für die Programmlänge $K(x) \leq x + c$ mit c als Konstante. Gibt es kein kürzeres oder gar ein Programm mit konstanter Länge, dann gilt $K(x) \approx |x|$. Damit ist x komplex. Andere Daten hingegen lassen sich sehr gut komprimieren und durch kurze Programme reproduzieren, wie etwa Strings beliebiger Länge als Wiederholung gleicher Zeichen: $x = 'aaaaa \ldots a'$, $|x| = const$. Hier lässt sich x als kurzes Programm mit konstanter Länge für beliebige $|x|$ realisieren. Auch die Berechnung und Ausgabe von $x = \pi$ in beliebiger Genauigkeit kann ein konstant kurzes Programm erledigen.

Die Kolmogorov Komplexität ist in diesem Zusammenhang zu unterscheiden vom Begriff der *Komplexität* in der Informatik [213]. Die Komplexitätstheorie im informatischen Sinne drückt den Bedarf an Rechenressourcen als Rechenzeit oder Speicherplatz als Funktion in Abhängigkeit der Eingabenlänge n durch $\mathcal{O}$ aus. Demgegenüber steht Kolmogorov's Definition als Reproduktion von Daten in Abhängigkeit zur Programmlänge.

In dieser Arbeit steht zunächst die Auseinandersetzung mit dem Thema komplexe Systeme im Vordergrund und daher wird der Begriff komplex in diesem Zusammenhang genutzt. Wird komplex im Zusammenhang der informatischen Berechenbarkeitstheorie genutzt, erfolgt dies explizit bzw. ist aus dem Kontext unmittelbar zu erkennen.

2.1.4 Informationssysteme

Ein Software-System ist nach Balzert [14] ein System, dessen Systemkomponenten und Systemelemente aus Software bestehen. Abbildung 2.1 zeigt die Einordnung von Begriffen, die im Zusammenhang dieses Verständnisses nach Balzert [14] üblich sind.

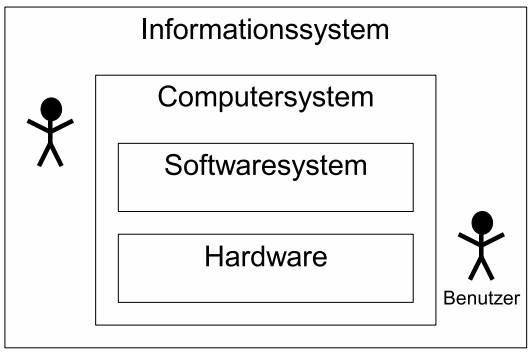


Abbildung 2.1: Begriffe im Zusammenhang des Systemverständnisses: Softwaresystem, Hardware, Computersystem, Informationssystem

Da Software immateriell ist, ist für das Vorhandensein von Software immer Hardware zur Erfüllung der intendierten Aufgabenstellung notwendig. Software und Hardware bilden zusammen das Computersystem. Der Aufbau und Betrieb von Softwaresystemen hängen maßgeblich von äußeren Einflussfaktoren ab [194], z.B. die Organisation, innerhalb derer das Computersystem bestimmte Aufgaben erfüllt. Diese Gesamtheit aus organisatorischem System und Computersystem wird als computergestütztes Informationssystem oder kurz als Informationssystem bezeichnet.

2.1.4.1 Relevante Verteilte Systeme

Der stetig steigende Bedarf nach Rechenleistung und die Vernetzung bilden die Basis verteilter Informationssysteme. Um einen Trend verteilter Informationssysteme aufzuzeigen, gibt die Klassifikation von Flynn [70] aus dem Jahre 1972 einen ersten Ansatz. Hier erfolgt die Einteilung anhand zweier Dimensionen: Anzahl der Befehlsströme und Anzahl der Datenströme, wobei in beiden Dimensionen zwischen einem und mehreren unterschieden wird.

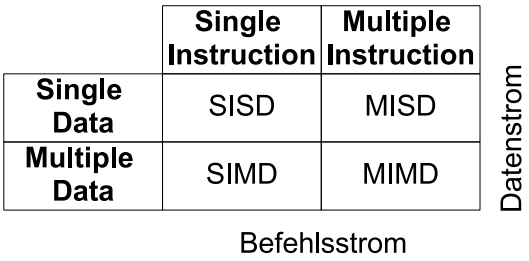


Abbildung 2.2: Flynn's Klassifikation [70] nach Anzahl der Befehls- und Datenströme

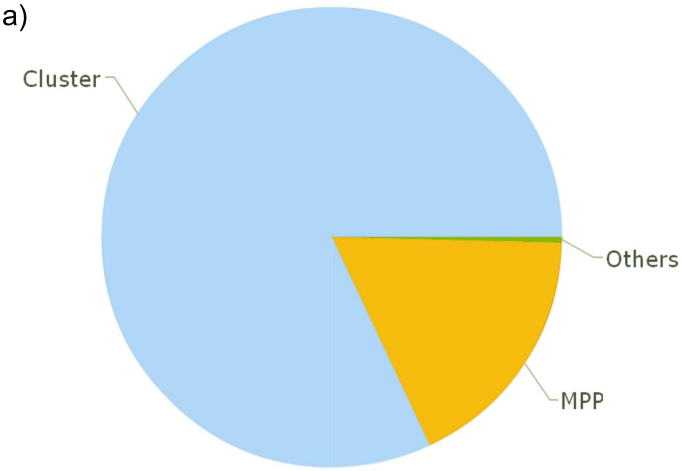
Flynns Klassifikation:

- **SISD**: Klassischer Einprozessor-PC.
- **SIMD**: Alle Prozessoren führen in jedem Schritt die gleichen Instruktionen auf verschiedenen Daten aus
- **MISD**: Unterschiedliche Instruktionen werden auf den selben Daten ausgeführt
- **MIMD**: Unterschiedliche autonome Prozessor arbeiten gleichzeitig und unabhängig auf verschiedenen Daten

Im Sinne der Flynn'schen Klassifikation [70] lassen sich verteilte Systeme in der Kategorie MIMD (engl. Multiple Instruction, Multiple Data) einordnen, denn hier sind die einzelnen Elemente des Systems über ein Netzwerk verbunden und kommunizieren und koordinieren ihre Aktionen über Nachrichten [48, 232]. Somit bearbeitet jeder Prozessor nur die eigenen Daten. Die Flynn'sche Klassifikation ist aus heutiger Sicht bereits zu grob, da für MIMD keine Unterscheidung zwischen gemeinsamem oder getrenntem Adressraum für die beteiligten Prozessoren unterschieden wird [4]. Tatsächlich spielt aktuell (November 2008) ein gemeinsamer Adressraum keine nennenswerte Rolle mehr, wie ein Blick auf die Architekturübersicht der 500 leistungsstärksten Supercomputer (www.top500.org [157]) zeigt. 99,6% der verwendeten Systeme sind MIMD Systeme mit stetig steigendem Anteil (Cluster, MPP). Es sind sogenannte lose gekoppelte Systeme bestehend aus einzelnen Rechnern mit eigenem Adressraum und Prozessor(en). Aufgrund der Fokussierung auf Verteilte Systeme werden SISD, SIMD und MISD nicht weiter betrachtet.

Den weitaus größten Teil dieser Systeme nehmen sogenannte Cluster ein, deren Anteil in den letzten zehn Jahren von null auf 82% gestiegen ist. Diese bestehen aus heterogenen Standardrechnern kommunizieren und über ein Netzwerk [11, 12]. Durch die Verwendung von Standardkomponenten sind Cluster im Vergleich mit anderen Supercomputerarchitekturen kostengünstiger. Der zweite nennenswerte Architekturtyp MPP (*Massive parallel processing* - Massive Parallelverarbeitung) ist mit einem sinkenden Anteil von 17,6% vertreten. Im Unterschied zum Cluster enthalten MPP-Systeme mehr Einzelrechner, die stärker gekoppelt

a)



b)

Architektur	Anzahl	Anteil
Cluster	410	82,00%
MPP	88	17,60%
Andere	2	0,40%
Summe	500	100,00%

Abbildung 2.3: Top500 Architekturzusammensetzung 11/2008 (siehe www.top500.org, Abruf vom 20.04.2009)

sind. Während Cluster üblicherweise mehreren Organisationen gehören können, gehören MPP meist einer Organisation. In beiden Architekturen werden Daten über Nachrichten im Netzwerk ausgetauscht. Im Sinne des hier genutzten Verständnisses verteilter Systeme gehören damit Grid-Systeme [74, 77, 78] ebenso wie das Internet, Unternehmensnetzwerke oder auch SETI@home[1] und Folding@home[2]. Der Trend geht damit hin zu lose gekoppelten Systemen, die auf Standardhardware basieren und die sich über unterschiedliche Organisationen spannen [11, 12].

Das bedeutet sowohl für aktuelle Supercomputer wie auch für verteilte Systeme eine Verteilung der Informationen über mehrere Knoten und zumindest für Cluster gegebenenfalls über mehrere Organisationen hinweg. Zugriff und Manipulation von Daten und Informationen fremder Knoten sind nicht direkt, sondern nur indirekt über Nachrichten möglich. Damit beschränkt sich die Auswahl der zu betrachtenden Ansätze in diesem Kapitel auf lose gekoppelte Systeme, die aus heterogenen Einzelrechnern bestehen und nachrichtenbasiert kommunizieren. Die nächsten Abschnitte geben einen Überblick über einzelne Entwicklungen in diesem Bereich. Eine nähere Untersuchung von Systemarchitekturen ist nicht Gegenstand dieser Arbeit und wird daher nur am Rande betrachtet.

2.1.4.2 Serviceorientierte Architekturen

Sogenannte *Webservices* bilden die Grundlage von *Serviceorientierte Architekturen* (SOA). Die Idee hierbei besteht in der Dekomposition und Kapselung einzelner Funktionen und deren standardisierte und XML-basierte Bereitstellung über ein Netzwerk. Abbildung 2.4 zeigt schematisch die Rollen in einer serviceorientierten Architektur und deren verwendete Standards.

Die Bestandteile von Serviceorientierten Architekturen lassen sich wie folgt charakterisie-

[1] setiathome.berkeley.edu - Über das Internet vernetzte Computer zur Suche nach extraterrestrischer Intelligenz

[2] folding.stanford.edu - Über das Internet vernetzte Computer zur Untersuchung der Proteinfaltung und damit verbundenen Erkrankungen

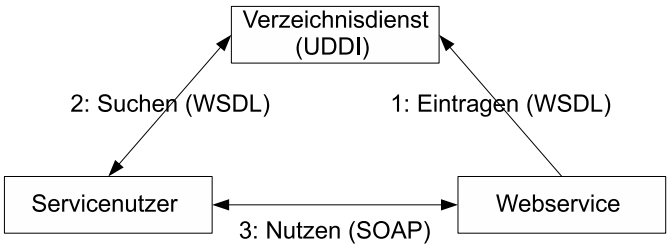


Abbildung 2.4: Webservice Funktionsweise und Protokolle

ren:

Webservice ist der Diensterbringer,

Servicenutzer verwendet Webservices und

Verzeichnisdienst enthält eine Liste aller registrierten Services.

Zunächst ist ein Webservice unter einer eindeutigen Adresse zu erreichen. Theoretisch registriert sich der Service als erstes bei einem Verzeichnisdienst, um seine Funktionalität bekannt zu machen. Praktisch wird diese Funktionalität nicht mehr verwendet. Dazu trägt er seine sogenannte Dienstbeschreibung mit den bereitgestellten Funktionen und Parametern in den Verzeichnisdienst ein. Hierbei wird der XML-basierte Standard *Web Service Description Language* (*WSDL*) verwendet. Nach der Registrierung ist die Beschreibung des Services für Servicenutzer abrufbar. Verzeichnisdienste verwenden dazu das sogenannte *Universal Description, Discovery and Integration* (*UDDI*) Format. Es dient der Beschreibung und Suche von Webservices. Durch die Suche im Verzeichnisdienst nach angebotenen Funktionalitäten gelangt der Servicenutzer schließlich an Adressen geeigneter Webservices. Das Wissen über die Syntax der Funktionsaufrufe und Parameter erhält er aus der WSDL Beschreibung. Mit Hilfe dieser Daten erfolgt schließlich der eigentliche Schritt: die Serviceausführung.

Dies geschieht über *SOAP* - ein Standard, mit dessen Hilfe Daten zwischen dem Servicenutzer und dem Webservice ausgetauscht und Remote Procedure Calls durchgeführt werden können. Ursprünglich war SOAP die Abkürzung für *Simple Object Access Protocol* (= Einfaches Objekt-Zugriffsprotokoll), seit Version 1.2 ist SOAP jedoch offiziell keine Abkürzung mehr, da es nicht (nur) den Zugriff auf Objekte ermöglicht. Vielmehr dient SOAP als Aufruf-Mechanismus für Webservices, Komponenten und eben auch für Objekte. SOAP stützt sich auf die Dienste anderer Standards: XML zur Repräsentation der Daten und Internet-Protokolle (z.B. HTTP, HTTPS) zur Übertragung der Nachrichten.

Serviceorientierung im Geschäftsumfeld nutzt die durch Webservices bereitgestellte technologische Basis, um mit Hilfe von Webservices Prozesse dynamisch und flexibel zur Laufzeit zusammenzustellen und auszuführen. Die Vorteile der Serviceorientierung liegen auf der Hand: offene Kommunikationsstandards bilden die gemeinsame Grundlage und sind für jedermann nutzbar. Standardprotokolle ermöglichen den Zugriff aus allen Netzen auch durch Firewalls hindurch. Die Fokussierung auf Standards erlaubt eine Realisierung in nahezu jeder Programmiersprache und den Einsatz auf heterogener Hardware. Die Flexibilität des Servicediscovery macht eine dynamische Nutzung unterschiedlicher Services möglich.

In der Praxis wird jedoch das Eintragen und Suchen mittels UDDI so gut wie nicht verwendet. Vielmehr wird Zugriff auf Webservices von den Servicenutzern bereits im Quellcode

kodiert. Damit entfällt die Flexibilität der Servicediscovery zur Laufzeit zugunsten verringerten Realisierungsaufwandes. Die Performance ist durch den Einsatz von XML ungünstig, was jedoch als zweitrangig angesehen werden kann. Damit sind fehlende Flexibilität und schlechte Skalierbarkeit die Hauptkritikpunkte Serviceorientierter Architekturen.

2.1.4.3 Grid Computing

Der Begriff Grid steht für Infrastruktur, die sicheren Zugang auf verteilte Ressourcen bereitstellt. Dazu existieren Protokolle und sogenannte *Middleware*, die Zugriff auf Funktionen zur Suche, Vereinigung und Nutzung von Diensten anbietet. Die Dienste sind als Weiterentwicklung von Web-Services zu sehen. Durch Nutzung heterogener Ressourcen kommt der Standardisierung von Schnittstellen und Protokollen eine bedeutende Rolle zu.

Foster und Kesselmann schreiben dazu [76]:

> "[...] is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization."

In der Forschung existieren eine Reihe von Grid Werkzeugen, wie das *Globus Toolkit* (*GT*) [73] oder *g-Eclipse*[3] [133]. Auch große Internet Dienstleister bieten mittlerweile ähnliche Dienste an, wie Google mit Google Apps[4] oder Amazons Cloudcomputing mit Amazon EC2[5]. Die Abgrenzung zu verwandten Begriffen und Konzepten ist nicht immer einfach, daher bietet Foster als einer der Mitbegründer des Grid Computing eine drei Punkte Checkliste [72] an. Ein Grid ist ein System, dass

Ressourcen koordiniert die nicht unter zentraler Kontrolle stehen. Das Grid integriert und koordiniert Ressourcen und Nutzer in unterschiedlichen Domänen. Es adressiert die verschiedenen Bereiche Sicherheit, Richtlinien, Abrechnung, Domänenzugehörigkeit und weitere, die in diesem Zusammenhang auftreten.

Standardisierte, offene und allgemeingültige Protokolle und Standards verwendet. Diese adressieren grundlegende Dinge, wie Authentifikation, Authorisation, Ressourcensuche und -zugriff.

Nichttriviale *Quality of Service* (QoS) Dienstleistungen anbietet. Zusammengeschaltete Dienste offerieren verschiedene Servicedienstgrade hinsichtlich Durchsatz, Verfügbarkeit, Sicherheit und weiterer Dimensionen.

Ein wichtiges Merkmal sind sogenannte *Virtuelle Organisationen* (VO), die durch den Zusammenschluss von Diensten entstehen. Hierbei arbeiten initial unabhängige und auch

[3] siehe http://www.geclipse.org/
[4] http://code.google.com/intl/en/appengine/
[5] aws.amazon.com/ec2/

verschiedenen Organisationen angehörende Dienste zeitlich begrenzt zusammen, um komplexe Dienste anzubieten bzw. Aufgaben zu bearbeiten. Die Beziehungen können dynamisch variieren in Abhängigkeit von Zugriffsbeschränkungen einzelner Dienste. Zum Beispiel bieten VO bestimmte Leistungen nur denjenigen an, die ein Sicherheitszertifikat besitzen. Die Rolle einzelner Dienste ist per se definiert, sondern wird im jeweiligen Kontext ausgewertet. Ein Dienst kann damit gleichzeitig die Rolle eines Kunden einnehmen und selbst Funktionen anbieten. Als Beispiel sei ein Reisebuchungsservice angeführt, der einzelne Dienstleistungen, wie Mietwagen und Flüge von anderen 'zukauft'. Hierbei geht die gemeinsame Dienstleistung über die Fähigkeiten eines einzelnen Dienstes weit hinaus.

Die Architektur muss den VO einen effektiven Aufbau von Strukturen zwischen beliebigen Teilnehmern ermöglichen. Hierbei spielt *Interoperabilität* eine zentrale Rolle. Daher ist die Grid Architektur primär eine Protokoll Architektur [78] zur Verhandlung, Aufbau, Management und der Nutzung von Verbindungen zwischen Nutzern und Diensten. Eine auf Standards basierende offene Architektur stellt ermöglicht Erweiterbarkeit, Interoperabilität, Portabilität und Codesharing. Es ist ebenso möglich, Softwareschnittstellen und Development Kits auf Basis der Dienste zur Verfügung zu stellen. Eine auf diesen Prinzipien vorgeschlagene schichtenbasierte Protokoll Architektur findet sich in [78] und ist in Abbildung 2.5 dargestellt.

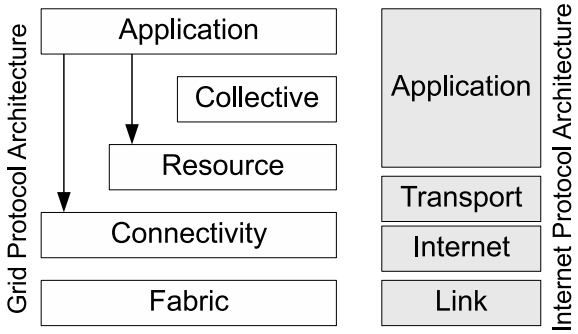


Abbildung 2.5: Schichtenbasierte Grid Architektur und Verbindung zum Internetprotokoll [78]

Die Anwendungsschicht (Application) beinhaltet Anwendungen innerhalb virtueller Organisationen. Diese bestehen aus Diensten und greifen selbst auf Dienst der unteren Schichten zu. In der Verbandsschicht (Collective) sind Protokolle und Dienste gekapselt, um Dienste zu koordinieren. Der gemeinsame Zugriff auf einzelne Dienste wird in der Ressourcenschicht (Resource) geregelt. Kommunikations- und Authentifikationsprotokolle für Grid-spezifische Netzwerktransaktionen sind in der Konnektivitätsschicht (Connectivity) zusammengefasst. Schließlich stellt die Fabrikschicht (Fabric) die Ressourcen bereit, welche durch die übergeordneten Schichten angesprochen werden können.

Das Globus Toolkit wird seit Ende der 90er Jahre entwickelt und bietet eine große Bandbreite von Funktionen als Dienste. Diese Dienste werden wiederum zur Entwicklung und dem Aufbau von Grid Infrastruktur und verteilten Anwendungen genutzt. Die Architektur [73] ist in drei Bereiche aufgeteilt:

- Eine Menge von **Infrastrukturdiensten** zur Bereitstellung von Ausführungsmanagement, Datenzugriff und Verschiebung, Replikationsmanagement, Monitoring und Su-

chen, Zugriffskontrolle und Gerätemanagement. Die meisten sind Java basierte Webservices, aber auch andere Sprachen und Protokolle sind möglich.

- Drei **Container** wurden zum Hosting von Java, Phyton und C basierten Services entwickelt. Die Container stellen Funktionalität bereit wie Sicherheit, Management, Discovery, Zustandsmanagement etc..
- Eine Menge von **Client Bibliotheken** erlaubt Java, Phyton und C Client Programmen die Ausführung von GT Diensten und User Diensten.

Die Schnittstelle zwischen Infrastrukturdiensten und User Diensten ist WS-I (*Web Services Interoperability*), eine Weiterentwicklung des *SOAP*-Protokolls. Damit baut das Globus Toolkit auf bewährte Technologien auf und fungiert gleichsam als Basis für Gridfunktionalität.

Existierende Grid-Technologien und insbesondere auch OGSA-basierende Grid-Middleware stellen bislang die Bereitstellung stabiler, sicherer, zuverlässiger und skalierbarer Plattformen zur gemeinsamen Nutzung verteilter Ressourcen durch kooperierende Partner mit gemeinsamen Zielen in den Vordergrund [74, 78]. Zur Koordination von Grid-Diensten werden überwiegend vordefinierte Protokolle oder Kompositionsmodelle genutzt. Die Koordination von Grid-Diensten zeichnet sich damit häufig durch statische, zentralisierte und nicht automatisierte Ansätze aus [6, 98]. Foster und Kesselmann führen in [75] weitere Herausforderungen auf. Hierzu gehört der Bedarf an Skalierbarkeit und Evolution in Richtung zukünftiger Systeme und Dienste. Ebenso sind signifikante Weiterentwicklungen in der Toolunterstützung und in der Performance Analyse erforderlich, um komplexe Gridstrukturen überwachen zu können.

2.1.4.4 Peer to Peer Computing

Ein Jahrzehnt nach dem Durchbruch des Internet entstanden als Nachfolger *Client-Server* basierter Dienste wie Napster die ersten sogenannten *Peer-to-Peer* (*P2P*) Anwendungen. Diese zum Datentausch (engl. *file-sharing*) entworfenen Anwendungen besitzen keine Zentrale, sondern die Aktivitäten sind verteilt über die Teilnehmer, sogenannte *Peers*. Allein in Deutschland erreicht das Volumen von P2P-Anwendungen knapp 70% des gesamten Datenverkehrs (Abbildung 2.6). Auch in anderen Regionen nimmt das P2P-Volumen einen Spitzenplatz ein.

Tatsächlich ist der Einsatzzweck von P2P nicht auf Datentausch begrenzt. Nach Oestreich [179] umfasst die P2P Technologie das folgende Spektrum:

- Datentausch (Filesharing)
- Instant Messaging (IM, dt. Nachrichtenaustausch)
- Verteiltes Rechnen (Distributed Computing)
- (P2P-) Collaboration

Die Bedeutung im betrieblichen Umfeld ergibt sich aus vielfältigen Nutzungsmöglichkeiten wie Distribution von Software, Telekommunikation, Audio- und Videostreaming (Voice over P2P und P2PTV [38]). Diese Anwendungen versprechen ökonomische Nutzbarkeit und führen zu Investitionen von mehreren hundert Millionen US-Dollar. Die Vorteile bestehen in der enormen Skalierbarkeit. Jeder Teilnehmer bringt Speicherplatz, Rechenleistung und

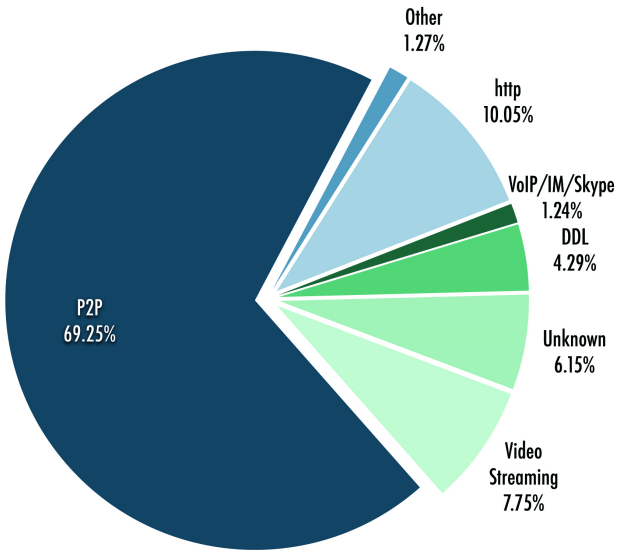


Abbildung 2.6: Aufteilung des Datenvolumens in Deutschland 2007 (Quelle: Internetstudie 2007, www.ipoque.com)

Bandbreite in das System mit ein und vergrößert damit die Ressourcen des Systems. Im Gegensatz zu Client-Server Anwendungen, deren Serverinfrastruktur den limitierenden Faktor darstellt, können P2P Netze und darauf basierende Anwendungen problemlos mit der Anzahl der Teilnehmer wachsen.

P2P Systeme verwenden ein eigenes Adressierungsschema und einen eigenen Namensraum außerhalb des *Domain Name Systems* (*DNS*) und Internet Protokoll basierter Adressen. Damit bilden P2P Anwendungen eigene Netze innerhalb des Internet. Die Suche von Teilnehmern gestaltet sich je nach Auslegung des Netzes. Existiert ein zentraler Server, der die Adressen der Teilnehmer vorhält, so kann dieser für Erstkontakte genutzt werden (Napster Modell). In völlig dezentralen P2P Ansätzen wird eine sogenannte *Distributed Hash Table* (*DHT*) genutzt. Hierbei wird jedem Teilnehmer ein Teil des Wertebereiches der Hashfunktion zugeteilt. Die Daten werden anhand der DHT auf die einzelnen Knoten verteilt. Kommen neue Peers hinzu, so teilen die benachbarten Knoten die Daten mit dem neuen Peer, verlässt ein Knoten das Netzwerk, so können andere sein Gebiet übernehmen. Der Speicherort von Daten wird anhand des Indexes durch Berechnung der Hash-Funktion ermittelt. Die Anfrage kann so von den Teilnehmern in Richtung des Index weitergeleitet werden. Dieses einfache und robuste Verfahren balanciert eine Datenmenge über eine Anzahl von Knoten. Das entstehende Netzwerk ist stark zusammenhängend und erlaubt effiziente Suche und ist gegenüber Topologieveränderungen robust gegen Clusterung der Teilnehmer. Ebenso bietet die starke Vernetzung verschiedene Routen zum gewünschten Ziel.

Zur Datenverbreitung und -verteilung existieren unter anderem sogenannte epidemische Algorithmen. Die Informtionsausbreitung geschieht hier ähnlich wie bei (Virus)Epidemien, indem die Informationen solange weitergegeben werden, bis sie allen bekannt sind. Die Kommunikation wird hierbei über zufällige Verbindungen verteilt und stellt eine Alternative zum Broadcast dar. Die Daten werden nicht nur an einer Stelle, sondern mehrfach abgelegt.

Herausforderungen für den Betrieb von P2P Netzen und darauf operierenden Anwendun-

gen sind vielfältig. Die Teilnehmer sind von ihren Ressourcen meist sehr heterogen in Bezug auf Rechenleistung, Bandbreite, Speicherplatz und auch hinsichtlich der eingesetzten Software. Der gemeinsame Nenner ist das verwendete Protokoll. Ohne zentralen Server entfällt hier die Möglichkeit umfassender Steuerung und Überwachung. Daten werden niemals 'garantiert' gespeichert sondern verschwinden unter Umständen ohne Nachfrage. Ebenso wie Daten ist keine Leistung eines P2P Netzes garantiert, denn Knoten können jederzeit das Netzwerk verlassen, ausfallen oder neu hinzukommen. Anwendungen müssen daher robust und flexibel auf Dienstleistungen zurückgreifen können. Ebenso stellt die massive Parallelität spezielle Anforderungen an Algorithmen zum verteilten Betrieb.

2.1.5 Adaptivität

Zustandsveränderungen aller Arten von Systemen sind essentiell, da hierdurch Reaktion auf vorherige externe sowie interne Ereignisse stattfinden. Hierfür existieren eine Vielzahl an Definitionen und Konzepten aus unterschiedlichsten Gebieten. Diese Arbeit beschränkt sich auf mathematisch-technische und evolutionäre Perspektiven. Die Begriffe *Adaptivität* und Anpassung werden synonym verwendet.

Zustandsänderungen können aufgrund äußerer Einflüsse, wie auch durch innere Prozesse hervorgerufen werden. In der Systemtheorie werden Systeme durch *Struktur* und *Verhalten* der Elemente charakterisiert. Dementsprechend beinhaltet Adaptivität ebenfalls beide Aspekte. Strukturelle Adaptivität kann zum Beispiel durch eine Änderung der Anzahl der Elemente oder deren Verbindungen definiert werden, wohingegen Verhaltensadaptivität die Funktionalität einzelner Elemente aber auch des Systemzustandes betrachtet. Aufgrund der wechselseitigen Abhängigkeiten von Struktur und Verhalten ist eine Abgrenzung nicht immer eindeutig. Gemeinsam ist allen Systemen, das Ursache-Wirkungsbeziehungen bereits dann nicht mehr erkennbar sind, wenn Interaktionen zwischen ihren Elementen als nichtlineare Rückkopplungen erfolgen (siehe hierzu Matthies [153]). Aus externer Sicht erweist sich die Analyse einzelner Systemelemente und deren Beitrag zur Gesamtleistung damit als schwierig bis unmöglich. Für das Verhalten des Gesamtsystems kann aus externer Sicht der Effekt von Zustandsänderungen beobachtet und beurteilt werden. Somit lässt sich auch eine Veränderung hinsichtlich definierter Kriterien erfassen. Damit kann Optimierung als eine spezielle Form von Adaptivität verstanden werden.

Kirn unterscheidet vier Bereiche [130] von Flexibilität, die im Zusammenhang mit Multiagententechnologie von besonderem Interesse sind. Ergänzt werden die vier Bereiche durch evolutionäre Anpassung:

- Die Perspektive der **Softwaretechnologie** fokussiert auf Engineeringprozesse und Entwurfsmuster [14]. Flexibilität ist der Bedarf an Wartbarkeit und die Anwendung in unterschiedlichen Anwendungsgebieten.

- Im Bereich der *künstlichen Intelligenz* [101] wird durch **maschinelles Lernen** Selbstadaptivität in (teil)autonomen intelligenten Systemen erreicht [237]. Adaptivität ist hierbei die Veränderung von Verhaltensmustern und Regeln als Reaktion auf äußere Reize.

- Die Perspektive der **Organisationstheorie** betrachtet künstliche Entitäten (Agenten) und deren Zusammenschluss zu Organisationen (Multiagentensysteme). Hierbei liegt

der Fokus auf konstitutionalisierenden Regeln zum Aufbau und Management von Organisationen. Dabei steht die Fähigkeit im Vordergrund, organisationale Strukturen als Reaktion auf Umweltveränderungen anzupassen.

- In der **Produktionstheorie** werden Systeme als Transformation von Ausgangsmaterialien in Endprodukte betrachtet. Flexibilität fokussiert hierbei auf die Transformationsfunktion, um möglichst kosteneffizient und nachfrageorientiert Endprodukte herzustellen.

- **Evolutionäre Anpassung** ist ein Prozess, bei dem Lebewesen in evolutionären Maßstäben besser an ihre Umgebung angepasst sind. Umgebung bedeutet in diesem Kontext die Gesamtheit externer Faktoren von geologischen Bedingungen über Feinde bis hin zu Artgenossen. Hierbei werden durch natürliche Selektion Individuen bevorzugt, deren Eigenschaften erhöhte Überlebenschancen bieten.

Gemeinsam ist allen vorgestellten Bereichen die Veränderung innerer Eigenschaften zum Zwecke der Veränderung des Gesamtverhaltens. Wesentlich ist die Rolle der Umgebung als auslösender Faktor.

2.1.6 Komplexe Adaptive Systeme

Komplexe Adaptive Systeme sind komplexe Systeme (engl. *Complex Adaptive System - CAS*), die sich ihrer Umwelt selbständig anpassen. Der Begriff wurde von Holland geprägt [245]:

> "A Complex Adaptive System (CAS) is a dynamic network of many agents (which may represent cells, species, individuals, firms, nations) acting in parallel, constantly acting and reacting to what the other agents are doing. The control of a CAS tends to be highly dispersed and decentralized. If there is to be any coherent behavior in the system, it has to arise from competition and cooperation among the agents themselves. The overall behavior of the system is the result of a huge number of decisions made every moment by many individual agents."

CAS sind als komplexe Systeme zu verstehen mit der Fähigkeit zur selbständigen Anpassung an ihre Umwelt. Diese Selbstadaptivität hat in den vergangenen Jahren zu enormem Forschungsinteresse geführt, um die darunterliegenden Mechanismen zu verstehen. Das Agentenparadigma und die agentenbasierte Computersimulationen haben zu neuen Bottom-Up Modellen beigetragen, die Emergenz, Dynamik und Adaptivität belegen. Ebenso sind biologische bzw. organische Ansätze vielversprechend für Adaptivitätsmechanismen. Organic Computing und Soft-Computing Ansätze, wie z.B. evolutionäre Verfahren gehören dazu. Gleichzeitig zeigen technische Systeme immer mehr Parallelen zu komplexen Systemen. Der Einsatz neuer Modelle, Verfahren und Mechanismen in technischen Systemen ist der nächste Schritt. Konsens herrscht darüber, dass moderne Informationssysteme von solchen Mechanismen im Hinblick auf ihre Robustheit und Adaptivität profitieren [8].

Um eine Einordnung und einen Vergleich heutiger Informationssysteme zu ermöglichen, werden in diesem Abschnitt die Eigenschaften komplexer adaptiver Systeme mit denen aktueller verteilter Systeme (siehe Abschnitt 2.1.4) verglichen. Es existieren ein Vielzahl ähnlicher Definitionen zum Begriff des komplexen adaptiven System. Tesfatsion gibt folgende Beschreibung [234]:

> "A complex adaptive system is a complex system that includes reactive units, i.e., units capable of exhibiting systematically different attributes in reaction to changed environmental conditions. "

Diese gibt gemeinsam mit Hollands Definition weiter oben einen Hinweis auf unterschiedliche Eigenschaften von CAS. Diese sind üblicherweise wie folgt definiert (siehe dazu [16, 113] und CSCS[6]):

Agenten basiert: Das System besteht aus individuellen Elementen.

Heterogenität: Die Agenten unterscheiden sich hinsichtlich ihrer Eigenschaften.

Dynamik: Die Eigenschaften der Agenten verändern sich mit der Zeit als Reaktion auf Umwelteinflüsse. Anpassung findet statt.

Feedback: Die Veränderungen der Agenten sind das Resultat von Feedback als Reaktion auf die Aktionen

Organisation: Agenten sind in Strukturen und Hierarchien organisiert. Die Struktur hat Einfluss auf die Entwicklung des Systems.

Emergenz: Neue Eigenschaften auf Systemebene entstehen (Makroebene), die auf Agentenebene nicht explizit definiert sind (Mikroebene).

Die Liste der Eigenschaften wird in Tabelle 2.1 mit den untersuchten verteilten Systemen Serviceorientierte Architekturen (SOA), Peer-to-Peer (P2P), Grid und Multiagentensystemen (MAS) abgeglichen.

Eigenschaft	SOA	P2P	Grid	MAS	CAS
Agenten	x	x	x	x	x
Heterogenität	x		x	x	x
Dynamik					x
Feedback		x	x	x	x
Organisation		x	x	x	x
Emergenz				x	x

Tabelle 2.1: Eigenschaften komplexer adaptiver Systeme und ausgewählter verteilter Systeme und Technologien im Vergleich

Tabelle 2.1 zeigt Ähnlichkeiten verteilter Systeme bzw. Technologien und CAS. Hierbei weisen besonders Multiagentensysteme bereits ähnliche Eigenschaften auf. Gerade aufgrund dieser Fähigkeiten finden sie häufig Einsatz bei Simulationen komplexer Systeme [234]. Trotz Flexibilität von Multiagentensystemen [130] ist die Anpassung und Bildung neuer Eigenschaften auf Umwelteinflüsse ein noch nicht ausreichend untersuchtes Problem.

[6]CSCS - Center for The Study of Complex Systems, http://www.cscs.umich.edu/old/complexity.html (Abruf vom 17.04.2009)

2.1.7 Zusammenfassung

Dieses Kapitel hat, basierend auf der Problemstellung, relevante verteilte Informationssysteme identifiziert. Diese Systeme sind serviceorientierte Architekturen, Grid und Peer-to-Peer Systeme. Hierbei wurde neben einem kurzen Überblick zu den Systemen auch Gemeinsamkeiten im Sinne der Systemwissenschaft aufgezeigt. Die Beschreibung lässt sich abstrakt auf Elemente und ihre Interaktionen zurückführen. Durch Definition komplexer Systeme wurde gezeigt, dass nicht immer eine vollständige Analyse verteilter Systeme möglich ist. Zentrale Steuerungs- und Optimierungsansätze und Top-Down Methoden sind damit limitiert und zeigen häufig nicht die gewünschte Wirkung. Darüber hinaus weisen natürliche CAS und zunehmend auch technische Systeme emergente Eigenschaften auf. Das Auftreten solcher Eigenschaften lässt sich nicht durch Dekomposition und Analyse einzelner Elemente erklären, sondern entsteht durch die Interaktionen der Elemente. Aktuelle zentrale Managementmethoden zur Steuerung von Adaptivtiät und Optimierung sind daher nicht ausreichend für den Einsatz in modernen Informationssystemen.

Die identifizierten Probleme in Abschnitt 1.2 weisen darauf hin, dass zentrale Steuerungs- und Optimierungsansätze und Top-Down Methoden zunehmend an ihre Grenzen stoßen, während natürliche CAS ein hohes Maß an Adaptivität zeigen. Daher stellt die Übertragung von Prinzipien natürlicher CAS auf technische Systeme einen vielversprechenden Ansatz dar.

2.2 Agentensysteme

Mit steigender Komplexität der zu modellierenden Problemstellungen hat die Bedeutung intelligenter Softwareagenten seit Ende der achtziger Jahre erheblich zugenommen [237, 255]. Insbesondere in Bereichen der Informationslogistik, in denen verteilte, dynamische Prozesse eine große Rolle spielen, hat sich die Verbreitung von Agententechnologie signifikant erhöht. Zu diesen Bereichen zählen z.B. Produktion, E-Commerce, Prozessüberwachung, Telekommunikation, Logistik, Geschäftsprozessmanagement [74]. Hierbei wird der impliziten Verteilung durch Modellierung unabhängiger Software-Entitäten Rechnung getragen, die entweder autonom oder im Auftrag eines anderen (Agenten) ein bestimmtes Ziel verfolgen [237].

Dieses Kapitel wählt den Bottom-Up Ansatz und betrachtet die Agententechnologie ausgehend vom Begriff des Agenten in Kapitel 2.2.1. Kapitel 2.2.2 betrachtet die Umgebung und Kapitel 2.2.3 zeigt unterschiedliche Agentenarchitekturen. Danach folgt in Kapitel 2.2.4 eine Charakterisierung sogenannter Multiagentensysteme (in der Literatur auch als Mehragentensysteme bezeichnet). Mechanismen der Kommunikation und Interaktion beschreibt Kapitel 2.2.5. Einen ökonomischen Ansatz beschreibt Kapitel 2.2.6 und weiterführende Ansätze sind in Kapitel 2.2.7 zusammengefasst.

2.2.1 Definition Agent

In der Literatur existieren viele Versuche, den Begriff *Agent* zu definieren. Frühe Definitionen sind beispielsweise noch stark an programmiersprachliche Konzepte angelehnt, wie die Definition von Genesereth und Ketchpel [83] zeigt:

> "[Agents] communicate with their peers by exchanging messages in an expressive agent control language. While Agents can be as simple as subroutines, typically they are larger entities with some sort of persistent control."

Sobald ein Problem kooperativ gelöst wird, muss direkte oder indirekte Interaktion erfolgen. Als Beispiele direkter Interaktion sei Kommunikation genannt, während indirekte Interaktion, sogenannte *Stigmergie* (engl. *Stigmergy* [29, 102]) über die Umgebung erfolgt. In der Agententechnologie steht der Nachrichtenaustausch mittels einer dedizierten Agentensprache im Vordergrund. Anstelle klassischer Funktions- oder Methodenaufrufe werden Informationen per Nachrichten versendet. Nachrichtenaustausch kann als Grundlage für eine erhöhte Flexibilität angesehen werden, da ein System aus Agenten seine interne Struktur zur Laufzeit ändern und somit dynamisch auf Umweltänderungen reagieren kann. Nachrichtenaustausch ist auch nach Entfernung bzw. hinzufügen von Agenten weiterhin möglich, ein Funktionsaufruf nach Entfernung der Funktion dagegen nicht. Eine solche erhöhte Flexibilität "ermöglicht den Einsatz [...] in offenen Umgebungen"[237]. Spätere Definitionen sind allgemeiner und verstehen Agenten als Softwareentitäten die ihre Umwelt wahrnehmen und verändern können [247, 254].

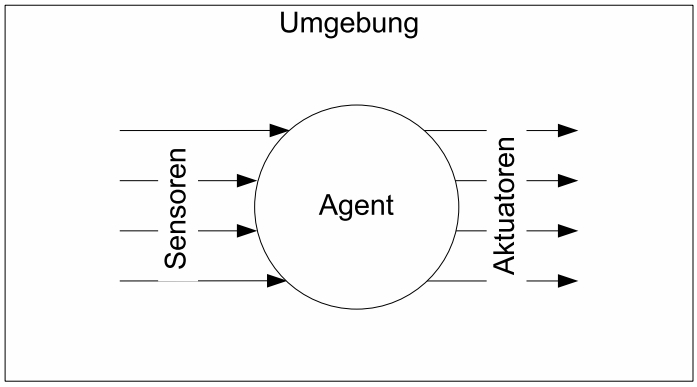


Abbildung 2.7: Agent in Umgebung: mittels Sensoren und Effektoren/Aktuatoren wird die Umgebung wahrgenommen und verändert

Abbildung 2.7 zeigt die abstrakte Sichtweise eines Agenten, der sich in einer Umgebung befindet. Mittels Sensoren wird die Umgebung wahrgenommen und durch *Effektoren* bzw. *Aktuatoren* verändert. Die Wahrnehmung und Beeinflußung der Umgebung bilden einen Zyklus, da eine geänderte Umgebung wieder Reaktionen des Agenten hervorrufen kann. Als Beispiel sei hier ein Agent genannt, der den Bestand eines Warenlagers überwacht. Unterschreitet der Bestand einen bestimmten Wert, so wird der Agent eine Bestellung auslösen. Sobald die Bestellung eingetroffen ist, hat sich die Umgebung des Agenten dahingehend verändert, dass eine erneute Bestellung für eine gewisse Zeitspanne nicht mehr notwendig ist.

Eine von weiten Teilen der Agentenkommunity akzeptierte, hinreichend allgemeine Definition von Agent ist in einem vielzitierten Aufsatz von Michael Wooldridge und Nicholas Jennings [255] erschienen. Die sogenannte "Weak Notion of Agency"besitzt vier grundlegende Eigenschaften:

- **Autonomie (autonomy)**: Agenten agieren ohne direkte Intervention von Menschen oder anderen Entitäten und haben mindestens partielle Kontrolle über ihre Aktionen und ihren Zustand.
- **Soziale Fähigkeiten (social ability)**: Agenten interagieren miteinander (oder auch mit Menschen) über eine Art Kommunikationssprache

- **Reaktivität (reactivity)**: Agenten nehmen ihre Umgebung wahr und reagieren zeitnah auf Veränderungen
- **Proaktivität (pro-activeness)**: Agenten reagieren nicht nur auf Veränderungen der Umgebung, sondern verfolgen aktiv eigene Ziele

Aufbauend auf dieser Grundlage existieren weitergehende Definitionen (Stronger Notion of Agency), die zusätzliche Eigenschaften beinhalten. Hierzu zählen Begriffe wie Lernfähigkeit, Mobilität, mentale Zustände oder emotionale Fähigkeiten. Als Beispiel sei hier der Belief, Desire, Intention (Überzeugungen, Ziele, Absichten, BDI) Ansatz genannt, der 1987 von Bratman [34] vorgestellt wurde. Eine ausführlichere Beschreibung dieses Ansatzes findet sich im Kapitel 2.2.3.2 - Deliberative Agentenarchitekturen. Anwendung findet dieser Ansatz in vielen Implementierungen, zum Beispiel die Jadex [193] Erweiterung des Agentensystems JADE ([20, 237]).

Der Begriff des Agenten wird nach Wooldridge und Jennings [256] im Rahmen dieser Arbeit folgendermaßen definiert:

> "An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible autonomous action in that environment in order to meet its design objectives."

Ein Agent ist ein gekapseltes Computersystem, welches sich in einer Umgebung befindet und flexibel und autonom Aktionen ausführt, um seine Ziele zu erreichen.

2.2.2 Agentenumgebung

Ausgehend von der Definition eines Agenten gibt es immer auch eine Umgebung (siehe Abbildung 2.7) in welcher sich dieser befindet. Russell und Norvig [205] geben dazu folgende Eigenschaftsdimensionen von Umgebungen an:

- **Zugreifbar vs. nicht zugreifbar** (accessible vs. inaccessible): Eine zugreifbare Umgebung kann vom Agent vollständig, fehlerfrei und aktuell wahrgenommen werden. In diesem Sinne ist die reale Welt und damit auch die meisten Umgebungen für Informationssysteme nicht vollständig zugreifbar.
- **Deterministisch vs. nicht deterministisch** (deterministic vs. non-deterministic): Eine deterministische Umgebung reagiert auf eine Veränderung (des Agenten) mit einer einzelnen, vorhersehbaren Aktion. Es ist absolut garantiert, dass diese Aktion eintritt.
- **Statisch vs. dynamisch** (static vs. dynamic): Eine statische Umgebung bleibt unverändert mit Ausnahme der vom Agenten durchgeführten Aktionen. Im Gegensatz dazu ändert sich eine dynamische Umgebung unabhängig von den Aktionen eines einzelnen Agenten, etwa weil andere Enititäẗen (z.B. Agenten) parallel darin operieren.
- **Diskret vs. kontinuierlich** (discrete vs. continuous): In diskreten Umgebungen existiert eine festgelegte, endliche Anzahl von Aktionen und Wahrnehmungen.

Agenten, die sich in realen Umgebungen oder in Softwareumgebungen befinden, sind immer mit den oben beschriebenen Charakteristika konfrontiert. Ein idealer Agent trifft immer

die "richtigen"Entscheidungen, die jedoch maßgeblich von der Qualität vorhandener Informationen abhängen.

Je zugreifbarer eine Umgebung ist, desto einfacher lassen sich Agenten konstruieren, die in ihr agieren. Ist ein Teil der notwendigen Informationen nicht zugreifbar, muss auf Basis unsicherer Information entschieden werden.

Wie bereits angedeutet, ist eine Umgebung, in welcher sich mehrere Agenten befinden, nach obiger Definition bereits nicht-deterministisch. Damit stellt sich aus Sicht eines einzelnen Agenten die Umgebung als die Menge aller anderen Agenten und der physikalischen bzw. der Softwareumgebung dar. Nichtdeterminismus bedeutet zum Einen, dass Agenten limitierten Einfluss auf ihre Umgebung haben. So kann Agent `A` einen anderen Agenten `B` (und damit einen Teil seiner Umgebung) nicht kontrollieren, was dazu führen kann, dass `B` aus Sicht von `A` nichtdeterministisch agiert. Auf der anderen Seite können Aktionen fehlschlagen und somit nicht zum gewünschten Umgebungszustand führen. Damit müssen in der Praxis nahezu alle Umgebungen als nichtdeterministisch angesehen werden [205, 254].

Diese Aussage führt zu der Schlussfolgerung, dass eine nichtdeterministische Umgebung immer auch eine dynamische Umgebung ist. Aus Sicht des Agenten muss davon ausgegangen werden, dass zwischen zwei Zeitpunkten t_0 und t_1 ohne Aktion die Umgebung einen anderen Zustand annehmen kann. Zusätzlich können andere Entitäten die Aktionen des Agenten beeinflussen. Damit muss damit gerechnet werden, dass Informationen zum Zeitpunkt des Eintreffens bereits veraltet sind und geplante Aktionen nicht zum gewünschten Ergebnis führen können, da zwischen Informationsaufnahme und vollständigem Plan bereits Zeit verstrichen ist.

Diskrete Umgebungen (z.B. Schachspiel) garantieren eine endliche Anzahl von Zuständen, wohingegen kontinuierliche Umgebungen (z.B. Navigation) unendlich viele Zustände annehmen können. Damit ist es für diskrete Umgebungen möglich, alle Zustände und optimale Aktionen dazu aufzuzählen. Dies ist jedoch in der realen Welt nicht möglich.

Damit ist eine nicht zugreifbare, nichtdeterministische, dynamische und kontinuierliche Umgebung die komplexeste Klasse. Hierfür wurde von Hewitt 1986 der Begriff *offene Umgebung* verwendet [108]. Die reale Welt fällt damit unter die komplexeste Klasse an Umgebungen. Jedoch ist bereits eine Softwareumgebung mit mehr als einem Agent aufgrund obiger Klassifikation per se nicht deterministisch und nicht zugreifbar aus Sicht eines Agenten. Daher wird in dieser Arbeit eine offene Umgebung vorausgesetzt.

2.2.3 Agentenarchitekturen

Dieser Abschnitt beschäftigt sich mit der internen Struktur von Agenten, um aus technischer Sicht eine Basis zur Umsetzung der Agenteneigenschaften (siehe dazu Abschnitt 2.2.1) bereitzustellen. In der Literatur existieren eine Vielzahl unterschiedlicher Ansätze, die zumeist eng an eine Anwendungsdomäne oder ein konkretes Problem gekoppelt sind. Müller [171] beschreibt vier grundlegende anwendungsunabhängige Schichten-Architekturarten, die andere Klassifikationen [132, 247, 258] einschließen. Diese werden im Folgenden kurz erläutert.

2.2.3.1 Reaktive Architektur

Reaktive Agenten führen Aktionen auf bestimmte Reize in ihrer Umwelt aus. Dafür werden in der strikten Umsetzung der reaktiven Architektur weder ein internes Modell der Welt noch eine Art Inferenz benötigt. Ein intelligentes Verhalten im Sinne der Weak Notion of Agency

(siehe Kapitel 2.2.1) kann trotzdem erreicht werden, wenn eine große Zahl solcher Agenten gemeinsam Aufgaben lösen. Hierzu führt Brooks [36] an, das intelligentes Verhalten ohne explizite Repräsentation und ohne Inferenz als emergente Eigenschaft komplexer Systeme entstehen kann. Ein Beispiel ist das in NetLogo™[251] entwickelte Modell in [199]: Eine Ameisenkolonie baut die Futterquellen im Allgemeinen in der Reihenfolge ihrer Entfernung zur Kolonie ab. Dafür nutzen die Ameisenagenten simple, vorgefertigte Verhaltensweisen, die im Zusammenspiel vieler Ameisen und Pheromone zum emergenten Gesamtverhalten führen.

Wooldridge [254] nennt folgende positive Eigenschaften: Einfachheit, ökonomisches Verhalten (im Sinne von eingesetzte Mittel vs. Ergebnis) und Robustheit gegenüber Fehlern. Zu den negativen Eigenschaften zählen unter anderem die Zugreifbarkeit ausreichender Informationen, da auf Basis aktueller Reize reagiert wird. Weiterhin können Informationen der Vergangenheit nur eingeschränkt genutzt werden (short-term view). Ebenfalls eingeschränkt sind die Möglichkeiten lernender reaktiver Agenten. Ein klassisches Softwareengineering Vorgehen zur Erstellung adäquater Methoden existiert aktuell nicht, da eine große Zahl von Agenten in Umgebungen enorme Interaktionskomplexität und Dynamik aufweisen. Dieses sogenannte emergente Verhalten ist per Definition im Voraus sehr schwer - wenn überhaupt - zu modellieren bzw. zu formalisieren.

2.2.3.2 Deliberative Architektur

Traditionelle Ansätze der Künstlichen Intelligenz gehen davon aus, dass intelligentes Verhalten auf Grundlage einer symbolischen Repräsentation der Umgebung und des gewünschten Verhaltens aufbaut. In diesem Sinne wurden deliberative Agenten zunächst als gekapselte Expertensysteme aufgefasst, deren Wissensbasis durch Inferenz zum Lösen von Theoremen genutzt wurde [82, 254]. Später wurden Konzepte aus unterschiedlichen Bereichen, wie z.B. der Verhaltensforschung hinzugefügt. Dazu zählen unter anderem Ziele und Pläne. Der *BDI* Ansatz ist ein weit verbreitetes deliberatives Agentenmodell [34], der in der Agentenwelt nach Weiss [247] wie folgt umgesetzt werden kann:

- **Beliefs (Wissen)**: Das aktuelle Weltmodell des Agenten.
- **Desires (Ziele)**: Grundlegende verhaltensbeeinflussende Faktoren. Ziele stellen erstrebenswerte Zustände dar.
- **Intentions (Absichten)**: Aus dem Weltmodell wird anhand eines gewählten Zieles (autonom) ein Plan generiert, der die Absichten des Agenten beinhaltet.

Der Vorteil reaktiver gegenüber deliberativen Agentenarchitekturen ist ein geringerer Planungsaufwand, um zu einer zielführenden Aktion oder Aktionsfolge zu kommen. Die in reaktiven Agenten vorgefertigten Reiz-Reaktionsmuster erlauben augenscheinlich schnellere Aktionen bei gegebenem Input. Daher ergeben sich bei gleichartigen Herausforderungen zumeist eine höhere Effizienz [237]. Demgegenüber liegt der Vorteil von deliberativen Agenten darin, dass neue Problemlösungsmöglichkeiten bei geeigneter Wissensbasis selbständig erkannt und verfolgt werden können. Daher sind reaktive Agenten an das per Entwurf und Implementierung antizipierte Verhalten gebunden und können hier effizient reagieren.

2.2.3.3 Interagierende Agenten

Ein interagierender Agent (*interacting agent*) stellt eine Schnittstelle dar. Die Interaktion kann sowohl mit einem menschlichen Anwender, zwischen Agenten bzw. innerhalb von Agentensystemen oder auch mit Fremd bzw. Legacy-Systemen erfolgen. Wie in [104, 135, 237] beschrieben, enthalten interagierende Agenten dennoch eine steuernde Intelligenz, da Informationen über die Ziele des Nutzers oder die Domäne bekannt sein müssen, um z.B. Vorschläge unterbreiten zu können. Ein persönlicher Assistent kann durch einen Agenten bereitgestellt werden und für den Nutzer zum Einen vereinfachte Interaktionen anbieten und zum Anderen den Nutzer mit anwendungsspezifischen Informationen versorgen, wie z.B. in GAIA Ansatz [135] beschrieben. Als Repräsentant für die Interaktionen zwischen Agenten sei der MEKKA [104] Ansatz genannt. Hier erfolgt eine Dreiteilung des Agenten in Kommunikator (Nachrichtenverarbeitung), Kopf (Kooperation) und Körper (Problemlösen).

2.2.3.4 Hybride Architekturen

Die bisher aufgeführten Architekturtypen stellen die Extrempunkte im Raum der Agentenarchitekturen dar. Dazwischen existieren eine Vielzahl von Mischlösungen, welche die Vorteile mehrerer Ansätze in sich vereinen [171]. Verschiedene Ansätze kombinieren reaktive und deliberative Komponenten als verschiedene Subsysteme eines Agenten um einerseits auf Reize zeitnah reagieren zu können und andererseits im Hintergrund langfristige Pläne aufgrund der Ziele zu generieren. Die einzelnen Komponenten sind in InteRRaP [172] und TouringMachines [67] als Schichten angeordnet, wobei reaktive Schichten Vorrang besitzen. Das von Smith and Taylor vorgestellte Egglets Framework [221] und die Arbeit von Babanov et al. [8] nutzen reaktive Agenten, die über einen evolutionären Mechanismus ihre Reiz-Reaktionsmuster verändern.

Die vorliegende Arbeit schlägt insofern einen *hybriden Ansatz* vor, als dass der einzelne Agent reaktiv und damit einfacher als ein vergleichbarer deliberativer Agent aufgebaut ist. Jedoch erfolgt eine Änderung der Reiz-Reaktionsmuster für jeden Agenten durch einen übergeordneten evolutionären Prozess.

2.2.4 Multiagentensysteme

In der Literatur gibt es eine Reihe von Definitionen für den Begriff *Multiagentensystem* (*MAS*) [121, 132, 145, 254], die unter folgender textueller Definition subsumiert werden können:

Definition 1 (Multiagentensystem)**.** *Systeme aus mehreren miteinander interagierenden Agenten, die sich in einer Umgebung befinden, werden als Multiagentensysteme bezeichnet.*

Die Anzahl der Agenten kann stark variieren je nach Problem und Typ der eingesetzten Agenten und Agentenframeworks. Die Bandbreite der Anwendungsfälle reicht von einem Agenten (z.B. Heizungssteuerung [247]), einige (Supply-Chain Management [64, 121, 252]) bis hin zu tausenden (Dynamic Pricing in der Informationsökonomie [128]) oder mehreren hunderttausend [151] Agenten zur Simulation eines look-up Systems für Peer-to-Peer Netzwerke. Die Agentenpopulation kann heterogen sein [121], wobei unterschiedliche organisatorische Rollen durch verschiedene Agententypen verkörpert werden. Das Rollenmodell, motiviert durch die Sozialwissenschaften und Organisationstheorie, ist Voraussetzung für institutionalisierte Koordination durch Strukturierung und unterstützt flexibles Systemverhalten [132]. Einen

organisationszentrierten Ansatz zur Gestaltung findet sich bei Kirn [129]. Eine beispielhafte Struktur eines Multiagentensystems ist in Abbildung 2.8 zu sehen.

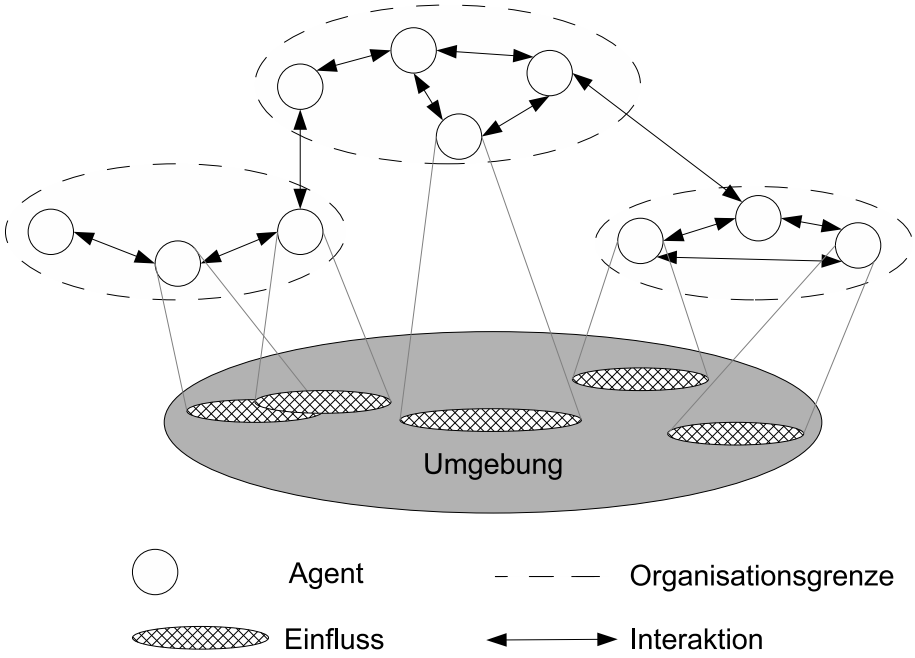


Abbildung 2.8: Beispiel für die Struktur eines Multiagentensystems, angelehnt an [254]

Wie Abbildung 2.8 zeigt, hat ein Agent unvollständige Informationen, limitierte Möglichkeiten und damit limitierten Einfluss auf seine Umgebung. Dazu zählen aus Sicht eines Agenten auch alle anderen Agenten. Durch die über alle Agenten verteilte Information existiert keine globale Systemkontrolle [121]. Daher nehmen Kommunikations- und Interaktionsmechanismen eine Schlüsselrolle für jegliches operatives Verhalten ein, wie z.B. Informationensaustausch, Koordination von Aktionen, Lösen von Konflikten oder um organisatorische Prozesse zu etablieren und zu optimieren. Diese Mechanismen werden in nachfolgendem Abschnitt 2.2.5 einführend behandelt.
Ferber [66] definiert ein Multiagentensystem bestehend aus sechs Bestandteilen wie folgt:

- Eine Umgebung E,
- Eine Menge von Objekten O in E,
- Eine Menge von Agenten A als Teilmenge von $O \supseteq A$,
- Eine Menge von Beziehungen R als Definition der Beziehungen zwischen den Objekten O,
- Eine Menge von Operationen Op, mit deren Hilfe Agenten andere Objekte aus O wahrnehmen und manipulieren können,
- Eine Menge von universalen Gesetzen, welche die Reaktionen der Umgebung auf Agentenoperationen definiert.

Der Aspekt der Sichtbarkeit wird in der sogenannten Multiagenten $\mathcal{VSK} - Logik$[7] von Wooldridge [257] durch die Sichtbarkeitsfunktion *vis* formalisiert und von Timm aufgegriffen [237]. Hiermit ist eine Partitionierung der Agenten und der Umgebung formal realisierbar. Damit lassen sich durch Hinzunahme von Beziehungen Strukturen zwischen den Agenten bzw. Mengen von Agenten realisieren. Nach [132] ist die Struktur und damit die Bildung von organisationalen Einheiten ein Rahmen, um Koordination mit unterschiedlichen Rollen und Fähigkeiten in flexiblen und skalierbaren Organisationen zu ermöglichen. Ein Unternehmen innerhalb einer Wertschöpfungskette ist z.B. durch den Vertrieb und damit durch Agenten in der Vertriebs-Rolle nach außen repräsentiert. Eine Charakterisierung von Strukturen in Multiagentensystemen in Anlehnung an organisationale Strukturen besteht nach [132] aus drei Dimensionen:

- **Fähigkeiten (capabilities)** Auf Grundlage heterogener Agenten entsteht die Fähigkeit zum Lösen von Problemen, die von einem einzelnen Agent nicht lösbar sind. Im Unterschied zum Parallelen Rechnen [196] oder Grid-Computing [78, 122, 201] wird damit in erster Linie nicht die Skalierbarkeit, sondern Flexibilität [130] adressiert. Damit wird das Management zur Bereitstellung von heterogenen Ressourcen für breite Anwendungsdomänen zu einem wichtigen Erfolgsfaktor. Forschungsgebiete in diesem Bereich sind z.B. Kooperatives verteiltes Planen [55, 57], Ressourcen Management [239].

- **Dauer (duration)** Ein weiterer Aspekt ist die Betrachtung der Dauer des Bestehens von Strukturen. Das Bestehen kann über die gesamte Laufzeit eines MAS anhalten oder auch für jedes Problem oder gar jede Transaktion neu assembliert werden. Für dauerhafte Strukturen lassen sich bereits während des Entwurfs der Kommunikations- und Koordinationsmechanismen finden und statisch implementieren. Dies erlaubt effiziente Entscheidungsfindung mit wenig Overhead für vordefinierte Probleme. Der Nachteil ist hier jedoch geringere Flexibilität des Gesamtsystems wenn sich die Umgebung ändert. Hier sind dynamische Strukturen im Vorteil, die adäquate und dynamische Reorganisationen zur Laufzeit ermöglichen. Jedoch sinkt damit die Effizienz, da ein gewisser Anteil der Ressourcen für Umstrukturierungen verwendet wird.

- **Entscheidungsfindung (decision-making)** Ähnlich wie bei der Dauer von Strukturen lässt sich ebenso die Entscheidungsfindung von statisch bis dynamisch umsetzen [206]. Auch hier entscheidet die Balance von gewünschter Flexibilität und Effizienz der zu realisierenden Lösung.

Obige Aufstellung lässt jedoch die Frage der Dimensionierung des Multiagentensystems offen, das heißt wie viele Agenten insgesamt beziehungsweise pro Rolle notwendig sind.

Die Infrastruktur bildet eine wichtige Basis, um Dienstabfragen und Kommunikation zu ermöglichen. So verursacht die Nutzung von sogenannten *Yellow Pages*[8] einen Flaschenhals der zutage tritt, wenn die Anzahl der Agenten, die Kommunikation und die Informationen im System zunimmt [151]. Ein Ansatz diesen Flaschenhals zu umgehen beschreibt das DIET Projekt [151]. Hier werden keine Yellow Pages genutzt, sondern Agenten interagieren auf einer Peer-to-Peer Basis und werden von der Infrastruktur auf Anfrage zufällig verbunden. Alle

[7] $\mathcal{VSK}$-Logik ist eine multimodal Logik mit den Modalitäten $\mathcal{V}$, $\mathcal{S}$ und $\mathcal{K}$, wobei $\mathcal{V}_\varphi$ für Sichtbarkeit (engl. visibility) der Information φ im aktuellen Zustand steht. $\mathcal{S}_\varphi$ bedeutet, der Agent nimmt φ wahr (engl. see) und $\mathcal{K}_\varphi$ bedeutet, der Agent weiss φ (engl. know).

[8] z.B. in JADE [20], ADEPT [120], OAA [152], ZEUS [177] oder auch RETSINA [229]

Ansätze nutzen dennoch eine zentralisierte Infrastruktur, um mindestens den initialen Kontakt herzustellen (Agentenframework) bzw. Systemsoftware und Hardware, um Nachrichten zu verschicken oder Rechenleistung, Bandbreite und Speicherplatz zu nutzen. Ebenso sind allen Agentenframeworks bzw. untersuchten Methodologien[9] die Begriffe Agent und Nachrichten gemeinsam. Implizit kann die Infrastruktur eines Agentenframeworks als Umgebung angesehen werden. Die von Ferber zusätzlich definierten Objekte ohne Agenten ($O \setminus A$) und Aktionen werden explizit lediglich von Gaia und Message verwendet. Organisationale Strukturen und Relationen auf Objekte werden lediglich in Gaia rudimentär unterstützt. Vielmehr werden organisationale Strukturen implizit durch die Interaktionen zwischen den Agenten aufgebaut und Aktionen sind durch Nachrichten an andere Agenten zu sehen. Universale Gesetze als Reaktion auf Aktionen finden in keinem untersuchten Framework bzw. Methodologie Anwendung. Weiterführende Ansätze werden in Abschnitt Grid Computing 2.1.4.3 beschrieben.

2.2.5 Agentenkommunikation und -interaktion

Als einer der wesentlichsten Bestandteile von dem was Multiagentensysteme verkörpern gilt die Kommunikation und darauf aufbauende Interaktionsmechanismen. Eine dazu passende Metapher liefert Michael Luck in [145]: “Computing as Interaction”. Abbildung 2.9 zeigt eine Übersicht von Koordinationsmechanismen. Dieser Abschnitt gibt einen kurzen Abriss über wichtige Kommunikations- und Interaktionsmechanismen im Agentenbereich nach [116].

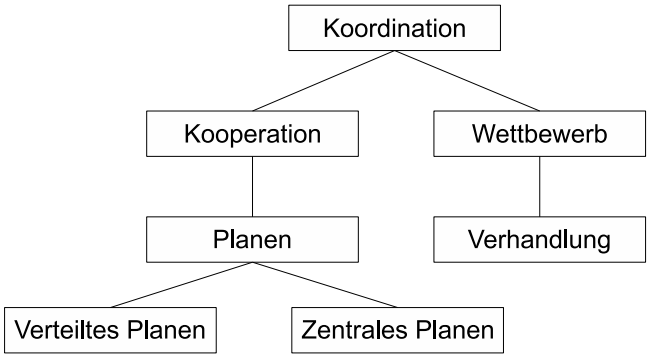


Abbildung 2.9: Koordinationsmechanismen nach [116]

2.2.5.1 Kommunikation

Im Allgemeinen - wie auch in dieser Arbeit - wird nachrichtenbasierte Kommunikation vorausgesetzt, wobei jedoch auch andere Formen möglich sind. Stigmergie [29] ist eine Form indirekter Koordination zwischen Agenten unter Nutzung der Umgebung, etwa in Form von Spuren oder Pheromonen. Diese Ansätze werden im Rahmen der vorliegenden Arbeit jedoch nicht weiter betrachtet. Kommunikation im Agentenbereich unterscheidet sich vom prozeduralen und objektorientierten Methodenaufruf durch den (asynchronen) Informationsaustausch in Form von sogenannten *Sprechakten*. Die Grundlagen der heute genutzten *Sprechakttheorie* gehen auf John Searle [217] zurück. Nach diesem Verständnis lässt sich jeder Sprechakt zerlegen in weitere Bestandteile:

[9] Gaia [259], Jade [20], Diet [151], Prometheus [186], Message [139]

1. die eigentliche Äußerung selbst,
2. die vollzogene Handlung, indem man etwas sagt und
3. die Konsequenz der Handlung beim Empfänger (Wirkung).

Auf Basis der Sprechakttheorie existieren verschiedene Agentenkommunikationssprachen (ACL - Agent Communication Language) zum standardisierten Informationsaustausch. Diese spezifizieren unterschiedliche Aspekte, welche für eine gemeinsame Kommunikation unerlässlich sind. Dazu zählen unter anderem Sprache, Protokolle und Nachrichtenformate. Beispiele für solche Agentenkommunikationssprachen sind z.B. *FIPA ACL* (Foundation for Intelligent Physical Agents) [24] oder das ältere *KQML* (Knowledge Query and Manipulation Language) [68]. Diese Standards legen den Nachrichtenaustausch auf Basis von Nachrichtenaufbau und Protokollen fest. Beide Standards unterstützen die optionale Verwendung von *Ontologien*. Eine Ontologie ist eine semantische Spezifikation zur Beschreibung von Begriffen, deren Eigenschaften und Beziehungen zueinander in einer formalen Art [100]. Im Rahmen dieser Arbeit sind die vorgestellten Standards insofern wichtig, als dass diese eine einheitliche und akzeptierte Kommunikationsbasis schaffen, auf deren Konsens weitere Konzepte und Mechanismen aufbauen.

2.2.5.2 Kooperation

Kooperation erreicht Koordination über ein gleiches Ziel aller Teilnehmer. Kooperatives verteiltes Problemlösen (Cooperative Distributed Problem Solving - CDPS) beschäftigt sich mit der Analyse von Ansätzen um in lose gekoppelten Systemen gemeinsam Probleme zu lösen, die durch einen Agent nicht zu lösen sind [57]. Ansätze für kooperatives Problemlösen sind zum Beispiel Aufgabenzerlegung und -verteilung (task decomposition and distribution) [116].

Ein Beispiel kooperativen Problemlösens ist das sogenannte *Contract Net Protokoll* [51, 222, 247]. Es ist primär zur Steuerung von Güter- und Dienstleistungsausschreibungen gedacht. In einem Agentensystem kann jeder Agent die beiden Rollen *Manager* und Contractor einnehmen. Das Protokoll läuft in drei Schritten ab. Der Manager schreibt einen Auftrag aus (z.B. per Multicast oder Broadcast), Contractoren evaluieren die Ausschreibungen und antworten ggf. mit einem Gebot. Aus der Menge der Gebote wählt der Manager einen geeigneten Contractor aus. Insgesamt stellt das Contract Net Protokoll einen effizienten Weg für direkt verbundene Agenten dar. Diese Fähigkeit kann Kooperationen in einigen Fällen vereinfachen und hierdurch den Prozess der Ausschreibung effizient gestalten.

2.2.5.3 Wettbewerb

Insbesondere im interorganisationalen Kontext kann davon ausgegangen werden, dass im Gegensatz zur Kooperation ein gleiches Ziel und damit einhergehender intrinsischer Kooperation teilnehmender Entitäten eher die Ausnahme darstellt. Sobald Agenten verschiedene Ziele verfolgen, befinden sich miteinander im *Wettbewerb*. Eine häufige Interaktionsform bei unterschiedlichen Zielen ist die *Verhandlung* [51, 247]. Zunächst werden die jeweiligen (konfligierenden) Positionen kommuniziert und dann durch schrittweises Entgegenkommen eine Vereinbarung erzielt. Verhandlungsmechanismen, egal ob Umgebungszentriert oder Agentenzentriert, sollten nach Weiss [247] idealerweise folgende Kriterien erfüllen:

Effizienz Es sollten keine Ressourcen beim Einigungsprozess verschwendet werden (z.B. übermäßiger Nachrichtenaustausch),

Stabilität Kein Teilnehmer sollte den Anreiz bekommen, von getroffenen Vereinbarungen abzuweichen,

Einfachheit Der Verhandlungsmechanismus sollte niedrige technische Anforderungen haben,

Verteiltheit Der Verhandlungsmechanismus sollte keiner zentralen Entscheidungsinstanz bedürfen,

Symmetrie Einzelne Teilnehmer sollten keine Vorteile gegenüber anderen haben,

Anreizkompatibilität Teilnehmer sollen ihre wahren Präferenzen offenbaren.

Insbesondere der letzte Punkt der Anreizkompatibilität ist Gegenstand aktiver Forschung. Hierbei soll der Verhandlungsmechanismus die wahren Präferenzen der Teilnehmer im Sinne der *Mechanismus-Design Theorie* [13] offenbaren und daher eine Entscheidung aufgrund des dem Versteigerungsgut intrinsischen Wertes ergeben. Auktionsteilnehmer sollen sich nicht strategisch verhalten, sondern aufgrund des Designs der Auktion ihre wahren Präferenzen einbringen. Zur Umsetzung der Anreizkompatibilität beschrieb William Vickrey 1961 die sogenannte *Vickrey Aktion* [243]. Hierbei bekommt der Bieter mit dem höchsten Preis die angebotene Ware zum Preis des zweithöchsten Bieters. Diese Form der Auktion wird unter anderem auf der Internetplattform *Ebay*[10] angewendet.

Nach Weiss [247] sind die bisher betrachteten Interaktionsansätze besonders für eine geringe Anzahl von Agenten geeignet, da die Kommunikationsprotokolle hierfür ausgelegt sind. Eine englische Auktion mit vielen Teilnehmern lässt sich elektronisch nur dann effizient realisieren, wenn die Kommunikationskosten niedrig sind [241] und die Teilnehmer in direktem Kontakt mit dem Auktionator stehen. Andere Mechanismen sind erforderlich, wenn die Anzahl der Agenten steigt oder unbekannt ist. Ähnlich verhält es sich mit den Kosten für die Kommunikation.

2.2.6 Agentenbasierte ökonomische Ansätze

Marktbasierte Verfahren sind ein aussichtsreicher Ansatz und aktives Forschungsgebiet, um die Beschränkungen der Koordinationsformen der vorherigen Kapitel zu erweitern. Elektronische Marktplätze stellen eine fruchtbare Verbindung ökonomischer Modelle mit Computertechnologie dar. Dieser Ansatz wird als *Agent-based computational Economics* (*ACE*) bezeichnet [234]. In der ökonomischen Theorie sind die selbstorganisierenden Fähigkeiten dezentraler Marktmechanismen von Interesse. Simulationen basieren auf ökonomischen Agenten und deren Interaktionen miteinander sowie mit der Umgebung. Preisvorgaben von außen sind nicht erlaubt, so dass allein die vorgegebenen Interaktionen Einfluss auf die Preise haben. Das Erreichen von Gleichgewichtspreisen in solchen Umgebungen kann dabei als emergent (vgl. hierzu den *AVALANCHE* Ansatz [64]) betrachtet werden. ACE wird von Tesfatsion als Untersuchungsansatz für ökonomische Systeme in Kontext komplexer adaptiver Systeme betrachtet [234].

Der ACE Modellierer spezifiziert den initialen Status des Systems, bestehend aus den Daten und dem Verhalten der Agenten und der jeweiligen Zugreifbarkeit durch andere Agenten.

[10] siehe www.ebay.de

Agenten repräsentieren alle Rollen in der Modellwelt, z.B. Organisationen, Kunden, Markt oder die Welt. Die Verhaltensweisen können unterschiedliche Protokolle und Kommunikationsstrukturen für Interaktionen sein, z.B. Vertrauensbewertungen von Partnern, Preisverhandlungen oder private Praktiken. Lernalgorithmen zur Anpassung gehören ebenfalls zum Repertoire. Durch den Modellierer sind auch Kosten zu berücksichtigen, z.B. Steuern und Abgaben, Transaktionskosten und andere. Das resultierende ACE Modell muss *dynamisch komplett* sein, um die Simulation flexibel und 'entwicklungsfähig' zu gestalten. Die Entwicklung des Systems basiert allein auf Grund der modellierten Agenten und ihrer lokalen Verhaltensmuster. Als Ziel der Simulationen unterscheidet Tesfatsion vier Ziele:

Empirisches Verständnis: Warum finden bestimmte Entwicklungen ohne zentrales Planen und Steuerung statt?

Normatives Verständnis: Hierbei geht es um Untersuchungen zur Natur der Resultate. Sind die Ergebnisse der Simulationen als gewünscht zu betrachten (siehe hierzu auch Mechanismus Design [13, 241])?

Theoriebildung: Trägt das Modell zum Verständnis und der Bildung oder Verifizierung (ökonomischer) Modelle bei.

Methodenentwicklung: Methoden und Werkzeuge zur Untersuchung ökonomischer Systeme werden benötigt.

Der wichtigste Punkt bei ACE ist die Untersuchung selbstorganisatorischer Fähigkeiten in dezentralen Märkten. Der dezentrale Koordinationsmechanismus AVALANCHE wird von Eymann [64] vorgestellt. Hier wird die Dynamik von Verhandlungsstrategien in dezentralen Märkten simuliert und untersucht. Insgesamt sechs unterschiedliche Verhandlungsparameter (Gewinnstreben, Konzession, Aufschlag, Bereitschaft zum Verhandlungsabbruch, Orientierung an Marktpreisen, Reputation) bilden die Grundlage komplexer, nicht-deterministischer Verhandlungsstrategien rationaler Agenten. Die Simulation gibt Einblicke in verschiedene Verhandlungsstrategien intelligenter Agenten und deren Veränderung im Wettbewerb. Ohne zentralen Mediator bilden sich nach kurzer Zeit Gleichgewichtspreise heraus. Eymann schreibt dazu:

> "Dadurch wird es möglich, generelle Fragestellungen, die mit der Koordination von Vielkomponentensystemen zusammenhängen, in experimenteller Weise zu betrachten. Der Vorteil einer Simulation auf Basis eines Multi-Agenten-Systems ist zweifellos die Verbindung zwischen Mikroverhalten der Agenten und Makroverhalten des Systems. "

Damit stellt ACE ein mächtiges Werkzeug dar, um die Wechselwirkungen von Agenten und Umwelt in komplexen Systemen zu untersuchen. Es lässt sich die wechselseitige Abhängigkeit zwischen Mikro und Makroebene untersuchen, wie auch Ursache-Wirkungsanalysen per Simulation durchführen.

2.2.7 Weiterführende Ansätze

Die Agententechnologie ist so weit ausgereift, dass sie in der Praxis an vielen Stellen bereits eingesetzt wird und in anderen Bereichen in neue Gebiete vordringt. Insbesondere

die Kommunikations- und Problemlösungsfähigkeiten von Agenten werden dabei als eine Ergänzung bereits bestehender Technologien genutzt. Ein vielfach verwendeter Ansatz kann unter dem Begriff *Kopf-Körper Architektur* subsumiert werden.

Der Agent bildet als Kopf eine Art intelligente Steuerungsschicht für darunterliegende (physische) Ressourcen bzw. Dienste. Dieser Ansatz wurde von Hasenkamp [104] et al. vorgestellt. Hier existieren Körper, Kommunikator und Kopf die jeweils unterschiedliche Aufgaben übernehmen und arbeitsteilig Agentenkonzepte (Kommunikation, Kooperation) mit dem Körper (Problemlösungsfähigkeit einer Ressource, z.B. Datenbank oder Finanzbearbeitungssoftware) verbindet. Aktuelle kooperative Geschäftsmodelle erfordern unter anderem das Potential zur schnellen Anpassung an veränderliche Situationen, sowie zur effizienten Leistungserbringung durch flexible und adaptive Geschäftsprozesse. Die Kopf-Körper Architektur kann hierbei helfen, Flexibilität in statische Abläufe zu integrieren. Die Akteure dieser Geschäftsprozesse werden hierbei in der Regel ökonomisch handeln und dementsprechend ihre individuellen Ziele zur Optimierung ihrer jeweiligen Ergebnisse und Ressourcenallokation verfolgen.

Eine abgewandelte Form der Kopf-Körper Architektur findet sich in [126]. Hier werden Ressourcen als 'units' bezeichnet und pro unit wird ein steuernder Agent eingesetzt. Die Motivation war dezentralisiertes Management verteilter Systeme in Telekommunikationsanwendungen. Auch in serviceorientierten Architekturen und zur Ausführung von Prozessen gibt es eine Reihe von Arbeiten von Buhler und Vidal [39, 40, 41] zum Thema Steuerung von Diensten. Hierbei helfen Agenten bei der Komposition und der Aufstellung (engl. *Enactment*) von Workflows. Hierzu werden agentenorientierte Koordinationsverfahren verwendet und zwei Dinge erreicht: Dynamisierung vormals statischer Workflows und Einführung semantischer Interoperabilität.

Die Kombination von Agenten und Diensten bildet die Vorstufe zur Anwendung in Grids. Anforderungen an Grid-Middleware der nächsten Generationen unter anderem die Unterstützung zur Erzeugung und Anpassung von Organisationen unter Berücksichtigung dynamischer und unvorhersehbarer Nachfrage oder Verfügbarkeit von Ressourcen [248]. Dezentrale Verfahren zur kooperativen Lösung veränderlicher Probleme durch verteilte Knoten werden insbesondere durch Disziplinen wie etwa der verteilten Künstlichen Intelligenz (VKI) und insbesondere Multiagentensystemen (MAS) thematisiert. Insbesondere Agenten werden dabei aufgrund ihrer Eigenschaften als geeignete Ergänzung der Grid-Technologie erachtet und unter anderem auch zur Verbesserung der Komposition von Grid-Diensten vorgeschlagen. Foster, Jennings und Kesselmann haben daher in ihrem wegweisenden Artikel 'Brain meets Brawn' [74] eine Kombination von Agententechnologie (Brain, dt. Intellekt) und Grid (Brawn, dt. Muskeln) vorgeschlagen. Die identifizierten zehn Forschungsthemen sind wie folgt:

- **Service Architektur** als dynamische Komposition, Betrieb und Wiederverwendung höherwertiger Dienste aus einfachen kompatiblen Diensten.
- **Vertrauen und Management** zwischen unbekannten Diensten. Hierbei spielt Authorisierung in virtuellen Organisationen eine große Rolle.
- **Systemmanagement und Fehlerbehebung** adressiert automatisiertes Management und die Erkennung, Diagnose und Beseitigung von Störungen.
- **Verhandlungen** zwischen den konfligierenden Zielen unterschiedlicher Dienste und virtueller Organisationen.

- **Dienstkomposition** sind im Kontext *virtueller Organisationen* (*VO*) ein aktuelles Forschungsfeld unter anderem für Dienstbeschreibungen, Finden von Diensten, Komposition, Monitoring, Management und Anpassung.
- **VO Formierung und Management** benötigt Forschung zur Definition von VO und zur Entscheidung von Initiierung und Auflösung.
- **Systemvorhersagbarkeit** in inhärent unvorhersagbaren Bereichen spielt eine Rolle bei der Zusage von Garantien gegenüber Anwendern (Performance, Quality of Service (QoS), Sicherheit, etc.).
- **Mensch-Maschine Zusammenarbeit** beinhaltet die nahtlose Integration in menschlichen Umgebungen.
- **Auswertung** verschiedener Ansätze, Methoden und Technologien erfordert entsprechende Definition von Benchmarks und Probleminstanzen. Herausforderungen liegen hierbei im Monitoring und Management massiv paralleler Systeme.
- **Semantische Integration** bedeutet Zusammenarbeit technischer Systeme auf höherer Ebene, z.B. durch den Abschluss von Verträgen. Hierbei spielen Ontologien, Beschreibungen, Semantische Mediation, Herstellung von Vertrauen und ökonomische Überlegungen eine Rolle.

Diese Arbeit adressiert einen Teil dieser Bereiche, namentlich das Management, im Sinne von Selbstadaption und Optimierung verteilter Systeme.

2.2.8 Zusammenfassung

Agenten sind als verteilte Problemlöser in dynamischer Umgebung konzipiert. Ihr größtes Versprechen ist Flexibilität [130]. Weitere Eigenschaften, die ihre Nutzung in verteilten Systemen prädestinieren, sind Autonomie und ihre sozialen Fähigkeiten (siehe dazu Weak Notion of Agency in Kapitel 2.2.1). Agenten sind für serviceorientierte Systeme oder Grid Umgebungen ein vielversprechender Ansatz, um Adaptivität durch Flexibilität zu ermöglichen. Hiermit lassen sich innerhalb komplexer Systeme Managementstrukturen aufbauen, die Bottom-Up Verhaltenssteuerung und Monitoring erlauben.

Zur Erreichung hoher Flexibilität liegt der Fokus jedoch auf schwergewichtigen, deliberativen Agenten (z.B. BDI Ansatz). Hier wird die Flexibilität mit zum Teil hoher Berechnungs- und Interaktionskomplexität (siehe Kapitel 2.2.5) erreicht. Sehr einfache reaktive Agenten sind wiederum für vordefinierte Aufgaben geeignet, die mittels simpler (emergenter) Regeln gemeinsam bearbeitet werden. Hierbei ist jedoch von einzelnen Agenten vergleichsweise wenig Flexibilität im Hinblick auf Änderung in der Umwelt zu erwarten. Diese Schwäche wird im Rahmen der Arbeit adressiert.

Für eine größere bzw. unbekannte Anzahl von Agenten sind deren Interaktionsmechanismen ungeeignet, da sie schlecht skalieren [247]. Die Integration innerhalb verteilter Systeme, wie z.B. Grids, erfordert weitere Forschungsarbeit. Der Einsatz von Agent based Computational Economics (ACE) ist ein vielversprechender Ansatz zur Bottom-Up Untersuchung, Modellbildung und Koordination in massiv verteilten und komplexen Systemen. Diese Arbeit adressiert einige von Foster et al. [74] aufgeführten offenen Punkte, die das Systemmanagement im Kontext von Adaptivität und Optimierung betreffen. Hierzu werden ökonomische und evolutionäre Ansätze verwendet.

2.3 Verteilte Evolutionäre Verfahren

Ein heuristisches Optimierungsverfahren ist die Klasse evolutionärer Algorithmen (EA) [18, 19, 60]. Hierbei wird der Prozess der natürlichen Evolution [49] genutzt und auf einer Menge möglicher Lösungen (Population) gearbeitet. Hierdurch entsteht ein inhärent verteiltes [111], tolerantes Verfahren, das unter Ungewissheit und unvollständigen Informationen gute Ergebnisse erzielen kann und gleichzeitig robust und adaptiv ist. Es werden keine speziellen Eigenschaften der Zielfunktion vorausgesetzt und große und nichtkonvexe Suchräume können exploriert werden [23]. Durch den populationsbasierten Ansatz kann das Finden guter Lösungen rechenintensiv werden; gleichzeitig lassen sich evolutionäre Verfahren durch ihre implizite Parallelität relativ einfach verteilen. Alba merkt an, dass Verfahren des *Softcomputing* (evolutionäre Verfahren, *Neuronale Netze*, *Fuzzy Logik*, etc.) robuster als symbolische Techniken sind [4].

Die Vereinheitlichung und damit einhergehende Konvergenz der Theorien ist für *parallele evolutionäre Algorithmen* (*PEA*) noch nicht abgeschlossen [4, 33, 176]. Dieser Abschnitt stellt den Stand der Forschung im Bereich paralleler evolutionärer Algorithmen vor. Hier werden zunächst die Gemeinsamkeiten evolutionärer Algorithmen dargestellt. Danach wird im Wesentlichen auf die Aspekte Klassifizierung paralleler evolutionärer Algorithmen (Master-Slave, Inselmodell, Nachbarschaftsmodell) und agentenbasierte evolutionärer Algorithmen eingegangen.

Begründet durch die Zielstellung dieser Arbeit liegt der Fokus auf verteilten Ansätzen im Bereich evolutionärer Verfahren, die in den Abschnitten 2.3.4 und 2.4 eingehend diskutiert werden. Dennoch verwenden alle Verfahren eine Reihe von allgemeinen Eigenschaften, die in Abschnitt 2.3.3 kurz dargestellt werden. Zunächst werden die Gemeinsamkeiten aller Verfahren angesprochen und danach werden die im Rahmen dieser Arbeit näher beleuchteten parallelen Verfahren genauer beschrieben. Abschließend erfolgt eine Abgrenzung zu weiteren verteilten Optimierungsverfahren in Abschnitt 2.3.7.

2.3.1 Optimierung

Für das Lösen von Problemen wird im Allgemeinen nicht nach irgendeiner Lösung gesucht. Stattdessen ist eine Lösung gefordert, welche die gestellte Aufgabe löst, vorhandenen Nebenbedingungen genügt und gleichzeitig möglichst wenig Ressourcen beansprucht, wie Goldberg [88] auf S.6 beschreibt:

> "[...] optimization seeks to improve performance toward some optimal point or points."

Damit beschreibt *Optimierung* die Suche nach optimalen Lösungen. Im mathematischen Sinne lässt sich die Lösung eines n-dimensionalen Problems ohne Beschränkung der Allgemeinheit als Vektor $x \in X$ darstellen. Hierbei ist $x = (x_1, x_2, ..., x_n)$ der Lösungsvektor und X der n-dimensionale zulässige *Suchraum*.

Das skalare *Optimierungsproblem* lässt sich dafür ohne Beschränkung der Allgemeinheit als *Minimierungsproblem / Maximierungsproblem* darstellen:

$$\begin{aligned} &\text{minimiere / maximiere } f(x), \\ &\text{in Abhängigkeit von } x \in X \end{aligned} \tag{2.1}$$

wobei $f : Y \rightarrow \mathbb{R}$ die *Zielfunktion* darstellt und $X \subseteq Y$ die Menge der gültigen Lösungen über Nebenbedingungen einschränkt.

Es liegt im besonderen Interesse dieser Arbeit, gute bzw. zufriedenstellende (engl. *satisfying*) Lösungen schnell zu finden, wie Goldberg anmerkt. Das Erreichen des Optimums ist weniger wichtig für Komplexe Systeme [88]. Hierbei spielt die Dynamik der Umgebung und des Systems selbst eine wichtige Rolle, denn was nützt die perfekte Lösung, wenn sich in der Zwischenzeit das Problem geändert hat.

2.3.2 Historie

Während die natürliche Evolution implizit parallel verläuft, d.h. ohne zentrale Instanz, gab es im Vergleich zu klassischen EA zu Beginn wenige Arbeiten zur Parallelisierung auf mehrere Rechner [88] und insbesondere zur vollständigen Verteilung auf Individuenebene. Bereits Hollands erste Vision im Jahre 1962 [110] spekulierte über die parallele Natur des Reproduktionsparadigmas und die damit verbundene inhärente Effizienz der Parallelverarbeitung. Erste theoretische Arbeiten erfolgten jedoch erst 1976 durch Bethke [21]. Er kalkulierte erste Komplexitätsabschätzungen für genetische Algorithmen auf parallele Maschinen. Grefenstette [99] untersuchte 1981 parallele Implementierungen genetischer Algorithmen. In den folgenden Jahren entstand eine Vielzahl von Forschungsarbeiten auf diesem Gebiet, z.B. von Banzhaf [15], Schwefel [216], Gorges-Schleuter [91] und in der *Parallel Problem Solving from Nature*[11] (PPSN) Reihe. Seitdem existieren eine Vielzahl unterschiedlicher Parallelisierungsansätze, deren Kategorisierung und Vereinheitlichung in Abschnitt 2.3.4 näher beleuchtet wird. Ebenso existieren für verwandte Bereiche kombinierte Ansätze, wie z.B. für Grid-Computing [33, 155, 240] und Multiagentensysteme [64, 106, 221], die in Abschnitt 2.4 vorgestellt werden.

Die Gründe für Parallelisierung lassen sich unter folgenden Gesichtspunkten zusammenfassen [4, 33, 176]:

- **Ressourcenverbrauch**: Für die Qualität der Optimierung ist eine sehr große Population hilfreich. Ebenso kann die Repräsentation einzelner Individuen sehr aufwändig werden [136]. Beide Faktoren führen zu hohem Speicherbedarf, der Verteilung notwendig macht.

- **Fitnessberechnung**: Es existieren Anwendungsbereiche, die Tage, Wochen oder auch Jahre für die oft aufwändige Fitnessberechnung eines Individuums benötigen [147]. Der einzige praktische Ausweg ist die Verteilung auf mehrere CPUs.

- **Ergebnisqualität**: Einzelne Optimierungsläufe können zu sehr unterschiedlichen Ergebnissen führen, da in nahezu allen nichttrivialen Problemen mehrere (Sub)Optima existieren. Durch parallele Suche kann vorzeitiges Konvergieren in einem Suboptimum zu vermieden werden.

- **Problemzerlegung**: Im Kontrast zur Populationsverteilung wird hier das Gesamtproblem in kleinere Teilprobleme zerlegt, die dann jeweils durch einen separaten evolutionären Algrithmus bearbeitet und am Ende zu einer Gesamtlösung zusammengeführt werden [244].

[11] siehe http://ls11-www.cs.uni-dortmund.de/PPSN/

- **Implizite Problemverteilung**: Implizit verteilte Probleme die sich z.B. über mehrere Organisationen erstrecken sind aufgrund geschützten geistigen Eigentums nicht zur Zusammenführung geeignet. Ebenso kann aufgrund von Ressourcenbeschränkungen (CPU, Speicherplatz, Bandbreite) die Zusammenführung von Informationen beschränkt oder verhindert werden [184]. Das Problem wird hier basierend auf implizit verteilter Information bearbeitet.

Die Gründe für Parallelisierung gelten für alle Arten evolutionärer Verfahren gleichermaßen. In den letzten zwei Jahrzehnten haben sich die verschiedenen Arten evolutionärer Algorithmen sehr stark durchmischt, so dass die theoretischen Grundlagen vereinheitlicht werden konnten [4, 18]. Folgende Übersicht zeigt einige Arten von Evolutionären Algorithmen und führt deren Besonderheiten auf:

Evolutionäre Strategien (ES) [22, 197] arbeiten auf Vektoren von reellen Zahlen und nutzen hauptsächlich Mutation und Selektion.

Genetische Algorithmen (GA) [88] basieren auf einer Bitstring-Kodierung.

Genetische Programmierung (GP) [136] bezeichnet meist evolutionär produzierte Programme, Datenstrukturen oder andere Konstrukte. Diese haben häufig komplexe Baumstrukturen.

Die einzelnen Verfahren bestehen zwar weiter, dennoch wird im Kontext verteilter Verfahren keine Unterscheidung betrachtet. Die nachfolgend dargestellten grundlegenden allgemeinen Eigenschaften sind gleich.

2.3.3 Allgemeine Eigenschaften

Zunächst wird eine initiale Menge an Lösungen (Individuen) erstellt, die entweder zufällig generiert [88] oder bereits vorbereitet [105] sind. Im nächsten Schritt müssen die Individuen im Sinne ihrer Lösungsqualität vergleichbar gemacht werden. Dazu wird jedem Individuum ein Fitnesswert zugeordnet. Es reicht jedoch, eine Ordnungsrelation über die Individuen aufzubauen [119]. Die Vergleichbarkeitsrelation kann ein- oder mehrdimensional (Pareto-Prinzip [52]) sein und dient als Grundlage der Selektion.

Kodierung

Unterschiedliche Probleminstanzen verlangen nach unterschiedlichen Repräsentationsformen. Daher variieren evolutionäre Algorithmen in der Art der Individuenrepräsentation [23]. Gängige Repräsentationen sind z.B. Bitstrings [88, 111], Vektoren reeller oder ganzer Zahlen [158], Bäume [105, 203], Graphen [230] oder auch beliebige andere Datenstrukturen. Theoretisch können alle Repräsentationen auf Bitstrings zurückgeführt werden, wie Holland vorschlägt [111]. Dies führt jedoch zu erhöhtem Aufwand wie beispielsweise zusätzlichen Reparaturoperatoren, um ungültige Repräsentationen zu reparieren.

In der Literatur existiert eine Vielzahl von Arbeiten über evolutionäre Operatoren, während die Repräsentation oft vernachlässigt wurde. Rothlauf untersucht in [203] den Einfluss von Repräsentationen auf die Performance. Hierbei werden Aspekte wie Redundanz, Skalierbarkeit, Lokalität untersucht und als Einflussfaktoren auf die Performance erkannt.

Eine wichtige Rolle spielen sogenannte Schemata als Templates konkreter Kodierungen. Das ursprünglich von Holland entwickelte *Schema-Theorem* [88, 111, 113] definiert Schemata als eine Teilmenge der Kodierung mit fixierten und variablen Bestandteilen und einer festgelegten Länge. Die Fitness eines Schemas ist die durchschnittliche Fitness aller Kodierungen, die auf das Schema passen. Das Schema-Theorem besagt, dass evolutionäre Verfahren parallel Schemata verarbeiten, denn jede Kodierung passt auf eine Menge von Schemata. Die Auswahl von Schemata erklärt die sogenannte *Building-Block Hypothese* [88, 111]. Rothlauf formuliert diese wie folgt [203]:

> "short, low-order and highly fit schemata can be recombined to form higher-order-schemata and complete strings with high fitness. "

Es werden also durch evolutionäre Suche ständig multiple Schemata verarbeitet, wobei kurze und fitte Schemata in den Lösungen zu Schemata höherer Ordnung (Länge) kombiniert werden.

Selektion

Während der Selektion werden Individuen anhand ihrer Fitness ausgewählt, deren Gene in einen sogenannten *Mating Pool* weitergegeben werden. Damit wird fitnessgesteuert über die Weitergabe der Gene entschieden: erfolgreiche Gene werden häufiger weitergegeben. Die Selektion muss über den Selektionsdruck folgende Punkte sicherstellen [88]:

- Vorzeitige Konvergenz vermeiden: besonders in kleinen Populationen kann ein Individuum (sog. *Superindividuum*) mit einer sehr großen Fitness relativ zur Gesamtfitness zu einseitiger Selektion (zu hoher Selektionsdruck) führen. Zu hoher Selektionsdruck führt zu vorzeitiger Konvergenz und damit zu sub-optimalen Ergebnissen.
- Dauerhaft gerichtete Exploration: später während des Verlaufes werden sich die Unterschiede zwischen den Individuen nivellieren bis hin zu minimalen Differenzen zwischen den besten und schlechtesten Individuen. In dieser Situation werden gute und schlechte Gene etwa zu gleichen Anteilen weitergegeben. Damit stoppt die zielgerichtete Optimierung und wird zu einer zufälligen Suche (engl. *random walk*) zwischen den mittelmäßigen Individuen. In diesem Fall ist der Selektionsdruck zu niedrig und die Suche dauert länger.
- Diversität erhalten: zusätzlich zu den ersten beiden Anforderungen soll die Diversität innerhalb der Population gewahrt bleiben, um vorzeitige Konvergenz zu vermeiden.

Damit hat die Selektion entscheidenden Einfluss auf das Ergebnis einer Berechnung. Die Wahl der besten Selektionsmethode hat über den Selektionsdruck sehr großen Einfluss auf die Konvergenzgeschwindigkeit eines evolutionären Algorithmus. Wie Goldberg et al. zeigt [87, 235], kann zu langsame oder zu schnelle Konvergenz zum Versagen des Algorithmus führen. Ein ideales Selektionsschema sollte einfach zu implementieren sein und effizient auf parallelen und nicht parallelen Architekturen funktionieren. Weiterhin sollte es die Möglichkeit zur Justierung des Selektionsdruckes bieten, um die Performance an unterschiedliche Domains anzupassen [159]. Hierbei entscheidet der Selektionsdruck über die Suchstrategie zwischen den beiden Extremen *Exploration* und *Exploitation* [207]. Exploration bedeutet das Erschließen

unbekannter Regionen von Suchräumen, während Exploitation die gefundenen Lösungen weiter verbessert. In der Biologie werden die natürlichen Pendants als *r-Selektion* (Exploration) und *K-Selektion* (Exploitation) bezeichnet [149]. r ist die Wachstumsrate einer Population, während K für die Kapazitätsgrenze (engl. *carrying capacity*) des Lebensraumes steht. Im Idealfall sorgt die Selektion dafür, dass eine Population zunächst bis zur Kapazitätsgrenze wächst und danach durch Konkurrenz die fittesten Individuen überleben.

Nach Sastry und Goldberg [208] lassen sich Selektionsverfahren weitestgehend in zwei Kategorien einteilen: (1) proportionsbasierte und (2) ordnungsbasierte Verfahren. Proportionsbasierte Verfahren selektieren Individuen anhand ihrer Fitness [88, 112]. *Fitnessskalierung* kann helfen, vorzeitige Konvergenz zu vermeiden und dauerhaft gerichtete Exploration sicherzustellen ([88] S.76). Die Fitnesswerte werden hierbei durch eine geeignete Abbildung skaliert um zu große und zu kleine Differenzen auszugleichen und den Selektionsdruck anzupassen. Beispiele für proportionale Selektion sind fitnessproportionale Selektion [111], stochastischer Rest Selektion (engl. stochastic remainder selection) [30, 35] und stochastische Universalselektion (engl. stochastic universal selection) [10]. Die Erstellung des Rankings und eventuelle Anpassungen zur Laufzeit sind weitere Aspekte bei der Anwendung proportionsbasierter Verfahren, die als Skalierungsproblemen bezeichnet werden [160, 208]. Ordnungsbasierte Verfahren sind von Skalierungsproblemen nicht betroffen, da ein Ranking der Individuen erstellt wird. Hierbei bestimmt der Rang der Individuen den Selektionsdruck und nicht die Verteilung der Fitness innerhalb der Population. Beispiele sind Tournament Selektion [35, 86], $\mu - \lambda$ Selektion [215] und rangbasierte Selektion [9, 10].

Rekombination

Rekombination ist ein evolutionärer Operator und produziert neue Individuen durch Vermischung des Erbgutes der Individuen aus dem Mating Pool [19, 60, 111]. Im Falle von Bitstrings wird dies auch als Crossover bezeichnet. Ziel der Rekombination ist der Austausch und die Bewahrung erfolgreicher, sogenannter Building Blocks, innerhalb des Erbgutes. Ein weiterer Aspekt ist die Exploration des Suchraumes durch Vermischung des Erbgutes. Zusätzlich kann ein Effekt der genetischen Reparatur eintreten (engl. genetic repair effect), beschrieben in [141]. Hierbei werden durch den idealen Rekombinationsoperator Teile des Erbgutes entfernt, die mitverantwortlich sind für verminderte Fittness. Durch Rekombination werden neue Bereiche des Suchraumes erschlossen (Exploration).

Wesentliche und oft genutzte Parameter für die Rekombination sind die Anzahl der Crossoverpunkte (engl. crossover points) und die Anzahl der Elternindividuen. Die Anzahl der Crossoverpunkte entscheidet darüber, an wievielen Stellen das Erbgut der Individuen aufgespalten und kombiniert wird (Schnittpunke). Gängige Werte sind z.B. 1 (single point), 2 (double point) und mehrere Punkte (multi point). Die Verallgemeinerung wird als uniform crossover bezeichnet und bedeutet, dass jede Stelle im Erbgut mit einer bestimmten Wahrscheinlichkeit weitergegeben werden kann [223]. Die Anzahl der Elternindividuen (engl. multiparent recombination) legt fest wieviele elterliche Gene vermischt werden, um Nachkommen zu erzeugen. Üblich sich hier meist zwei Elternindividuen, um einen guten Kompromiss zwischen Aufwand (Implementierung) und Resultaten (Vermischung des Erbgutes) zu erzielen.

Mutation

Die durch Rekombination erzeugten Nachkommen (engl. offspring) werden weiter manipuliert. Mutation ist ebenfalls ein explorativer genetischer Operator. Diese ungeschlechtliche Variation resultiert in einer zufälligen Änderung des Erbgutes [23]. Die Wahrscheinlichkeit einer zufälligen Änderung wird als Mutationsrate bezeichnet.

Es existieren unterschiedlichste Verfahren für unterschiedliche Repräsentationsformen. Eine einfache Möglichkeit stellt das 'Kippen' von Bits in einem Bitstring dar [88]. Für reelle Zahlen existiert der Mutationsoperator des *Breeder Genetic Algorithmus* von Mühlenbein [168, 169]. Hierbei wird pro reeller Zahl z eine Gauß-Verteilung mit Erwartungswert $z = \mu$ und einer wertebereichsabhängigen Standardabweichung verwendet, um die mutierte Zahl z' zu erzeugen. In Datenstrukturen wie Graphen und Bäumen werden Teile getauscht, entfernt oder verändert.

Populationsgröße

Natürliche Populationen sind dynamisch in Raum und Zeit und unterliegen damit einer beständigen Änderung. Evolutionäre Verfahren nutzen zumeist eine statische Populationsgröße [148] (*steady-state*), dabei hat die Wahl der 'richtigen' Populationsgröße entscheidenden Einfluss auf die Robustheit und das Ergebnis [134]. Kleine Populationen können schneller vorzeitig konvergieren und damit suboptimale Lösungen liefern. Grosse Populationen hingegen erfordern teils erheblichen Rechenaufwand von Tagen, Wochen oder gar Jahren [147]. Schut und Eiben [61] sehen hohes Potential für neue Variationen und empfehlen insbesondere die Betrachtung der Populationsgröße. Erste Resultate mit dynamischen Populationsgrößen sind vielversprechend [90, 134, 148].

Goldberg, Deb und Clark [90] berechnen die Populationsgröße anhand statistischer Annahmen über das Problem (Komplexität, Länge). Koumousis und Katsaras [134] schlagen variable Populationsgröße ähnlich einem Sägezahnmuster vor. Zusätzlich wird die Population im Zyklus des Sägezahnmusters partiell reinitialisiert. Die Performance ist der von Standard EA überlegen. Arabas et al. schlagen eine maximale Lebensdauer und Alter pro Individuum vor. Hierbei erhält jedes Individuum bei seiner Entstehung eine maximale Lebensdauer basierend auf der Fitness. Das Alter wird mit 0 initialisiert und jede Generation um eins erhöht. Erreicht das Alter die Lebensdauer, wird das Individuum aus der Population entfernt. Damit wird die Populationsgröße veränderlich. Eine ausführliche Untersuchung weiterer Verfahren findet sich in [59].

In natürlichen Umgebungen ist die Populationsgröße durch die Kapazität des Lebensraumes bzw. *Tragfähigkeit* (engl. carrying capacity) der Umgebungsenergie bzw. der vorhandenen Nahrung begrenzt. Populationsgrößen stabilisieren sich bei Erreichen der Tragfähigkeit. Menczer [156] verwendet von der Umgebung bereitgestellte Energie, die den Individuen zur Verfügung gestellt wird. Hierbei limitiert die Energiemenge die Anzahl der Individuen. Otto et al. [182, 185] schlägt eine nachfragebasierte Populationsgröße vor. Dabei stellt die Umgebung ähnlich wie bei Menczer anstelle Energie Geld zur Verfügung. Die Menge des Geldes hängt von der Höhe der Nachfrage ab. Die Populationsgröße wird durch das vorhandene Geld bzw. die Nachfrage begrenzt. Die umgebungsbasierten Verfahren sind eher für online oder Echtzeitadaption bzw. langlaufende Berechnungen mit eventuell veränderlichen Problemen geeignet.

2.3.4 Klassifikation

Die Einteilung paralleler evolutionärer Verfahren in der Literatur ist in den letzten Jahren vereinheitlicht worden. Aktuell herrscht Konsens über die Klassifikation in drei Hauptkategorien [2, 4, 33] neben der *panmiktischen* Variante mit einer einzigen Population (engl. *panmictic* oder *canonical* [112]). Abbildung 2.10 zeigt eine Übersicht über folgende Einteilung:

Master-Slave Modell (Abbildung 2.10b): ein Masterknoten speichert die Population und führt die Operationen Selektion, Rekombination und Mutation aus. Die Fitnessberechnung wird verteilt auf den Slaveknoten ausgeführt.

Insel Modell (Abbildung 2.10c): Mehrere Subpopulationen laufen parallel und tauschen Individuen miteinander aus.

Diffusions Modell (Abbildung 2.10d): Die Individuen sind in einer räumlichen Struktur (meist 2D Grid) in einer sogenannten *Nachbarschaft* (grau hinterlegt) angeordnet mit einem Individuum pro Knoten. Nur Individuen innerhalb einer kleinen Nachbarschaft können miteinander interagieren.

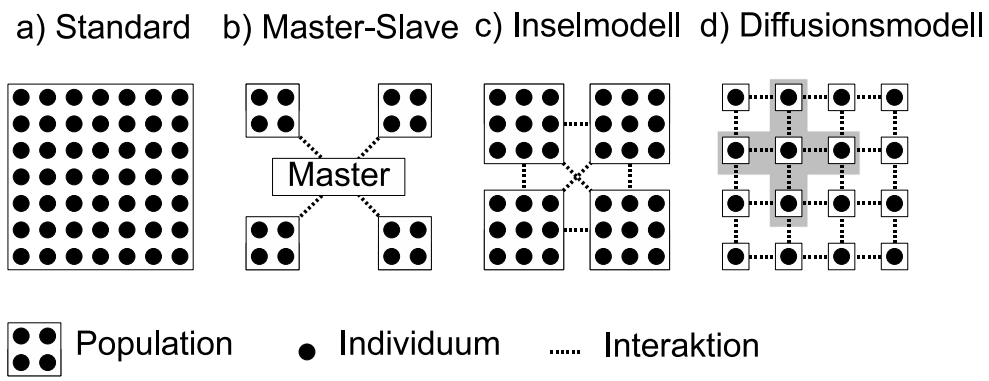


Abbildung 2.10: Klassen paralleler evolutionärer Algorithmen nach [2, 4, 33] im Vergleich zu klassischen EA

Daneben gibt es weitere Mischformen zwischen den genannten, die als hierarchische parallele Ansätze bezeichnet werden [2, 42], da verschiedene der oben genannten Ansätze auf unterschiedlichen Ebenen kombiniert werden. Ebenso existieren weitere Zwischenformen, deren Eigenschaften aus den oben genannten abgeleitet wurden. Eine Übersicht hierzu gibt [176].

Im Zusammenhang paralleler evolutionärer Verfahren sprechen Alba, Tomassini [4] und Luque, Alba, Dorronsoro [2] von strukturierten Populationen. Dabei wird unterschieden zwischen der Strukturierung einer Population und ihrer (parallelen) Implementation. Jede der oben genannten Strukturierungsmöglichkeiten kann auf unterschiedliche Weise (z.b. paralleles Prozessorarray, Grid oder sequentiell, siehe dazu Abschnitt 2.1.4) implementiert werden, während die jeweiligen Eigenschaften unverändert bestehen bleiben.

Die folgenden Abschnitte gehen näher auf die einzelnen Kategorien ein und beleuchten dabei vor allem die mögliche Verwendung im Hinblick auf Dezentralität. Die Fitnessberechnung und genetischen Operatoren lassen sich in allen Fällen sehr gut parallelisieren, da sie unabhängig voneinander ausgeführt werden können [33]. Die Selektion ist der Flaschenhals,

da hier globale Informationen notwendig sind, um die Individuen zu vergleichen. Ein weiterer wichtiger Punkt ist die Wahl der richtigen Populationsgröße, da diese entscheidenden Einfluss auf Robustheit und Ergebnis [2, 134] hat.

2.3.4.1 Master-Slave

Eine einfache Möglichkeit der Parallelisierung [2] ist die Verteilung der Fitnessberechnung. Beim Master-Slave Ansatz übernimmt ein Masterknoten die Koordination, während die Slave-Knoten für die (aufwändige) Fitnessberechnung zuständig sind. Der Masterknoten führt die Operationen Selektion, Rekombination und Mutation aus und arbeitet auf der Gesamtpopulation. Dieser Ansatz ist damit aufgrund folgender Charakteristika sehr interessant [2]:

- Die Exploration erfolgt exakt nach denselben Gesetzen wie bei klassischen EA. Damit können existierende Gesetzmäßigkeiten, Erkenntnisse und Best-Practices direkt angewendet werden. Im asynchronen Fall verhält sich ein Master-Slave Algorithmus genau wie ein Single-Population EA.
- Die Implementierung gestaltet sich vergleichsweise einfach und findet daher schneller Verwendung in praktischer Hinsicht.
- Bisherige Ergebnisse zeigen in vielen Fällen deutliche Performancesteigerungen gegenüber dem klassischem Ansatz.

Die Laufzeit wird durch zwei Einflussfaktoren wesentlich determiniert: Zeit für Berechnungen und die Zeit für Kommunikation zwischen Master und Slaves. Zunächst sendet der Master an jeden Slave einen Teil der Population zur Auswertung. Die Slaves starten die Berechnung, wenn sie ihren Teil der Population erhalten haben und senden das Resultat nach Beendigung der Berechnung zurück. Je mehr Slaves genutzt werden, desto schneller läuft die Berechnung, allerdings zu Lasten des Kommunikationsaufwands. Dieser steigt mit jedem Knoten weiter an. Besonders wenn die Kommunikation synchron erfolgt, hat die Kommunikation bei vielen Slaves schnell einen negativen Einfluss auf die Gesamtperformance. Die Anzahl der Slaves lässt sich nicht beliebig erhöhen, sondern es gibt ein Optimum. Jeder zusätzliche Slave verringert dann die Gesamtberechnungsleistung. Im asynchronen Fall wird dieser Effekt abgemildert und die Berechnung läuft insgesamt effizienter. Das Verhalten ist dann nicht mehr äquivalent zu einem klassischen EA (siehe erster Punkt der o.g. Charakteristika) und weitere Parameter und zusätzlicher Aufwand sind erforderlich.

Insgesamt steht mit dem Master-Slave Ansatz ein einfaches Verfahren zur Verfügung, das jedoch im Hinblick auf die Anforderungen nicht geeignet ist. Zum einen ist die Skalierbarkeit je nach technischen Gegebenheiten bis hin zu einer gewissen Anzahl an Prozessoren gegeben, darüber hinaus fällt diese sogar ab. Der Master ist in diesem Fall der Flaschenhals. Adaptivität während des Betriebes ist nicht gegeben, eine manuelle Änderung der Parameter ist notwendig. Ebenso verhält es sich mit Änderungen der Umgebung, auf die manuell reagiert werden muss (z.B. durch Neustart). Kommunikation ist hierbei ein wesentlicher Bestandteil. Gibt es hier Probleme, findet keine weitere Verarbeitung mehr statt.

2.3.4.2 Insel-Modell

Beim Insel-Modell (engl. *island model*) [2, 4, 42] handelt es sich um tatsächlich voneinander unabhängige, verteilte Inseln (mehrere Populationen bzw. engl. *multi-population* oder auch

Demes) mit je einem unabhängigen evolutionären Algorithmus. Daher wird hier auch von verteilten EA's (engl. *distributed EA*'s) oder kurz dEA gesprochen. Anhand einer sogenannten Migrationsstrategie (engl. *migration policy* werden Individuen ausgetauscht, um die Qualität der einzelnen Individuen in den Inseln zu verbessern. Die Untersuchungsschwerpunkte liegen bei diesem Modell auf (1) Wahl der optimalen Populationsgröße der Inseln und (2) der richtigen Migrationsstrategie.

Goldberg, Deb und Clark [90] leiten aus der Problemkomplexität die Populationsgröße ab. Ebenso beschreiben Harik et al. [103] dieses Verfahren als gambler's ruin. Hierbei wird für eine gewünschte Erfolgswahrscheinlichkeit in Abhängigkeit weiterer statistischer Größen die Populationsgröße berechnet. Erfolgswahrscheinlichkeit bedeutet hier die vollständige Durchdringung der Population mit einem gewünschten Building Block[12]. Eine weitere einfach zu realisierende Möglichkeit ist das parallele unabhängige Berechnen mit mehreren Populationen. Anschließend wird lediglich das beste Ergebnis weiterverwendet. Alternativ kann bei konstanter Ergebnisqualität die Populationsgröße der Einzelpopulationen reduziert werden, um das gewünschte Ergebnis schneller zu erreichen. Wie Cantú-Paz und Goldberg [43, 44] zeigen, ist dieses Vorgehen zumindest für linear zerlegbare Funktionen nicht sehr effizient. Sehr viel effizienter als unabhängige Populationen ist die Nutzung von Migration. Damit erweitert sich für jede der Subpopulationen das genetische Material, da von den Nachbarpopulationen anhand der Migrationsstrategie neue Individuen hinzukommen. Für den Moment wird von einer Migration nach Konvergenz der Populationen ausgegangen. Jede Population wird durch diese Migration von ihren Nachbarn beeinflusst; im Falle von mehrfacher Migration auch von den Nachbarn der Nachbarn usw. Diese Erweiterung der Populationen wird als *Nachbarschaft*, bzw. im Falle mehrfacher Migration als *erweiterte Nachbarschaft* bezeichnet und hat Einfluss auf das gambler's ruin Modell: eine weitere Reduktion der (Sub)Populationsgröße ist möglich.

Wie in Abschnitt 2.3.3 beschrieben, reguliert die Selektion über den Selektionsdruck die Konvergenz und hat damit großen Einfluss auf das Ergebnis. Zusätzlich besteht ein entscheidender Zusammenhang zwischen Populationgröße und Selektionsdruck, wie Beck und Mühlenbein schreiben [18, 170]. Die entscheidende Rolle hierbei spielt die Migration, da diese die (erweiterte) Nachbarschaft der Inseln determiniert. Es entsteht somit für jeden EA, der auf einer Population arbeitet ein viel größerer Pool an genetischem Material. Durch die weitgehende Isolation der einzelnen Populationen können diese in unterschiedlichen Regionen des Suchraumes arbeiten. Ein weiterer wichtiger Fakt ist eine gleichmäßige Verbreitung guter Lösungen durch die Populationen [2]. Erfolgreiche Berechnungen verlangen daher nach einer geeigneten Migrationsstragegie, deren Parameter folgende wesentliche Punkte beinhalten:

- **Topologie**: Hiermit wird die Nachbarschaft der Populationen definiert. Diese Graphenstruktur bestimmt, welche Inseln Migranten miteinander tauschen, senden oder empfangen dürfen. Oft ist diese Topologie analog zur physikalischen Topologie der darunterliegenden Hardware.

- **Migrationsinterval**: Der Austausch von Individuen erfolgt einem zeitlichen Intervall. In der englischsprachigen Literatur wird der Begriff *migration gap* verwendet. Hierbei kann festgelegt sein, nach wievielen Generationen periodisch Migranten geschickt werden. Ebenso gibt es wahrscheinlichkeitsbasierte Ansätze, die anhand einer gegebenen Wahrscheinlichkeit jede Generation Migranten senden oder nicht.

[12] ein abgegrenzter Teilbereich bestimmter Gene, welcher die Fitness eines Individuums mit hoher Wahrscheinlichkeit verbessert, siehe auch [88]

- **Migrationsrate**: Für jede Migrantion muss festgelegt werden, wieviele Individuen betroffen sind. Hier kann die genaue Anzahl oder ein Prozentsatz von der Population gegeben sein.
- **Migrantenselektion**: Ähnlich wie bei der Selektion wird hier entschieden, welche Individuen für die Migration ausgewählt werden.
- **Migrantenersetzung**: Ebenso wie bei der Selektion gibt es unterschiedliche Strategien für die Einbringung von empfangenen Migranten in eine bestehende Population. Eine verbreitete Strategie hierfür ist das Ersetzen schlechter Individuen durch gute Migranten (gut → schlecht), z.B. [143, 167]. Theoretische Berechnungen für die verschiedenen Migrantenersetzungsstragegien ergaben erwartungsgemäß eine bessere Performance von gut → schlecht gegenüber anderen Strategien, wie gut → zufällig, zufällig → schlecht oder zufällig → zufällig [2].

Damit funktioniert das Insel-Modell auch bei langsamer Kommunikation aufgrund geringer Bandbreite [33]. Das gilt jedoch nur, solange die beiden Parameter Migrationsrate und Migrationsintervall entsprechend gering dimensioniert sind. Selbst bei einem Ausfall einiger Populationen können die restlichen vorerst ungestört weiterarbeiten. Hierbei sind ohne Migration Performanceeinbußen in Kauf zu nehmen. Damit ist bei entsprechend gestalteter Topologie eine nahezu unbegrenzte Skalierbarkeit gegeben. Ebenso ist eine weitgehende Fehlertoleranz gegenüber Ausfällen von Prozessoren bzw. Bandbreite gewährleistet. Während ältere Ansätze meist synchronen Migrantentausch in heterogenen Netzwerken voraussetzen, untersuchte Chong [46] als einer der ersten asynchrone Kommunikation. Darauf aufbauend wird von Branke, Kamper und Schmeck [33] der Effekt von Heterogenen Rechnern (speziell deren unterschiedliche Verarbeitungsgeschwindigkeit) untersucht. Naturgemäß verbreiten schnellere Knoten ihre Migranten eher und öfter und beeinflussen damit negativ die Konvergenz der Gesamtpopulation, da gute Migranten eine Population 'übernehmen' können und damit die Diversität erheblich einschränken. Branke et al. [33] empfehlen daher eine adaptive Migrationsstrategie, so dass jede Population Migranten zu einem gegebenen Zeitpunkt anfordern kann (z.B. nach Konvergenz). Ebenso erwies sich die Ringstruktur der Inseln als vorteilhaft. Dies dürfte sich in stark verteilten und dynamischen Umgebungen nicht einfach durchsetzen lassen. Möglich wären hier selbstorganisierende Populationen, die selbständig eine Ringstruktur anstreben und diese auch bei Ausfall von Teilen des Netzwerkes selbständig wiederherstellen. Ebenso wird von Branke [33] für heterogene Netzwerke ausschließlich asynchrone Kommunikation empfohlen. Wie alle bisherigen evolutionären Verfahren enthält ein Individuum jeweils eine komplette Lösung, daher müssen alle problemrelevanten Informationen für jede Population vollständig vorliegen.

Ein weiterer Punkt, der im Zuge des adaptiven Migrantenimports bereits angeklungen ist, betrifft die Einteilung nach sogenannten *uniform* bzw. *homogenen* und *nonuniform* bzw. *heterogenen* parallelen Populationen. Tanese [233] untersuchte als einer der Ersten Effekte unterschiedlicher Mutations- und Crossoverraten in verschiedenen (nonuniformen) Populationen. Ebenso existieren Ansätze unterschiedlicher Repräsentationsgenauigkeit, die von Lin [144] untersucht wurden.

Aufgrund seiner verteilten Populationen wird das Insel-Modell als *coarse grained* (grobkörnig) bezeichnet [144], im Gegensatz zum *fine grained* (feinkörnigen) Diffusions-Modell.

2.3.4.3 Diffusions-Modell

Der in dieser Arbeit nachgegangenen Frage nach Dezentralisierung kann mit dem Diffusions-Modell die vorläufig beste Antwort gegeben werden. Alternative Bezeichnungen in der Literatur sind *fine grained* (dt. feinkörnig) oder cellular EA (*cEA*). Durch die parallele gitterartige Struktur (siehe Abbildung 2.10d) wird konzeptuell eine einzige Population über mehrere Knoten verteilt. Dabei enthält jeder Knoten wenige, zumeist jedoch nur ein Individuum [80]. Mutation und Rekombination erfolgt lokal auf jedem Knoten, während die Selektion anhand der Nachbarschaft arbeitet. Ebenso wie beim Insel-Modell wird die Gesamtpopulation in überlappende Nachbarschaften strukturiert. Abbildung 2.10d zeigt exemplarisch die Nachbarschaft (grau hintelegt) für einen Knoten bzw. Individuum. Damit entscheidet die Struktur des Gitters über die Nachbarschaftsbeziehungen. Wichtig ist an dieser Stelle anzumerken, dass die Nachbarschaft über die Population und nicht über den Suchraum definiert ist.

Auf jedem Knoten läuft ein evolutionärer Algorithmus, dessen Population aus einem (oder wenigen) Individuen besteht. Kontinuierlich wird in der Nachbarschaft des Knotens nach Individuen zur Reproduktion gesucht. Die Nachkommen ersetzen das jeweilige Individuum. Durch überlappende Nachbarschaften werden genetische Diffusions- und Migrationseffekte erzielt. Die Überlappung der Nachbarschaften ergibt sich aus der Nachbarschaftsform und -größe. De Jong untersucht in [207] verschiedene Nachbarschaftsformen anhand ihrer Informationsausbreitung. Die Zeit bis zur Ausbreitung eines besten Erbmaterials über die gesamte Population wird als *Takeover time* bezeichnet (siehe dazu auch Abschnitt 2.3.6). Branke, Anderson und Schmeck zeigen in [32] vier Selektionsverfahren für ein $N \times N$ Gitter mit einem Aufwand von $\mathcal{O}(\sqrt{}(N))$. Dennoch ist die globale Selektion der Flaschenhals beim Diffusions-Modell.

Obwohl cEA ursprünglich für massiv parallele Architekturen, wie etwa Parallelcomputer, entworfen wurden, wird dieses Konzept auch für andere verteilte Systeme oder auch auf Monoprozessorarchitekturen angewandt. Es ist damit zunächst ein Modell zur Strukturierung von Populationen mit gleichen Eigenschaften auf unterschiedlicher Hardware. In verteilten Umgebungen ist der Aufwand für Aufrechterhaltung des Grids gegenüber dEA höher, da für jeden Schritt Nachrichten ausgetauscht werden müssen.

2.3.4.4 Mischformen

Es existieren unterschiedlichste Kombinationsformen der oben genannten Kategorien. Hierbei ist zu unterschieden zwischen hierarchischen Mischformen, bei denen unterschiedliche Klassen ineinander geschachtelt werden. Diese Art der Kombination bezeichnet Cantù-Paz [42] als hierarchische genetische Algorithmen und Luque et al. als hybride Modelle [80]. Als Beispiel wird ein Insel Modell auf oberster Ebene verwendet und jede Insel selbst ist als Master-Slave realisiert. Theoretisch sind beliebige Schachtelungstiefen möglich, auch wenn in der Praxis selten mehr als zwei Ebenen verwendet werden.

Eine weitere Taxonomie schlägt Nowostawski [176] vor. Die verschiedenen Ansätze werden nach folgenden Kriterien gruppiert:

- Wie wird die Fitness bewertet und Mutation angewendet?
- Wird eine oder mehrere Subpopulationen verwendet?
- Wie werden Individuen über mehrere Subpopulationen ausgetauscht?
- Wie wird Selektion angewendet (lokal oder global)?

Insgesamt beschreibt Nowostawski acht unterschiedliche Klassen. Allerdings tauchen in dieser Einteilung die Kategorien cEA und dEA auf. Alle anderen Arten der Parallelisierung sind hybrider Natur. Daher werden indirekt die allgemein identifizierten drei Klassen bestätigt.

Eine weitere Einteilung gibt Luque et al. [80] als sogenannte *Strukturierte Genetische Algorithmen* bzw. *strukturierte Populationen*. Die Idee hierbei ist die Klassifikation anhand von Parametern, die den Aufbau und die Kopplung von Strukturen bestimmen:

1. Anzahl der Subpopulationen (wenig - viel)
2. Größe der Subpopulationen (klein - groß)
3. Grad der Interaktion zwischen den Subpopulationen (wenig - viel)

Abbildung 2.11 zeigt einen 3D-Würfel mit den drei Hauptkategorien innerhalb der strukturierten Population Klassifikation. Während dEA üblicherweise mittelgroße Subpopulationen besitzen, haben cEA normalerweise nur ein Individuum. In dEA sind die Subpopulationen meist lose gekoppelt und können auch autonom arbeiten, erfordern cEA eine enge Bindung der einzelnen Subpopulationen. Durch deren Anordnung im Gitter sind die Nachbarschaften klar definiert und Interaktionen sind notwendig. Master-Slave Interaktionen sind ebenfalls häufig und vor allem auch notwendig, um den Algorithmus auszuführen. In Master-Slave existieren wenige Subpopulationen, während dEA mehr und cEA aus einer sehr großen Anzahl hiervon bestehen. Die in Abbildung 2.11 gezeigten Punkte sind keine genaue Verortung der einzelnen Klassen, sondern stellen vielmehr eine Art Schwerpunkt dar.

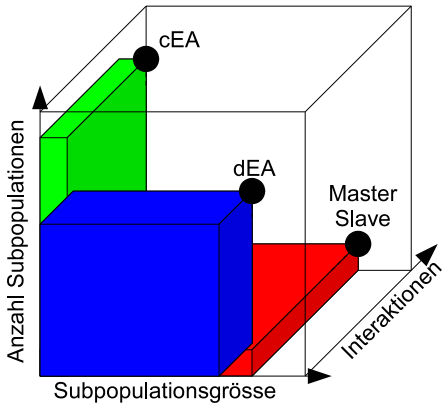


Abbildung 2.11: Strukturierte Populationen

Generell sind die Kommunikationsvoraussetzungen aller Klassen trotz der Unterschiede niedrig, daher kann auch preiswerte Standardhardware zum Einsatz kommen [2]. Das Verhalten paralleler EA mit räumlich verteilten Populationen bietet algorithmische Vorteile [50, 167], da Änderungen zunächst lokal Entstehen und sich räumlich ausbreiten müssen. Hierbei können besser mehrere Regionen gleichzeitig exploriert und auch mehrere Optima gefunden werden.

Die $1:1$ Umsetzung von strukturierten Populationen auf Hardware ist denkbar, doch lassen sich alle Ansätze auf nahezu jede Hardware abbilden (virtualisieren). So ist z.B. das Diffussionsmodell auf nur einem Prozessor denkbar. Die Vorteile liegen in veränderter Migration von Lösungen durch das System und z.B. in der Exploration mehrerer Optima. Die

Anwendung verteilter EA auf beliebigen Hardwarestrukturen ist noch sehr neu und es existieren dazu wenige Untersuchungen. Alba hat daher als zukünftige Themen die Anwendungen in räumlich getrennten Clustern (Grid, Peer-to-Peer Netzwerke) identifiziert [80].

Die beim Diffusionsmodell und damit auch in verschiedenen Mischformen auftretende globale Selektion stellt einen Flaschenhals in stark verteilten cEA dar. Eine Möglichkeit die zu umgehen, ist die sogenannte *lokale Selektion*.

2.3.5 Lokale Selektion

Lokale Selektion führt eine strukturierte Umgebung ein, in der sich Individuen befinden [92, 207]. Die Struktur enthält Knoten (Individuen) und Kanten. Selektion erfolgt in einer *Nachbarschaft* die durch die Struktur bestimmt wird. Potentielle Partner werden nur innerhalb der lokalen Nachbarschaft gesucht. Abbildung 2.12 zeigt verschiedene Arten von Nachbarschaften in einer 2D Gitterstruktur.

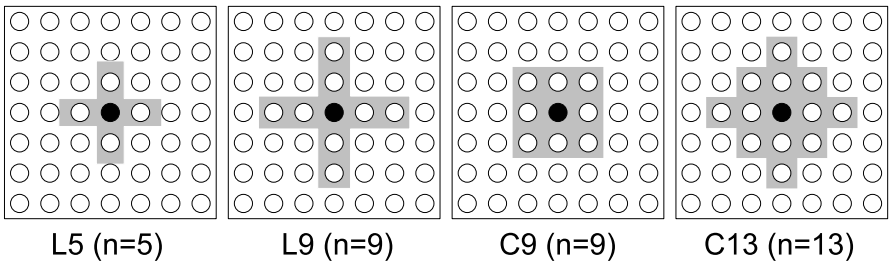


Abbildung 2.12: Nachbarschaftsformen in 2D Gitterstruktur nach [207]

Die von Sarma [207] untersuchten Nachbarschaftsstrukturen sind unter Anderem ein Kreuz mit Abstand eins (L5), Abstand zwei (L9), Stern mit Abstand eins (C9) und schiefer Stern mit Abstand zwei (C13). Auf jedem Knoten befinden sich ein Individuum und ein EA mit Populationsgröße eins. Bei der Selektion werden die Knoten und deren Individuen innerhalb der Nachbarschaft betrachtet. Das neue Individuum ersetzt das alte. Die überlappenden lokalen Nachbarschaften verhalten sich wie ein globaler Migrationsmechanismus im Grid. Es sind für jede beliebige Struktur Nachbarschaften vorstellbar. In Ringen (die Population ist wie ein Ring angeordnet) erstreckt sich die Nachbarschaft entlang des Ringes. In Netzwerken sind alle mit einer vorgegebenen Anzahl von Schritten erreichbaren Knoten Teil der Nachbarschaft.

Die Größe und Form der Nachbarschaften beeinflussen die Migration und damit den Fluss genetischer Informationen durch das Grid. Diese Form der Performancebetrachtung wird *Takeover time* genannt und wird im folgenden Abschnitt 2.3.6 beschrieben. Die lokal verwendete Selektionsmethode kann beliebig wie bei globaler Selektion auch verwendet werden. Dazu zählen Tournament Selektion [35, 86], $\mu - \lambda$ Selektion [215] und rangbasierte Selektion [9, 10].

Forschungsergebnisse von Sarma und De Jong [207] zeigen für 2D Gitterstrukturen qualitativ den gleichen Selektionsdruck wie für globale Selektion. Quantitativ ist der Selektionsdruck allerdings schwächer, da Zeit für das Propagieren der Lösung durch die Gitterstruktur benötigt wird. Der Durchmesser der lokalen Nachbarschaft spielt hierbei eine entscheidende Rolle.

2.3.6 Takeover time

Unterschiedliche Topologien in Populationsnetzwerken haben direkten Einfluss auf den Informationsfluss zwischen den Individuen und damit auf die Ausbreitung genetischer Informationen. Die Topologie verteilter evolutionärer Algorithmen leitet den Fluss genetischer Informationen und damit den Selektionsdruck innerhalb der Population. In klassischen EA arbeitet der Selektionsoperator auf der gesamten Population, wodurch $n : n$ Interaktionen möglich sind. Dadurch verschwinden weniger fitte Allele sehr schnell zu Gunsten vorteilhafter Variationen [188]. Demgegenüber stehen cEA mit sehr begrenzter Nachbarschaft. In diesen Topologien breiten sich erfolgreiche Allele sehr viel langsamer aus [85, 204, 207]. Die Vorteile liegen darin, dass die Gesamtpopulation langsamer konvergiert und die Diversität länger erhalten bleibt (siehe hierzu auch den Abschnitt Selektion 2.3.3).

Eine Methode zur Quantifizierung der Geschwindigkeit, mit der sich genetische Informationen in Populationen ausbreiten ist die sogenannte *Takeover time*[13] (deutsch Übernahmezeit) [89]. Hiermit lässt sich der Selektionsdruck innerhalb von Populationen quantifizieren und verschiedene Selektionsoperatoren und -methoden vergleichen.

Gegeben sei eine Zufallsvariable $V_i(t) \in \{0, 1\}$, welche das Vorhandensein des besten genetischen Codes in Knoten $i|1 \leq i \leq n$ ausdrückt durch $V_i(t) = 1$ bzw. eines schlechteren genetischen Codes durch $V_i(t) = 0$ zum Zeitpunkt t darstellt. n bezeichnet die Populationsgröße. Die Zufallsvariable

$$N(t) = \sum_{i=1}^{n} V_i(t) \tag{2.2}$$

bezeichnet die Anzahl der Kopien des besten genetischen Codes innerhalb der Population zum Zeitpunkt t. Initial ist genau ein bestes Individuum vorhanden ($V_i(1) = 1 \wedge V_j(1) = 0$ für alle $j \neq i$). Damit ergibt sich nach Rudolph [204] folgende Definition:

Definition 2 (Takeover time)**.**
Die Erwartungswert $E(T)$ mit $T = min\{t \geq 1 : N(t) = 1\}$ bezeichnet die Takeover time der Selektionsmethode über der Population.

Die Takeover time gibt die minimale Anzahl von Schritten an, welche für die Übernahme der gesamten Population durch Kopien des besten genetischen Codes notwendig sind. Anhand Definition 2 wird die Zeit (bzw. die Anzahl der Schritte) gemessen, bis eine Population vollständig aus Kopien des besten Individuums besteht. Die untersuchten Graphenstrukturen der Takeover time sind nach [204] ungerichtete Graphen. Die Wahrscheinlichkeit der Auslöschung für Kopien der Gene des besten Individuums wird mit null angenommen. Ist dies nicht der Fall, wird die Definition der Takeover time selbst problematisch, da z.B. das initial beste Individiuum ausgelöscht werden kann [204]. Abbildung 2.13 zeigt die Informationsausbreitung in einem 7×7 Gitter für zwei unterschiedliche Nachbarschaften (grau hinterlegt) nach [207]. Die verzerrenden Effekte von Rekombination und Mutation werden durch ausschließliche Verwendung von Selektion vermieden. Hohe Takeover times sind damit ein Hinweis auf geringen Selektionsdruck und umgekehrt deutet eine niedrige Takeover time auf hohen Selektionsdruck hin. Die Takeover time für Nachbarschaft $L5$ beträgt 6 und für Nachbarschaft $C9$ lediglich 3 Schritte. Damit ist der Selektionsdruck für $L5$ kleiner als bei $C9$.

[13] Aufgrund der ausschließlichen Verwendung des englischen Begriffes Takeover time in der Literatur wird in dieser Arbeit die englische Version beibehalten.

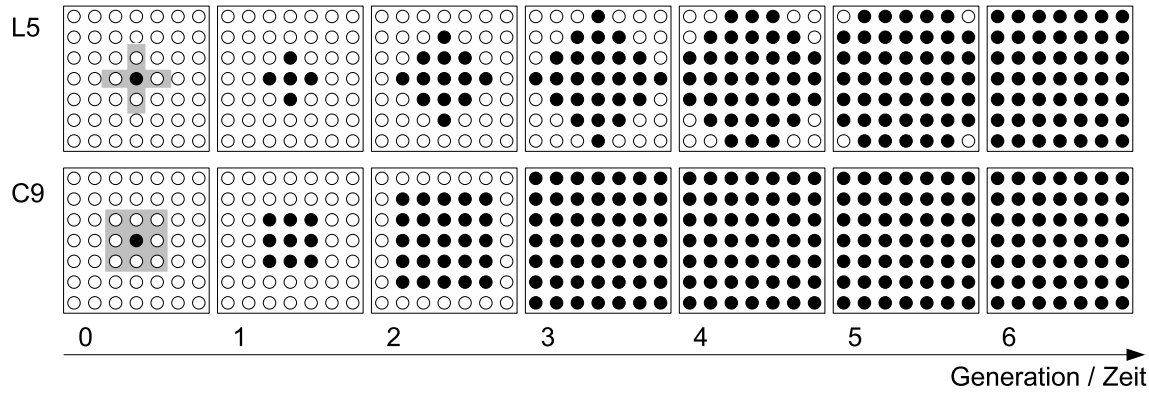


Abbildung 2.13: Informationsausbreitung mit unterschiedlichen Nachbarschaften (im linken Bild jeweils hinterlegt) nach [207] in einem 7×7 Gitter

Logistisches Modell

Obwohl die Takeover time für klassische EA vielfach untersucht wurde, sind Analysen und Modelle für parallele EA noch relativ neu. Sarma und De Jong [207] präsentierten im Jahr 1996 ein logistisches Modell und führten detailierte empirische Analysen für verschiedene Nachbarschaftsgrößen und -formen für synchrone $N \times N$ cEA durch. Dieses relativ einfache Modell mit logistischem Wachstum wurde vielfach aufgegriffen und erweitert für dEA, Ringstrukturen [85] oder asynchrone Updates [3]. Gorges-Schleuter [93] untersuchte die logistischen Wachstumskurven numerisch unter der Annahme einer unendlichen Population. Giacobini [84] führt Untersuchungen zur Takeover time in zufälligen Netzwerken durch und betrachtet Graphen als Verteilung von Individuen und Kanten. Als ein Resultat stellt sich heraus, dass zufällig strukturierte Populationen und panmictische Populationen etwa dem selben Selektionsdruck unterworfen sind. Das Logistische Modell von Giacobini berechnet wie folgt die Erwartungswerte $E[N(t)]$ für die Anzahl der Individuen in der übernommenen Population N zum Zeitpunkt t:

$$N(t) = \begin{cases} N(0) = 1 \\ E[N(t)] = E[N(t-1)] + (n - E[N(t-1)])\frac{E[N(t-1)]}{n}, \end{cases} \tag{2.3}$$

wobei n die Anzahl der Knoten ist und von einem synchronen Update aller Knoten in diesem logistischen Modell ausgegangen wird. Zum Start existiert genau ein Individuum mit dem fittesten Erbgut ($N(0) = 1$). Weiterhin werden zusammenhängenden Graphen vorausgesetzt, auch wenn die Struktur schwach vernetzt ist. Die Resultate werfen die Frage auf, wie sich die Takeover time für unzusammenhängende Graphen beschreiben lässt. Sind diese statisch, wird nie der gesamte Graph übernommen:

$$\lim_{t \to \infty} E[N(t)] < n$$

Für beliebig große t gilt damit obiger Ausdruck. Diese Arbeit wird der Frage nachgehen und ein Modell aufzeigen, welches Abschätzungen für $E[N(t)]$ auch für unzusammenhängende Graphen erlaubt.

Hypergraph Modell

Ein weiteres Modell ist das *Hypergraph Modell*. Es handelt sich hierbei um eine Generalisierung von Kanten als ein Paar von Knoten hin zu Hyperkanten als Verbindung beliebiger Teilmengen von Knoten. Sprave [224] stellte 1999 eine einheitliche Beschreibung für Populationsstrukturen mittels Hypergraphen vor. Die Methode basiert auf der Berechnung des sogenannten *Populationsdurchmessers* (engl. *population diameter*) und der aktuellen Populationsstruktur. Chakraborty et al. [45] und Alba und Luque [3] zeigten für ringförmige dEA Strukturen dass das Hypergraph Modell genauere theoretische Vorhersagen bei gegebener Migrationsrate und -frequenz liefert.

Generell zeigt sich, dass ein- bzw. zweidimensionale Gitterstrukturen gegenüber klassischen EA den Selektionsdruck verringern und damit das explorative Potential evolutionärer Suche erhöhen [85]. Die Untersuchung zufälliger Graphen durch Newman [175] und nicht regulärer Topologien durch Giacobini et al. [84] zeigten ähnlichen Selektionsdruck wie klassische EA. Vergleichbares zeigte die Untersuchung in *skalenfreien Netzwerken* (engl. *scale-free network*) , solange das beste Individuum initial zufällig positioniert wurde. Andererseits bewirkt die Positionierung des besten Individuums in einem sogenannten Hub-Knoten (am meisten vernetzte Knoten) eine starke Reduktion der Takeover time.

Untersuchungen zu heterogenen Netzwerken mit unterschiedlicher Berechnungsgeschwindigkeit der Knoten, dynamische Netzwerkstrukturen und auch sehr schwach vernetzten Knoten werden von aktuellen Arbeiten nicht berücksichtigt. Giacobini [84] geht explizit von verbundenen Graphen aus, auch bei schwacher Vernetzung. Insbesondere sind daher Modelle von Interesse, die Aussagen über nicht zusammenhängende Graphen machen und daher auch Teilpopulation betrachten können.

2.3.7 Abgrenzung

In diesem Abschnitt werden weitere verteilte Optimierungsverfahren beschrieben und eine Abgrenzung vorgenommen.

2.3.7.1 Distributed Problem Solving

Distributed Problem Solving (DPS, zu deutsch: verteiltes Problemlösen) betrachtet sogenannte Problemlöser, die gemeinsam Aufgaben bearbeiten, deren Erfüllung weit über die Fähigkeiten eines einzelnen Problemlösers hinausgehen. In diesem Kontext wird auch häufig der Begriff *Cooperative Distributed Problem Solving* verwendet (CDPS) [56]. Problemlöser oder oft auch einfach Agenten arbeiten geschlossen zusammen und kennen die Kompetenzen der jeweils andern genau. Die Agenten wurden im Hinblick auf Zusammenarbeit in bestimmten Problemen entwickelt und in DPS werden die Existenz eines konkreten Problems und auch dessen Lösungswege vorausgesetzt. Im Prozess der Bearbeitung erfolgt manchmal die Erarbeitung eines Planes. Damit ist der Begriff des *verteilten Planens* und entsprechende Methoden eng mit DPS verwandt.

Der erste Schritt beim DPS ist die Aufteilung und Verteilung von Aufgaben, das sogenannte *Task Sharing*. Die Idee: wenn ein Agent viele Aufgaben zu tun hat, wird ein Teil der Aufgaben auf andere Agenten mit weniger oder keinen Aufgaben verteilt. Die Schritte sind folgende [56]:

1. **Aufgabenzerlegung**: Eine Liste von Aufgaben, die an andere Agenten delegiert werden

können, wird erstellt. Hierzu werden unter anderem auch Aufgaben in Teilaufgaben zerlegt, die an unterschiedliche Agenten weitergegeben werden.

2. **Aufgabenallokation**: Zuweisung, bzw. delegieren der Aufgaben an geeignete Agenten.

3. **Aufgabenausführung**: Agenten erledigen die (Teil)Aufgaben. Dies kann auch rekursives kooperatives Problemlösen sein, d.h. Teilaufgaben werden weiter zerlegt und delegiert.

4. **Synthese der Resultate**: Die Ergebnisse werden an den Verteiler zurückgegeben und dort vereinigt.

Die einzelnen Schritte können je nach Problem einfach oder sehr aufwändig sein. Zur Abarbeitung dienen unterschiedliche Mechanismen der Kooperation. Häufig wird das in Abschnitt 2.2.5 beschriebene Contract-Net Protokoll verwendet und die Tasks an geeignete Bearbeiter ausgeschrieben. Soweit der Manager im Sinne der Aufgabenverteilung geeignete Bearbeiter findet, sind keine weiteren Koordinationsprozesse erforderlich. Sind jedoch keine Agenten willens oder verfügbar, so werden alternative Methoden verwendet, z.B. Ausschreibung an alle Agenten (Broadcast Contracting), Wiederholung nach einiger Zeit, die Auswahl geeigneter Bearbeiter erweitern oder alternative Zerlegungen für die Aufgabe finden.

Nach Verteilung der Aufgaben werden die Resultate gegebenenfalls kombiniert und in die übergeordnete Lösungen eingearbeitet, auch *Result Sharing* genannt. Hierbei unterscheiden sich die Kontexte von Ausschreiber und Bearbeiter. Auch zwischen verschiedenen Agenten muss die gleiche Aufgabe nicht immer das gleiche Resultat hervorbringen. Verschiedene Wege existieren, um die Gesamtqualität und -performance zu verbessern [56]:

- Paralleles Bearbeiten der gleichen Aufgabe durch mehr als einen Problemlöser und abschließender Vergleich der Resultate erhöht die Wahrscheinlichkeit, dass die Ergebnisse korrekt sind.

- Agenten versuchen neben den zugewiesenen Aufgaben auch weitere Aufgaben anhand ihrer Kompetenz zu lösen. Dabei werden Lösungen gegenseitig ergänzt und hierdurch vervollständigt.

- Zur Weiterentwicklung eigener Lösungen tauschen Agenten Informationen über ihre jeweiligen Lösungen aus. Hierbei entstehender Wissenstransfer kann neue Erkenntnisse bei der Präzisierung eigener Lösungen bringen.

- Auch wenn ein Agent die Kompetenz zur Lösung einer aufwändigen Aufgabe selbst besitzt, so kann er im Sinne der schnelleren Abarbeitung die Aufgabe zerlegen und Teile davon delegieren.

Um Aufgabenzerlegung, -delegation und -kombination effizient zu erledigen, bedarf es der sogenannten verteilten Planung (*Distributed Planning*). Im Prinzip kann die verteilte Planung ebenfalls als verteiltes Problemlösen betrachtet werden mit dem Spezialproblem der Planerstellung. Aufgrund der speziellen Natur der Planung ist jedoch die Verwendung gesonderter Methoden sinnvoll. Durfee at al. unterschiedet hierbei folgende Techniken:

- Zentrales Planen für verteilte Pläne

- Verteiltes Planen für zentrale Pläne
- Verteiltes Planen für verteilte Pläne

Insbesondere die letzte der genannten Techniken ist sehr anspruchsvoll und die Literatur hierzu ist sehr vielschichtig. Besondere Beachtung verlangt die Repräsentation, Planformulierung und Ausführung. Zur weiteren Lektüre sei auf [55, 56, 57] verwiesen.

Im Kontext dieser Arbeit sind einige Besonderheiten hinsichtlich DPS zu beachten. Durfee und Rosenschein [55, 56, 57, 58] gehen von den Annahmen aus, dass Agenten grundsätzlich zur Kooperation bereit sind, dass gemeinsame Ziele vorherrschen und dass die Verhaltensweisen a priori festgelegt sind. Dazu schreibt Durfee in [56] S. 121:

> "[...] the agents have been designed to work together; or the payoffs to self-interested agents are only accrued through collective efforts: or social engineering has introduced disincentives for agent individualism [...] "

und weiter zur Kooperationsfähigkeit ([56] S. 121):

> "[...] group coherence is hard to realize among individually-motivated agents [...]"

Obwohl die Gründe, die zur Entstehung von DPS geführt haben ähnlich den Gründen dieser Arbeit sind, so ist der beschriebene Ansatz unterschiedlich. Die in dieser Arbeit angenommene Autonomie und Nutzenmaximierung der einzelnen Akteure macht die Anwendung von DPS unmöglich. Weiterhin ist bei grundsätzlich geforderter Adaptivität eines Gesamtsystems ein vorgefertigtes Lösungsmuster als nicht ausreichend anzusehen. Aus den genannten Gründen wird dieser Ansatz nicht weiter verfolgt.

2.3.7.2 Schwarmintelligenz

Schwarmintelligenz basiert auf dem kollektiven Verhalten einzelner Agenten. Typischerweise gibt es eine Population lokal interagierender Agenten. Diese Einzelindividuen folgen einfachen Regeln, während das Gesamtverhalten sehr komplex werden kann.

Ameisenalgorithmen

Ein Repräsentant im Kontext von Schwarmintelligenz sind sogenannte *Ameisenalgorithmen* (engl. *Ant Colony Optimization - ACO*) [47, 54]. Auf Basis sogenannter *Pheromone* (Duftstoffe) und einfachen Verhaltensregeln entstehen die Wege von Ameisen, welche nach und nach zum kürzesten Weg konvergieren. Wann immer eine Ameise Futter zum Nest trägt, hinterlässt sie eine Pheromonspur deren Stärke sich nach der Qualität der Futterquelle richtet. Andere Ameisen werden von einer solchen Pheromonspur in ihrem Verhalten beeinflusst, d.h. ihre Entscheidungen basieren auf vorhandenen Pheromonspuren. Wann immer eine Ameise sich entscheidet welcher Weg zu gehen ist, richtet sich die Wahrscheinlichkeit primär nach der Stärke des Pheromons. Pheromone verflüchtigen sich nach einer bestimmten Zeitspanne von selbst und ermöglichen damit Vergessen altenr Wege und das Lernen neuer Wege. Für zwei unterschiedliche Wege zum selben Ziel gilt damit, dass der kürzere Weg schneller von der Quelle zum Nest führt und daher Ameisen diesen Weg im Vergleich zum längeren Weg während einer Zeitspanne öfter durchlaufen. Hierdurch wird die Pheromonspur stärker als beim kürzeren Weg und mehr Ameisen folgen dem kürzeren Weg.

Ameisenalgorithmen wurden ursprünglich Anfang der 90er Jahre für Kombinatorische Probleme genutzt [54]. Das Problem des Handlungsreisenden (TSP - Traveling Salesperson Problem) [227] ist ein Beispiel für erfolgreiche Anwendung. Ebenso sind Anwendungen in Telekommunikationsnetzwerken zum Routing entstanden. Aufgrund der vorrangigen Ausrichtung auf Routing und der oft engen Interaktion der Ameisen wird dieser Ansatz nicht weiter verfolgt.

Partikelschwarmoptimierung

Ein weiterer Vertreter von Schwarmintelligenz ist die *Partikelschwarmoptimierung* (engl. *Particle Swarm Optimization - PSO*) als globale Optimierungsverfahren. Hierbei steht jedes Partikel für eine Punkt oder eine Fläche im einem N-dimensionalen Suchraum. Die Partikel bewegen sich durch den Suchraum und orientieren sich an anderen (üblicherweise besseren) Partikeln. PSO wurde 1995 erstmalig beschrieben von James Kennedy und Russel Eberhart [127] und hat sich seitdem stark weiterentwickelt und eine Reihe von Verfeinerungen erhalten. Der Anwendungsbereich ist mittlerweile breit gefächert und reicht vom Antennenentwurf, Funktionsallokationen in Fahrzeugnetzwerken [71] bis hin zur Verwendung in Computer Grids [164].

2.3.7.3 Artificial Life

Artificial life (künstliches Leben) [1, 37, 156] beschäftigt sich mit der Erforschung von Leben jenseits biologischer Grundlagen. Das Ziel ist die Untersuchung von Leben oder lebensähnlichen Prozessen durch Simulationen und Computermodelle. Fragestellungen umfassen auch die Erschaffung von Leben mit nicht-biologischen Grundlagen z.B. im Computer.

Die Simulationen werden oft als agentenbasierte Simulationen durchgeführt. Hierbei sind Agenten zumeist sehr einfach strukturierte Elemente, die mit dem Verständnis des Agenten in dieser Arbeit (siehe Abschnitt 2.2 (Weak Notion of Agency)) nicht viel gemeinsam haben. So können Agenten zum Beispiel Zellen eines Organismus oder soziale staatenbildende Insekten verkörpern. Eine weitere Methode zur Untersuchung und ein verwendetes Erklärungsmodell sind evolutionäre Verfahren. Diese stellen einen Grundbaustein biologischen Lebens dar und bilden daher dieses Prinzip entsprechend am Computer ab. Weitere Techniken umfassen Neuronale Netze und zelluläre Automaten [162]. Auch wird in vielen Ansätzen eine Umgebung vorausgesetzt, die Energie bereithält und Untersuchungen zu Stoffwechsel ermöglicht.

Die Modelle zeigen oft trotz ihrer Einfachheit komplexes Verhalten und helfen, wichtige Fragestellungen im Zusammenhang mit Evolution zu erklären. Im Rahmen dieser Arbeit werden verschiedene Aspekte im Zusammenhang mit Artificial life aufgegriffen (Agenten, evolutionäre Verfahren, Umgebungen). Es ist jedoch nicht die Intention dieser Arbeit, Leben zu erklären oder die Forschung auf dem Gebiet künstlichen Lebens zu erweitern.

2.3.8 Zusammenfassung

In diesem Abschnitt wurden parallele evolutionäre Verfahren beschrieben. Durch ihre implizite Parallelität stellen sie eine ideale Technologie für verteilte Systeme dar.

Neben Darstellung allgemeiner Eigenschaften wurde die in der Literatur weitgehend akzeptierte Klassifikation in drei Klassen aufgeteilt: Master-Slave, Insel-Modell und Diffusions-Modell. Eine Taxonomie zur Einordnung der genannten Hauptklassen betrachtet die Anzahl der Subpopulationen, deren Größe und die Interaktionshäufigkeit. Diese Taxonomie wird als

strukturierte Population bezeichnet. Die 1 : 1 Umsetzung strukturierter Populationen auf Hardware ist denkbar, doch lassen sich alle Ansätze auf nahezu jede Hardware abbilden (virtualisieren). So ist z.B. das Diffussionsmodell auf nur einem Prozessor denkbar. Die Vorteile liegen in veränderter Migration von Lösungen durch das System und z.B. in der Exploration mehrerer Optima. Die Anwendung verteilter EA auf beliebigen Netzwerkstrukturen ist noch sehr neu und es existieren dazu wenige Untersuchungen. Alba hat daher als zukünftige Themen die Anwendungen in Grid und Peer-to-Peer Netzwerke identifiziert [80]

Weitere Untersuchungen zur Takeover time haben gezeigt, dass heterogene Netzwerke mit unterschiedlicher Rechengeschwindigkeit der Knoten, dynamische Netzwerkstrukturen und auch sehr schwach vernetzten Netzwerke von aktuellen Arbeiten nicht berücksichtigt wurden. Giacobini [84] geht explizit von verbundenen Graphen aus, auch bei schwacher Vernetzung. Insbesondere sind daher Modelle von Interesse, die Aussagen über nicht zusammenhängende Graphen machen und daher auch Teilpopulation betrachten können.

2.4 Agentenbasierte Evolutionäre Verfahren

In der Literatur existiert eine Vielzahl an Arbeiten zu evolutionären Agenten oder Agenten im weiteren Kontext evolutionärer Verfahren. Der Fokus dieser Arbeit liegt in der Untersuchung verteilter Optimierung mit Hilfe proaktiver und autonomer Agenten (siehe Abschnitt 2.2). Damit erfolgt eine Einschränkung der betrachteten Literatur. Arbeiten, die folgende Kriterien erfüllen, wurden von weiterer Betrachtung ausgeschlossen:

1. Nutzung panmiktischer evolutionärer Verfahren mit zentral verwalteter Population. Der Begriff Agent wird in diesem Zusammenhang gleichsam als konzeptueller 'Container' für Individuen verwendet.

2. Nutzung evolutionärer Verfahren, wobei evolutionäre Operatoren nicht vom Agent selbst kontrolliert oder ausgeführt wurden. Hierbei handelt es sich, wie in Punkt 1 angeführt, bei den Agenten nicht um selbständig agierende Systemelemente. Stattdessen besteht eine Abhängigkeit zu einer externen evolutionären Instanz.

3. Artificial life Ansätze, deren Fokus auf der Erstellung von Erklärungsmodellen für Leben, künstliches Leben oder dafür benötigte Prozesse liegt (siehe auch Abschnitt 2.3.7.3).

4. Artificial life Ansätze, deren Optimierungsprozess ohne Ziel bzw. ohne Intention eines Systemdesigners verläuft. Die Erklärung biologischer Prozesse ist nicht das Anliegen dieser Arbeit.

Als Vorläufer evolutionärer Agenten kann das von Holland beschriebene *Learning Classifier System* (*LCS*) [113] betrachtet werden, die wiederum auf ältere Arbeiten verweisen [111]. Hierbei handelt es sich um ein adaptives Regelsystem im Sinne verstärkendem Lernen im Kontext von Agenten.

Das Regelsystem besteht aus einer Menge von 'Wenn - Dann' Regeln. Systemreize werden mit dem 'Wenn' Teil der Regel abgeglichen und bei Erfüllter Bedingung wird die Aktion des 'Dann' Teiles ausgeführt. Aktionen können sowohl in externe Aktionen münden, als auch intern weitere Regeln triggern. Wann immer die Umgebung eine Belohnung für erfolgreich

ausgeführte Aktionen (engl. *payoff*) bereitstellt, so geht dieser payoff an die aktive Regel. Intern ist jeder Regel eine Fitness zugeordnet, die die Summe der gesammelten payoffs darstellt. Erfolgreiche Regeln geben Teile des payoffs an diejenigen Regeln weiter, die zur Aktivierung geführt haben. Üblicherweise läuft der Prozess der Aktivierung von Regeln so ab, dass mehrere Regeln darum konkurrieren, ihre Aktionen auszuführen. Dies läuft im Bieterverfahren (engl. *credit assignment*) ab. Regeln bieten einen bestimmten Teil ihrer Fitness während dieses Auswahlprozesses. Damit wirken Regeln wie Hypothesen für die Zukunft, deren Eintrittswahrscheinlichkeit an der Höhe des Gebots abzulesen ist. Regeln mit hoher Fitness deuten auf erfolgreiche Vorhersagen hin. Je höher das Gebot, umso höher ist die Wahrscheinlichkeit, dass eine Regel den Auswahlprozess gewinnt. Von Zeit zu Zeit werden die Regeln anhand ihrer Fitness durch einen genetischen Algorithmus verändert, um Adaption des Regelsystems zu ermöglichen.

Die Einbettung dieses *verstärkenden Lernverfahrens* in den Agentenkontxt bezeichnet Holland als adaptive Agenten [113], wenngleich Hollands Agentenbegriff abweichend vom Agentenbegriff dieser Arbeit verwendet wird. Das von Holland angestrebte Ziel war die Beschreibung komplexer adaptiver Systeme und daher erfolgte kein Nachrichtenaustausch zwischen Agenten. Vielmehr wurden Agenten im *Echo* System gesteuert und sind damit nicht im Sinne von kommunikationsfähigen Agenten zu sehen. Darüber hinaus arbeitete der genetische Algorithmus auf einer Population von Regeln innerhalb eines Agenten. Agenten sind somit als evolutionär nicht interagierende Elemente anzusehen. Diese Funktionsweise wird wird auch als sogenannter 'Pittsburgh-Typ' bezeichnet. Eine Reihe weiterer Varianten von LCS entstand, die sich in verschiedenen Punkten unterscheiden (z.B. mehrere Regelpopulationen anstelle von einer Einzigen, Representation der Regeln, Fitnessdefinition der Regeln).

Basierend auf den Arbeiten von Holland [111] und Kapsalis et al. [123] beschreiben Smith und Taylor in [221] das *Egglet Framework* für evolutionäre Verfahren in Agenten-basierten Systemen. Hierbei wurde das zentralisierte GA Design entfernt und sowohl die genetischen Informationen als auch die Operatoren zur Reproduktion (Selektion, Rekombination, Mutation) in den Agenten (=Egglet) verlagert. Agenten erhalten die Möglichkeit, neue Agenten zu erzeugen und lokal nach Partnern zu suchen. Während der Partnersuche werden sogenannte `Plumage` Objekte verwendet, die einerseits genetische Informationen und andererseits eine vom Sender 'selbst deklarierte Fitness' enthält. Die Fitness ist problemabhängig. Im Beispiel wurde das OneMax Problem[14] bearbeitet und als Fitness dient die Anzahl von Einsen. Hat ein Agent genügend `Plumage` Objekte gesammelt, so sucht er sich aus diesen das mit der höchsten Fitness aus, erzeugt ein Kind und stirbt (steady-state Verhalten).

Einige Punkte dieses Ansatzes machen einen Einsatz im Hinblick auf die Zielstellungen dieser Arbeit schwierig. Hierbei fällt auf, dass eine Einbettung innerhalb verteilter dynamischer Systeme problematisch ist. Ein Punkt betrifft die Verwendung einer speziellen, auf das Problem zugeschnittenen Fitnessfunktion (OneMax). Damit ist die Ausgestaltung der Fitnessfunktion lokal im Agenten a priori fixiert und verhindert Anpassung an dynamische Problemstellungen. Ebenso wird der Einsatz als optimierende Systembestandteile erschwert durch das steady-state Verhalten des eingesetzten Algorithmus. Es ist somit immer die gleiche Anzahl Agenten vorhanden und die populationsbasierte Dimension von Adaptivität fehlt. In der Tat würde das vorgestellte Reproduktionsverhalten zu einer beständigen Erhöhung führen, wenn Agenten nach Reproduktion nicht sterben. Ein weiterer nicht beachteter Aspekt ist fehlende Kollaboration zwischen den Agenten. Insgesamt stellt die Agentenpopulation einen

[14] Maximierung der Anzahl an Einsen in einem Bitstring [209]

evolutionären Adaptionsmechanismus dar, der ohne Stimuli der Umgebung (Anreize, Anpassung)das Problem isoliert bearbeitet.

Eine Weiterentwicklung zur Vereinigung ökonomischer und biologischer Prinzipien untersucht Smith et al. in [220] im Avalance Projekt [65] und Eymann verwendet diesen Ansatz in [64] zur Realisierung einer dezentralen Wertschöpfungskette. Hierbei wird ein dezentraler Marktplatz mit elektronischen Verhandlungen abgebildet. Die Wertschöpfungskette besteht aus Holzfällern, die Holz an Zimmermänner verkaufen, die wiederum Tischler beliefern. Schließlich werden fertige Tisch an Kunden verkauft. Jeder der unterschiedlichen Rollen wird durch einen Agententyp repräsentiert. Um gemeinsam das Produkt 'Tisch' herzustellen, müssen die Agenten miteinander kollaborieren. Die Agenten verhandeln untereinander über Angebote und Gegenangebote versuchen damit jeweils unter ökonomischen Gesichtspunkten gute Preise zu erzielen. Dabei wird die Verhandlungsstrategie durch evolutionäre Prozesse auf ihrer Strategie verändert. Zur Strategie zählen insgesamt sechs Verhandlungsparameter, wie z.B. 'Habgier' (Wahrscheinlichkeit von Zugeständnissen) und 'Zufriedenheit' (Wahrscheinlichkeit der Verhandlungsfortsetzung. Die Agenten sind kollaborativ in ein *Informationsökosystem* eingebettet und ihre Fitness ist Geld. Ein Agent, der im Vergleich zu anderen schneller handelt oder die bessere Verhandlungsstrategie besitzt, erzielt ein höheres Einkommen und ist damit fitter. Damit ist eine problemunabhängige interne Bewertung des Erfolgs gegeben. Agenten zahlen Steuern für Speicherplatz, Bandbreite und CPU Zeit. Der Plumage Mechanismus von [221] wurde übernommen, jedoch das steady-state Verhalten nicht. Agenten können bankrott gehen, wenn ihr Geld aufgebraucht ist. Andererseits sind Klone erfolgreicher Agenten möglich. Die Erklärung ist in der mikroökonomischen Theorie verortet: erfolgreiche Unternehmen finden schnell Nachahmer, die leicht veränderte Strategien nutzen. Rekombination lässt sich in diesem Zusammenhang durch die Kombination beobachteter und erfolgreicher Strategien mit bereits bestehenden Strategien erklären.

Die Intention und die Ergebnisse wurden zur Erklärung und Herbeiführung dezentraler Koordinationseffekte verwendet. In diesem Falle steht nicht das Systemverhalten im Vordergrund, sondern die mikroökonomische Anpassung einzelner Wirtschaftssubjekte und Preise innerhalb des Systems. Die Ergebnisse sind bezüglich der Erzielung eines dezentralen und systemweiten Konsens bemerkenswert. Es entsteht emergent eine marktbasierte Ordnung auf Basis individueller Nutzenmaximierung. Hinsichtlich der Problemstellung der vorliegenden Arbeit addressiert Eymann in [64] den Aspekt der Konsenserreichung trotz Dezentralisierung von Informationen. Ebenso ist durch die ökonomischen Betrachtungen ein wirkungsvolles Mittel zur dezentralen Koordination beschrieben. Damit verbleibt als ein offener Punkt die Untersuchung der Effekte auf das Gesamtverhalten des Systems. Dezentrale Optimierungselemente könnten diese Koordinationsmechanismen zur Ableitung von Aktionen zur Beeinflussung des Gesamtsystemverhaltens auszunutzen. Eine weitere nicht gelöste Frage beinhaltet notwendige Reproduktionskriterien vor den Hintergrund des vorgestellten ökonomischen Mechanismus.

Eymann [64], Holland [113] und Menczer [156] verwenden von der Umgebung in Form von Kunden bzw. als Belohnung bereitgestelltes Zahlungsmittel. Damit werden zwei wichtige Zwecke erfüllt. Einerseits sind Agenten damit in der Lage, erfolgreiche Aktionen von wirkungslosen Aktionen im Hinblick auf eine externe Instanz zu unterscheiden. Dies ermöglicht Lernvorgänge und ist ein wichtiger Baustein hin zur Adaptivität von Systemen an veränderliche Umgebungen. Holland und Eymann gehen noch einen Schritt weiter und verwenden das Zahlungsmittel als 'Fluss' (Flow) [113] nicht nur zwischen Umgebung und Agent, sondern auch zwischen den Agenten. Holland schreibt über den Fluss:

> "[...] the nodes are processors - agents - and the connectors designate the possible interactions. In cas the flows through these networks vary over time; moreover, nodes and connections can appear and disappear as the agents adapt or fail to adapt. Thus neither the flows nor the networks are fixed over time. "

Damit kann der Fluss von Zahlungsmitteln als Katalysator erfolgreicher Agenten und deren systemrelevanter Beziehungen und Strukturen innerhalb von CAS angesehen werden. Holland schreibt über die Eigenschaften folgendes:

> "There are two properties of flows, well known from economics, that are important [...] The first of these is the multiplier effect [...], which occurs of one injects additional resource at some node. Typically the resource is passed from node to node, possibly being transformed along the way, and produces a chain of changes [...]. The simplest examples come from economics. When you contract to build a house, you pay the contractor, who pays the tradesmen, who in turn buy food and other commodities, and so on, stage by stage through the economic network. [...] The second property is the recycling effect - the effect of cycles in the networks. "

Damit betrachtet Holland Energie- und Geldströme als gleichwertig im Sinne des Flusses. Durch die Weitergabe und das Wiederverwenden im Netzwerk erlaubt der Fluss bedarfsgerechte Verteilung von Ressourcen in komplexen Netzwerkstrukturen. Dabei ist nicht die Struktur entscheidend für den Fluss, sondern der Bedarf einzelner Knoten. Die mikroökonomische Nutzung des Fluss-Konzeptes zur Anpassung komplexer Systeme auf äußere und innere Anforderungen bedarf weiterer Untersuchung im Hinblick auf deren Verwendung in Optimierung in CAS.

Untersuchungen lokaler Selektion und die Verwendung eines Schwellenwertes zur Reproduktion θ wurden von Menczer [156] durchgeführt. Hierbei wurde das Fluss-Konzept teilweise angewendet, indem Agenten Energie aus der Umgebung aufnehmen und als Entscheidungsgrundlage ihrer Reproduktionsfähigkeit verwenden. Die Anwendung des Ansatzes wurde unter anderem zur Suche in Graphen verwendet. Agenten bewegen sich durch einen Graph anhand ihrer Suchstrategie und nutzen dabei die jeweiligen Kanten, um sich im Graphen zu bewegen. Hierbei gibt die Umgebung bei erfolgreicher Suche (Knoten wurde vorher noch nicht besucht) eine Belohnung in Form von Energie an die Agenten ab. Diese lokale Energie bezeichnet Menczer als *endogene Fitness*. Diese sammeln anhand ihrer Suchstrategie Energie. Sobald die Menge an Energie die sogenannte *Reproduktionsschwelle* überschritten wurde, erfolgt eine Reproduktion und die Energie wird mit dem erzeugten Kind geteilt. Ist keine Energie vorhanden, so stirbt der Agent ähnlich wie im Ansatz von Eymann [64]. Die Reproduktionsschwelle wird als durchschnittliche Energie aller Agenten festgelegt. Damit multiplizieren sich Agenten mit überdurchschnittlicher Energie in der Agentenpopulation. Die Anzahl der Agenten wird von der Tragfähigkeit der Umgebung bestimmt. Dabei ist die bereitgestellte Energiemenge der limitierende Faktor. Die lokale Selektion ist in diesem Ansatz ein Mechanismus, die Agentenpopulation an die Tragfähigkeit der Umgebung anzupassen (Menczer [156]) und dient weniger der Optimierung, da kein Austausch genetischen Materials stattfindet. Stattdessen erfolgt die Selektion zur Reproduktion (Split+Mutation) allein anhand der lokal vorhandenen Energiemenge.

Die hier noch nicht ausreichend untersuchten Problembereiche beziehen sich auf den Aufbau (komplexer) Interaktionsstrukturen und deren Optimierung. Das verwendete Beispielproblem ist dafür jedoch nicht geeignet. Weiterhin wurde die Reproduktionsschwelle als gegeben

vorausgesetzt und vom Systemdesigner definiert. Hierbei stellt sich die Frage nach einer sinnvollen Ableitung dieser Größe oder einem selbstadaptiven Verhalten. Ebenfalls problematisch ist die Vorgabe als Durchschnitt über der gesamten Population. Dieses Wissen ist in den relevanten Informationssystemen unter Umständen nicht global kommunizierbar. Damit ist die (zentrale) Festlegung der Reproduktionsschwelle für alle Agenten ein weiterer Untersuchungsgegenstand. Zusätzlich ist die Agentenpopulation homogen in dem Sinne, dass alle Agenten gleiche Aufgaben ausführen. Dies kann in komplexen Systemen nicht als gegeben vorausgesetzt werden und macht eine Anwendung schwierig.

Forschungsbedarf

Damit ergibt sich an der Schnittstelle evolutionärer Verfahren und Agenten folgender Überblick im Kontext dieser Arbeit. Smith et al. stellt ein agentenbasiertes evolutionäres Framework [221] vor belässt die Fitnessfunktion als direkt problemabhängige Größe. Weiterhin erfolgt die Optimierung nicht als Anpassungsleistung aufgrund äußerer Veränderungen. Damit ist ein Einsatz in dynamischen und heterogenen Umgebungen nicht möglich. Holland [113], Menczer [156] und Eymann [64] verwenden jeweils von der Umgebung bereitgestellte Ressourcen (Energie, Geld) als Basis für Anpassungen. Hierbei erfolgt in den Ansätzen von Holland und Eymann jeweils eine Untersuchung adaptiver und koordinierender Effekte des (Ressourcen)Flusses in komplexen Systemen. Menczer verwendet Energie für eine Anpassung der Populationsgröße, um Suchgeschwindigkeit in Graphen zu optimieren. Interaktionen und Fluss von Energie zwischen den Agenten finden nicht statt. Hier kann das entstehende (Such)system als nicht komplex erachtet werden.

Der Forschungsbedarf ergibt sich als Kombination der vorgestellten Arbeiten. Dies beinhaltet zum Einen die Nutzung eines evolutionären Frameworks mit Dezentralisierung aller notwendigen Operatoren zur Optimierung von Problemen in komplexen Systemen. Zum Anderen verspricht der Einsatz des Fluss-Konzeptes auch den Einsatz zur Optimierung. Weiterhin bestehen im Hinblick auf die lokale Selektion und die Ableitung geeigneter Reproduktionsschwellen und -zeitpunkte noch offene Fragen: Wie und von wem wird die Reproduktionsschwelle festgelegt? Wie ist die Fitness fremder Agenten richtig einzuschätzen? Hierbei kann die Verknüpfung biologischer und ökonomischer Modelle als vielversprechender Ansatz angesehen werden. Weiterhin besteht Bedarf nach Evaluation und Suche geeigneter Probleminstanzen, um entsprechende Komplexität zu untersuchen. Im Hinblick auf die sogenannte endogene Fitness führt Mitchell [163] auf Seite 10 an, dass es momentan kein mathematisches Modell zum Schema Theorem mit endogener Fitness gibt.

2.5 Zusammenfassung

Die vorangegangenen Abschnitte haben offene Punkte zur Optimierung in verteilten Systemen im Hinblick auf die Zielsetzung dieser Arbeit identifiziert. Zunächst wurde die Ähnlichkeit natürlicher komplexer Systeme mit technischen Systemen gezeigt. Die hierbei in biologischen Systemen vorkommenden Eigenschaften wie Adaptivität, Robustheit und Fehlertoleranz und deren Übertragung auf technische Systeme sind Gegenstand der Forschung auf vielen Gebieten, wie Informatik, Physik, Biologie, Soziologie, Systemwissenschaft und weiterer.

Abbildung 2.14 zeigt die Vorgehensweise zum Finden geeigneter Methoden für eine Umsetzung der Anforderungen. Hierzu wurden nach Identifikation und Evaluation relevanter Informationssysteme (IS) Methoden und Technologien unterschiedlicher Gebiete untersucht.

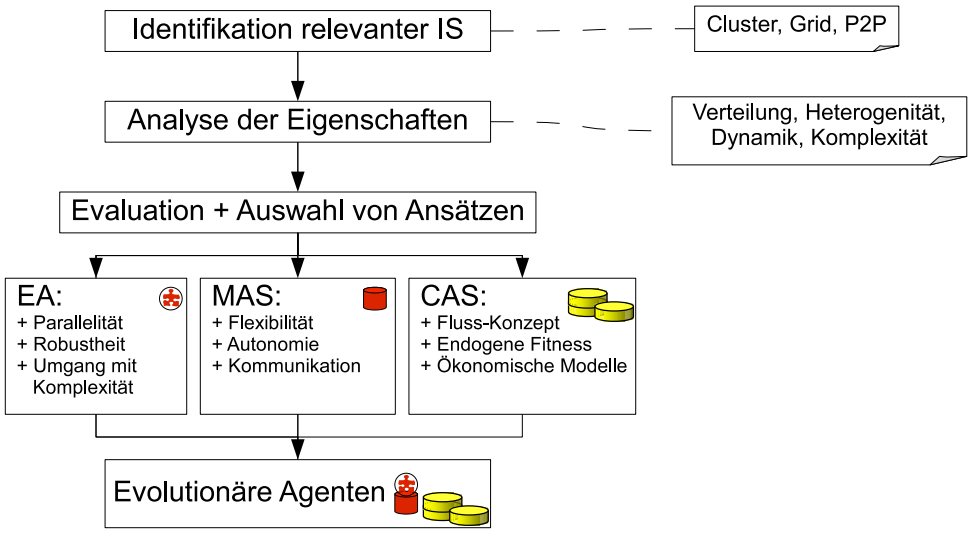


Abbildung 2.14: Ableitung evolutionärer Agenten aus vorhandenen Forschungsansätzen

Auf dem Gebiet der verteilten Optimierung wurden evolutionäre Verfahren durch ihre implizite Parallelität ausgewählt. Ebenso ist Robustheit im der Umgang mit hoher Komplexität von Suchräumen, Problemen und Lösungen gegeben. Multiagentensysteme zeichnen sich durch eine hohe Flexibilität aus, die in dynamischen und unbekannten Umgebungen die geforderte Adaptivität bereitstellen. Kommunikation als zukünftig wichtigstes Merkmal verteilter Software wird durch die Agententechnologie auf unterschiedlichen Ebenen bereitgestellt. Dazu gehören unter anderem nachrichtenbasierte Kommunikation, Anwendung von Kommunikationsprotokollen und auch semantisch angereicherte Interaktionen, die im heterogenen Umfeld notwendig sind. Eine Verbindung der Vorteile (siehe Abbildung 2.14) beider Paradigmen ist ein vielversprechender Ansatz zur Erreichung der Zielvorgaben.

Die gewählten evolutionäre Verfahren und die Agententechnologie weist in den untersuchten Forschungsarbeiten noch einige Bereiche auf, die näher untersucht werden müssen. Im Kontext evolutionärer Verfahren hat die Untersuchung und Anwendung in heterogenen Strukturen, wie Grid und Peer-to-Peer Systeme, erst vor wenigen Jahren angefangen. Trotz Vorhandenseins evolutionärer Frameworks mit dezentralen Operatoren sind für Anwendungen in verteilten Systemen noch Fragen offen. Hierzu zählt die Ausgestaltung endogener Fitness, lokaler Selektion und das ebenfalls lokale Initiieren von Reproduktion zur richtigen Zeit. Ebenfalls vielversprechend ist die Verbindung evolutionärer und ökonomischer Modelle, die weiterer Untersuchung bedürfen.

Im Hinblick auf Agentensysteme merkt Weiss [247] an, dass agentenorientierte Kollaborationsmuster nur für eine kleine Anzahl von Agenten geeignet sind. Der Einsatz in massiv parallelen Umgebungen mit Millionen von Teilnehmern stellt zum Beispiel für Contract Net Protokolle ein Ausschlusskriterium dar. Ebenso geht das verteilte Problemlösen in der Agentendomäne grundsätzlich von kooperationsbereiten Agenten aus. Der Kontext dynamischer und virtueller Organisationen lässt diesen Schluss jedoch nicht zu. Hier konkurrieren die Teilnehmer primär um Ressourcen und erst in zweiter Linie folgt eine zeitlich befristete Kooperation. Auch die Annahme gleicher Ziele ist im Kontext virtueller Organisationen als nicht gegeben zu betrachten. Trotz Flexibilität und Kommunikation auf semantisch hoher Ebene sind Verhaltensweisen per Design festgelegt und lediglich in den vorgesehenen Grenzen

adaptiv.

Zusammenfassend ergeben sich folgende Problemstellungen als Teilaspekte der Forschungsfrage (Adaptivitäts- und Optimierungsansatz in verteilte Systemen, Kapitel 1.2.4):

- Wie kann die Systemgröße selbstadaptiv gesteuert werden (Anzahl Agenten)?
- Wie kann das Gesamtsystemverhalten als Resultat aus lokalen Verhaltensmustern selbstadaptiv gesteuert werden?
- Wie können Systemgröße und Systemverhalten selbstoptimierend gestaltet werden?
- Wie kann eine adäquate lokale Leistungsbewertung der Agenten erreicht werden?
- Wie lässt sich aus lokaler Leistungsbewertung eine lokale Selektionsmethode ableiten?
- Wie verhält sich die Takeover time für dynamische, schwach vernetzte und heterogene Netzwerkstrukturen?
- Lässt sich ein verteiltes Optimierungsverfahren auch in dynamische, schwach vernetzte und heterogene Netzwerkstrukturen, z.B. Grid, Cluster, Peer-to-Peer Systeme, realisieren?
- Wie lässt sich Nachfrageorientierung erreichen?

Die Intention dieser Arbeit ist es, eine Kombination aus evolutionärem und agentenbasiertem Verhalten zu realisieren, um die gewünschte Selbstadaptivität und Selbstoptimierung zu ermöglichen. Hierbei wurden die bestehenden Ansätze analysiert und hinsichtlich ihrer Vorteile ausgewertet. Dies sind im Einzelnen:

- Integration biologischer und ökonomischer Perspektiven und Bereitstellung von Geld durch die Umgebung (Nachfrage)
- Finanzielle Transaktionen zwischen Agenten bzw. Agent und Umgebung und damit Geldfluss im System
- Endogene Fitness ermöglicht die problemunabhängige Bewertung von Aktionen auf Agentenebene

Kapitel 3

Evolutionäre Agenten

Nach Einführung von Agenten und evolutionären Verfahren erfolgt in diesem Kapitel die Einführung evolutionärer Agenten. Abschnitt 3.1 enthält das formalisierte Modell. Darauf aufbauend wird in Abschnitt 3.2 die evolutionäre Optimierung im Agentenkontext entwickelt und abschließend in Abschnitt 3.3 analysiert.

3.1 Formalisierung

Dieses Kapitel beschreibt die notwendige formale Basis für den weiteren Verlauf dieser Arbeit. Die in Kapitel 1.2 aufgeworfenen und in Kapitel 2 diskutierten Probleme der Verteilung, Heterogenität, Dynamik und Komplexität erfordern eine abstrakte Architektur für evolutionäre Agenten, die eine angemessene Balance zwischen Adaptivität, Robustheit gegenüber Änderungen bei gleichzeitiger Effizienz ermöglichen.

3.1.1 Umgebung

Für das Konzept der Umgebung ist es notwendig, für die geforderte Spezifikation eine minimale, generelle und möglichst flexible Beschreibung im Sinne der Anforderungen zu liefern.

3.1.1.1 Struktur der Umgebung

Global existiert gemäß der Definition Ferbers (vgl. [66]) eine Agentenumgebung, die Basisdienste wie Nachrichtenversand, Laufzeitumgebung etc. bereitstellt. Entsprechend der verteilten Umgebung ist eine Partitionierung der globalen Agentenumgebung notwendig, um beliebige physikalische bzw. virtuelle Netzwerkstrukturen abzubilden. Die Partitionen bzw. virtuellen Agentenumgebungen bieten formal die gleichen Basisdienste an, schränken jedoch die Sichtbarkeit (vgl. VSK Logik [257]) und damit den Aktionsbereich einzelner Agenten oder Agentengruppen individuell ein. Somit lassen sich individuelle Umgebungen für Agenten oder Gruppen von Agenten definieren. Das generelle Konzept virtueller Agentenumgebungen ist nicht beschränkt auf die Abbildung physikalischer Strukturen, sondern erlaubt ebenso die Modellierung sozialer Strukturen von Organisationen, wie bspw. von Kirn (vgl. [129]) beschrieben.

Hierzu sei $A = \{a_1, a_2, \ldots, a_n\}$ die Menge der Agenten (siehe hierzu Abschnitt 3.1.2) und weiterhin sind alle möglichen Teilmengen von A definiert durch die Potenzmenge $\mathcal{P}(A)$. Die Struktur der Agentenumgebung ist folgendermaßen definiert:

Definition 3 (Struktur der Agentenumgebung)**.**
Die Struktur bzw. der Graph der Agentenumgebung $G_U(\mathcal{P}(A))(V, E)$ oder kurz $G_U(V, E)$ ist ein gerichteter Graph mit Knotenmenge $V \subseteq \mathcal{P}(A)$ und Kantenmenge E. Jeder Knoten $v \in V \mid v \in \mathcal{P}(A)$ stellt eine virtuelle Agentenumgebung dar und jede Kante $e = (v_i, v_j) \in E$ ist eine gerichtete Verbindung zwischen zwei virtuellen Agentenumgebungen.

Jeder Agent $a \in A$ ist in mindestens einer virtuellen Umgebung enthalten:

$$\forall a \; \exists v \in V \tag{3.1}$$

daraus folgt, dass für alle Agenten eine Umgebung im Sinne von Definition 3 existiert und damit Agenten ausserhalb von Agentenumgebungen nicht zulässig sind:

$$\bigcup_{v \in V} (a \in v) = A \tag{3.2}$$

Zur Veranschaulichung soll das Konzept der strukturierten Agentenumgebung am Beispiel eines dEA erläutert werden. Dazu zeigt Abbildung 3.1 vier Inseln des dEA entsprechend als vier virtuelle Agentenumgebungen mit $v_1 = \{a_1, a_2, a_3, a_4\}, v_2 = \{a_5, a_6, a_7, a_8\}, v_3 = \{a_9, a_{10}, a_{11}, a_{12}\}, v_4 = \{a_{13}, a_{14}, a_{15}, a_{16}\}$ und $V = \{v_1, v_2, v_3, v_4\}$. Die gerichteten Kanten sind $e_1 = (v_1, v_2), e_2 = (v_2, v_1), e_3 = (v_2, v_4), e_4 = (v_4, v_2), e_5 = (v_3, v_4), e_6 = (v_4, v_3), e_7 = (v_1, v_3), e_8 = (v_3, v_1)$ und $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$.

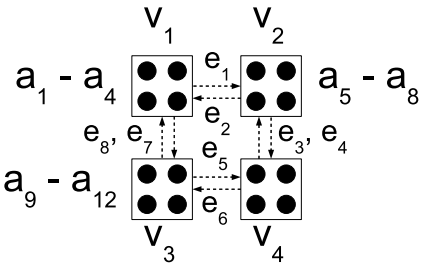


Abbildung 3.1: Beispielstrukturgraph einer Agentenumgebung V bestehend aus vier virtuellen Umgebungen v_1, v_2, v_3, v_4 und deren Kanten

Es existiert formal genau eine Umgebung, die durch Untermengen partitioniert werden kann. So, wie bei parallelen EA die Knoten bzw. Demes oder Subpopulationen. Darüber hinaus lässt sich eine Partitionierung auch im Sinne der Organisationstheorie (vgl. [129], bzw. Abschnitt 2.2.4) nutzen, um beispielsweise Organsiationen, Gruppen, Teams etc. zu modellieren.

3.1.1.2 Objekte

Der vorliegende Ansatz fokussiert als Softwareansatz auf nachrichtenbasierter Interaktion zwischen den Agenten. Daher beeinflussen oder manipulieren Agenten einander nicht direkt, sondern versenden Nachrichten als Intention für Aktionen anderer Agenten. Objekte sind Dinge in einer Agentenumgebung, deren Zustand wahrgenommen bzw. gegebenenfalls manipulierbar ist. Wie in der Robotik können Objekte Informationsträger sein und als Teil der Agentenumgebung für Agenten wahrnehmbare Informationen bereitstellen. Gleichwohl dienen Objekte zur Repräsentation unterschiedlichster Kontexte.

Objekte sind auch Elemente, welche ein Agent besitzen kann. Dazu zählt Geld genauso wie Informationen oder Wissen, z.B. in Form eines Vertrages zwischen Agenten. Eine Manipulation von Objekten kann ein Besitzerwechsel im Zuge einer Transaktion Geld(objekt) gegen Informations(objekt). Objekte können aber auch Services sein, die Informationen vorhalten bzw. deren Präsenz durch Agenten wahrgenommen wird.

Der Einfachheit halber wird auf explizite formale Spezifikation sogenannter Adress- bzw. Diensteverzeichnisse (engl. *Yellow Pages*) und die tatsächlichen Dienste verzichtet. Diese Funktionalität kann durch ein spezielles Objekt wahrgenommen werden. Ein Datenbank-Objekt kann Informationen über verschiedene Dienste bereitstellen. Ebenso kann dies auch durch spezielle Agenten des Agentensystems realisiert werden (vgl. Yellow Pages im JADE Framework [20].

Definition 4 (Objekt)**.**
Ein Objekt $O = (o_1, \ldots, o_n)$ *ist ein endliches n-dimensionales Tupel.*

3.1.1.3 Sichtbarkeit

Die Partitionierung der Agenten gibt einen ersten Hinweis auf die Sichtbarkeit der Umgebung aus der Perspektive von einzelnen Agenten. Entsprechend dem $\mathcal{VSK}$ Ansatz [257] bestimmt die Sichtbarkeitsfunktion $vis(a)$ (*visibility function*), was in der jeweiligen Umgebung für einen Agenten $a \in A$ sichtbar und damit wahrnehmbar ist.

Hierdurch wird es zum Beispiel ermöglicht, die Umgebungspartitionierung $1:1$ auf eine Agentenpopulation zu übertragen. Ebenso ist es möglich, unabhängig von den physikalischen Gegebenheiten die Sichtbarkeit entsprechend organisatorischer Strukturen zu definieren. Zur Veranschaulichung sei der Begriff Sichtbarkeit am Beispiel einer Gitterstruktur erläutert. In einer torusförmigen Gitterstruktur besitzt jeder Knoten genau vier Nachbarknoten. Die Sichtbarkeit ermöglicht die Einschränkung der sichtbaren Nachbarknoten auf genau die vier Nachbarknoten. Ein weiteres Beispiel ist die Verwendung von Diensteverzeichnissen (Yellow Pages). Hier können Dienste individuell im Sinne bestimmter Zugriffsrechte für einzelne Agenten sichtbar gemacht werden.

Definition 5 (Sichtbarkeit)**.**
Die Abbildung $vis : A \mapsto \{G'_U, O', A'\}$ *heißt Sichtbarkeitsfunktion mit folgenden Eigenschaften:*

- $G'_U \subseteq G_U$ *ist ein Teilgraph der Agentenumgebungsstruktur*
- $O' \subseteq O$ *ist eine Teilmenge der vorhandenen Objekte*
- $A' \subseteq A$ *ist eine Teilmenge der Agenten*
- $a \in A$ *ist ein Agent, für den* G'_U, O' *und* A' *sichtbar sind*
- *Jeder Agent kann sich selbst wahrnehmen:* $\forall a \in A$ *gilt* $a \in vis(a)$
- *Ausgehend von Gleichung 3.1 wird weiterhin festgelegt, dass jeder Agent mindestens die Umgebungen wahrnimmt, in denen er sich selbst befindet:* $\forall a \in A \; \exists V \mid V := v \mid a \in v$

Damit definiert die Sichtbarkeitsfunktion eine Relation zwischen der Menge der Agenten und den Mengen der jeweils wahrnehmbaren Umgebungsstruktur, Agenten und der vorhandenen Objekte.

3.1.1.4 Definition Agentenumgebung

Aus den vorherigen Definitionen von Umgebungsstruktur, Sichtbarkeit und Objekten ergibt sich die Agentenumgebung. Im Unterschied zur Definition Ferbers (siehe Abschnitt 2.2.4) sind Objekte keine Agenten, sondern passive Elemente.

Definition 6 (Agentenumgebung)**.**
Eine Agentenumgebung ist ein Tripel

$$U = (G_U, vis, O)$$

wobei gilt:

- G_U *ist die Struktur der virtuellen Agentenumgebungen*
- vis *ist die Sichtbarkeitsfunktion, die festlegt, was für einen Agenten wahrnehmbar ist.*
- O *ist die Menge der Objekte*

Der Zustand der Agentenumgebung zu einem bestimmten Zeitpunkt t ergibt sich damit als Momentaufnahme $U(t)$. Der Initialzustand $U(t_0)$ ist derjenige Zustand, den die Agentenumgebung direkt nach Systemstart bzw. nach der Initialisierung einnimmt. Abbildung 3.2 zeigt zwei virtuelle Umgebungen v_1 und v_2. Die Sichtbarkeit von Agent a_1 ist durch die schraffierte Flächte gekennzeichnet. a_1 sieht die anderen Agenten in $v_1 : a_2, a_3, a_4$ und das Objekt o_1. Die gesamte virtuelle Umgebung v_2 ist für ihn unsichtbar.

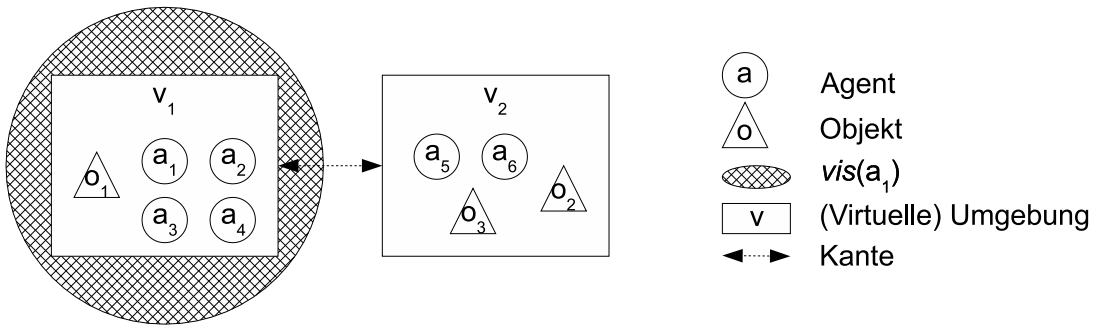


Abbildung 3.2: Beispielhafte Agentenumgebung mit Kennzeichnung der Sichtbarkeit von Agent a_1

Die Eigenschaften von U sind anhand den von Russell und Norvig [205] festgelegten Eigenschaftsdimension wie folgt zu beurteilen (vgl. Abschnitt 2.2.2):

- **Nicht zugreifbar**: Eine vollständige Wahrnehmung der Umgebung wird nicht garantiert. Die Sichtbarkeit eines Agenten ist eingeschränkt durch $vis(a)$.
- **Nicht deterministisch**: Aus Sicht eines Agenten kann sich die Umgebung ohne dessen Zutun verändern. Weiterhin muss eine Aktion nicht zu einem garantierten Ergebnis führen (etwa wenn der Empfänger einer Nachricht nicht mehr existiert und deshalb keine Antwortnachricht verschickt).
- **Dynamisch**: Die Umgebung kann sich unabhängig von den Aktionen eines einzelnen Agenten verändern (z.B. durch Andere Agenten).

- **Kontinuierlich**: Formal ist obige Definition als kontinuierlich einzustufen, da die wahrnehmbare Umgebung nicht notwendigerweise a priori festgelegt und endlich sein muss.

3.1.2 Agent

Dieser Abschnitt beschreibt ein formales Modell der verwendeten Agenten. Dieses basiert auf der in Kapitel 2.2.1 eingeführten allgemeinen Definition (siehe dazu auch [247]) und dient selbst als Grundlage formaler Erweiterungen in nachfolgenden Abschnitten.

3.1.2.1 Innerer Zustand

Der innere Zustand $z \in Z = \{z_0, z_1, \ldots, z_n\}$ eines Agenten beinhaltet das Wissen des Agenten. z kann eine ggf. beliebig komplexe Struktur sein und dient dem Agenten als Speicher beliebiger Informationen. Im Gegensatz zu einem rein reaktiven Agent ohne inneren Zustand basiert damit der Informationsverarbeitungsprozess zumindest teilweise auf seinem inneren Zustand und damit nicht nur auf dem aktuellen Zustand der Umgebung. Der initiale innere Zustand eines Agenten $z_0 \in Z$ ist derjenige Zustand direkt nach Start bzw. Initialisierung des Agenten.

3.1.2.2 Wahrnehmung

Ein Agent nimmt maximal den per Sichtbarkeitsfunktion vis (Definition 5) definierten und für ihn selbst einsehbaren Ausschnitt der Umgebung wahr. Diese Wahrnehmung wird detailliert beschränkt auf die für den Agenten erlaubten Zustände bei Objekten bzw. Aktionen anderer Agenten (siehe Abschnitt 3.1.2.3). Dies bedeutet, dass Agent a_1 mit $vis(a_1) = \{a_2\}$ nicht alle Aktionen von Agent a_2 sieht, sondern nur die für ihn bestimmten Nachrichten. Andernfalls wäre a_1 in der Lage, alle Nachrichten eines für ihn sichtbaren Agenten 'mitzuhören'. Ebenso verhält es sich mit Objekten im Sichtbarkeitsbereich eines Agenten. Hier kann ebenfalls nur auf den erlaubten Teilzustand zugegriffen werden bzw. dieser manipuliert werden. Als Beispiel für die Wahrnehmung sei der Empfang einer Nachricht genannt. Ein anderes Beispiel ist die Wahrnehmung des Lagerbestandes mittels eines RFID Sensors. Hiermit empfängt der Agent alle ein- und ausgehenden Güterobjekte.

Eine weitere Einschränkung der von der bereitgestellten Informationen durch vis kann erfolgen, wenn der Agent selbst nicht in der Lage ist, die ihm zur Verfügung stehenden Informationen (richtig) wahrzunehmen. Ein Fehler in der Sensorik führt zu eingeschränkter Wahrnehmung. Beispielsweise liefert ein defekter RFID Sensor keine oder falsche Daten. Das Fehlen entsprechender Sensoren führt ebenso zu eingeschränkter Wahrnehmung seitens des Agenten. Hierbei führen die per $vis(a)$ unterscheidbaren Zustände der Umgebung beim Agenten zu nicht unterscheidbaren Zuständen.

Zur Veranschaulichung wird obiges RFID-Beispiel ergänzt um einen weiteren Umgebungszustand 'Temperatur über $0°C$'. Zur einfachen Repräsentation sei 'RFID Chip erkannt' mit x bezeichnet und 'Temperatur über $0°C$' mit y. Beide Informationen sind per $vis(a)$ sichtbar, jedoch nimmt Agent a lediglich x wahr. Damit kann $vis(a)$ insgesamt vier Zustände annehmen:

$$vis(a) = \{\{x, y\}, \{x, \neg y\}, \{\neg x, y\}, \{\neg x, \neg y\}\}$$

Für a lassen sich daher insgesamt nur die zwei Zustände der Umgebung $\{\{x, *\}, \{\neg x, *\}\}$ unterscheiden, da y nicht erkannt wird. Mit $'*'$ ist das Wildcard Symbol bezeichnet für ein

beliebiger Zustand von y. Damit kann a aufgrund eingeschränkter Wahrnehmung lediglich wahrnehmen, ob sich ein RFID Chip in Sensorreichweite befindet oder nicht. Damit ergibt sich für die Sensoren zur Wahrnehmung eines Agenten folgende Definition

Definition 7 (Sensoren)**.**
Die Wahrnehmung eines Agenten a ist definiert durch die Teilmenge

$$Sen_a \subseteq vis(a)$$

Hiermit stehen einem Agenten a alle für ihn wahrnehmbaren Informationen in Sen_a zur Verfügung. Die Anzahl der gleichzeitig wahrnehmbaren Informationen wird mit $|Sen_a|$ bezeichnet. Dieses Konzept erlaubt die gleichzeitige Wahrnehmung endlich vieler Informationen als Input für die Verarbeitung.

3.1.2.3 Aktionen

Die Fähigkeiten eines Agenten werden abstrakt als Aktionen bezeichnet. Aus Sicht eines Agenten wird mittels Aktionen die Umgebung verändert. Aktionen zwischen den Agenten sind üblicherweise Nachrichten, da sich Agenten nicht direkt manipulieren können. Als Beispiel sei eine Anfrage eines Agenten an einen anderen Agenten genannt. Eine solche Anfragenachricht führt nicht zwingend zu einer intendierten Aktion des empfangenden Agenten. Voraussetzung für die Ausführung einer Aktion auf einen Agenten ist die Sichtbarkeit zwischen den kommunizierenden Agenten.

Im Gegensatz zu Agenten können Aktionen auf Objekte direkt angewendet werden, sofern der jeweilige Agent das Objekt wahrnehmen bzw. manipulieren darf. Beispielsweise das Hinzufügen von Daten in eine Datenbank. Damit ergibt sich folgende Definition:

Definition 8 (Aktionen)**.**
Die Menge der möglichen Aktionen eines Agenten wird repräsentiert duch

$$Akt = (akt_1, \ldots, akt_m)$$

3.1.2.4 Transferfunktion und Strategie

Mit Hilfe einer geeigneten Transferfunktion setzt der Agent die wahrgenommene Umgebung und seinen inneren Zustand in Aktionen und einen neuen inneren Zustand um. Die Transferfunktion übernimmt die Rolle einer 'Abbildungsmaschine', die dauerhaft die Umgebungsreize und den inneren Zustand in Aktionen und einen neuen inneren Zustand überführt. Wie bereits in Abschnitt 3.1.2.1 ausgeführt, erlaubt erst die Verwendung eines inneren Zustandes die Realisierung proaktiver Agenten.

Der Agent startet in einem initialen Zustand z_0. Das Verhalten zeigt Abbildung 3.3. Durch Sen wird der für den Agenten sichtbare und wahrnehmbare Ausschnitt der Umgebung wahrgenommen und zusammen mit dem inneren Zustand z als Definitionsmenge der Transferfunktion φ verwendet. φ bildet beide Mengen ab in einen neuen inneren $z' \in Z$ und Aktionen aus A. Durch den inneren Zustand kann leicht proaktives Verhalten erzeugt werden, etwa als ein Zähler, welcher zurückgezählt bei 0 eine Aktion auslöst. Weiterhin wird angenommen, dass φ kontinuierlich ausgeführt wird etwa im Sinne einer Endlosschleife in zeitlich kontinuierlichen Umgebungen oder für diskrete Zeitintervalle in jedem Schritt.

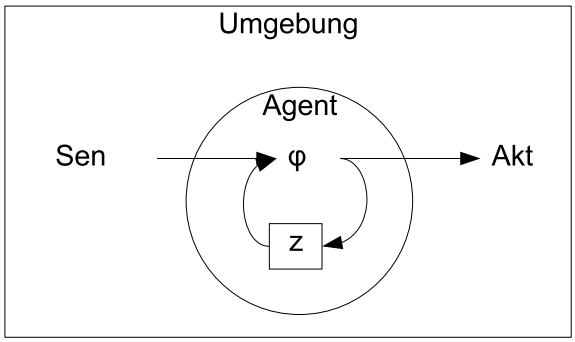


Abbildung 3.3: Agent mit innerem Zustand z, Wahrnehmung Sen und Aktionen Akt und der Transformationsfunktion φ

Definition 9 (Transferfunktion)**.**
Die Transferfunktion ist eine Abbildung

$$\varphi := (Sen, z) \mapsto (Akt, z') \quad z, z' \in Z$$

des 2-Tupels Sensoren, innerer Zustand (Sen, z) *in ein zweielementiges Element Aktionen, Zustand der Form* (Akt, z')*.*
Die Mengendefinition lautet damit:

$$\varphi : Sen \times Z \rightarrow Akt \times Z$$

Für agentenspezifische Adaption von φ benötigt jeder Agent zusätzlich eine Strategie s:

Definition 10 (Strategie)**.**
Eine Strategie $s = (s_1, s_2, \ldots, s_m)$ *bezeichnet den variablen Teil der Transferfunktion* φ *eines Agenten gegenüber anderen Agenten.*

Darin ist jener Teil des Verhaltens, kodiert in φ, enthalten, den jeder Agent individuell verwendet. Dies kann über die Kodierung einzelner Zahlen, Terme bis hin zu beliebig komplexen Funktion reichen. Die Strategie lässt sich ohne Beschränkung der Allgemeinheit als Vektor $s = (s_1, s_2, \ldots, s_m)$ realisieren und ermöglicht somit eine vereinfachte Analyse des Agenten auf Basis der unterschiedlichen Strategievektoren.

Als Beispiel soll der oben verwendete Agent mit Sensor zur Bestandsermittlung *Bestand* eine Funktion für die Überwachung und automatische Auslösung von Bestellungen für Lagerbestände eines Produkts ausführen. Die Bestellauslösung für einen solchen reaktiven Agenten kann vereinfacht durch folgende Funktion definiert werden:

$$\varphi(Bestand) = \begin{cases} 0, & \text{wenn } Bestand > Mindestbestand \\ Lagergroesse - Bestand, & \text{wenn } Bestand \leq Mindestbestand \end{cases}$$

Bestand bezeichnet den aktuellen Bestand des Produkts im Lager, der maximal mögliche Güterbestand ist vorgegeben mit *Lagergroesse*. Der $Mindestbestand$ bezeichnet die Schwelle, ab der wieder nachbestellt werden muss. Der Erfolg der Strategie eines Agenten in diesem

Beispiel lässt sich an der Wahl des *Mindestbestand*es festmachen. Ist dieser zu hoch, entstehen überhöhte Lagerkosten, während zu niedrige Lagerbestände Lieferengpässe und damit eventuelle Regresszahlungen nach sich ziehen.

Der Einfachheit halber wird folgende Notation vereinbart um sprechenden Zugriff auf einzelne Strategieparameter zu erhalten: $s(s_1) \mapsto s_1$. Hierdurch lassen sich Agenten und deren Verhalten vereinfacht anhand ihrer Strategie beschreiben. Die Analyse des Verhaltens kann damit auf die Analyse der unterschiedlichen Strategien zurückgeführt werden.

3.1.2.5 Definition Agent

Die formale Definition eines Agenten ergibt sich damit wie folgt:

Definition 11 (Agent)**.**
Ein Agent a ist ein Tupel der Form

$$a = (Akt, Sen, Z, \varphi, s)$$

dabei ist:

- *Akt ist die Menge der möglichen Aktionen (siehe Definition 8),*
- *Sen ist die Menge der für den Agenten erfassbaren Wahrnehmungen, (siehe Definition 7)*
- *Z die Menge der inneren Zustände mit einem initialen Zustand z_0 (siehe Abschnitt 3.1.2.1),*
- *φ die Transformationsfunktion, die die Wahrnehmungen und den inneren Zustand in Aktionen umsetzt und den inneren Zustand anpasst (siehe Definition 9),*
- *s die Strategie des Agenten (siehe Definition 10).*

Der in diesem Kapitel vorgestellte Ansatz orientiert sich an der in Abschnitt 2.2.1 vorgestellten '*Weak Notion of Agency*' von Wooldridge und Jennings [255]. Diese wird hier kurz vor den Hintergrund obiger Definitionen reflektiert:

- **Autonomie**: Agenten agieren nach dem Start bzw. der Initialisierung autonom und führen φ aus. Hiermit haben sie mindestens partielle Kontrolle über ihre Aktionen und ihren Zustand.
- **Soziale Fähigkeiten**: Agenten interagieren miteinander durch den Versand von Nachrichten
- **Reaktivität**: Agenten nehmen ihre Umgebung mittels *Sen* wahr und reagieren zeitnah auf Veränderungen
- **Proaktivität (pro-activeness)**: Agenten reagieren nicht nur auf Veränderungen der Umgebung, sondern verfolgen aktiv eigene Strategie

3.1.3 Multiagentensystem

Anhand der vorherigen Abschnitte ergibt sich zusammenfassend die Definition eines Multiagentensystems MAS:

Definition 12 (Multiagentensystem (MAS))**.**
Ein Multiagentensystem $MAS = (U, A)$ ist ein 2-Tupel, bestehend aus

- *Einer Umgebung U gemäß Definition 6*
- *Einer Menge von Agenten $A = \{a_1, a_2, \ldots, a_n\}$ gemäß Definition 11*

Das Verhalten während des Laufes eines Multiagentensystems hängt ab von der Umgebung U (Struktur, Objekte), den Strategien s und Transformationsfunktionen der Agenten φ und deren jeweiligem initialen Zustand z_0. Unter gegebenen Umständen bestehen verschiedene Reaktionsmöglichkeiten auf der MAS Systemebene, Veränderungen herbeizuführen. Eine Möglichkeit ist, die Strategie (siehe Definition 10) der Agenten zu verändern. Dies ist für die Transformationsfunktion der Agenten der variable Teil. Eine weitere Möglichkeit besteht im Entfernen oder Hinzufügen von Agenten zum System. Auch dies bedeutet eine Strategieänderung auf globaler Ebene (MAS), ausgelöst durch lokale Veränderungen (Agentenebene).

3.1.3.1 Strategie

Gemäß der Strategie des Agenten (Definition 10) lässt sich die globale Strategie des Multiagentensystems wie folgt definieren:

Definition 13 (Strategie des Multiagentensystem)**.**
Sei $MAS = (U, A)$ ein Multiagentensystem und $A = \{a_1, \ldots, a_n\}$ eine Menge von Agenten. Dann heißt

$$S_A = (s_{a_1}, s_{a_2}, \ldots, s_{a_n})$$

Strategie des Multiagentensystems, bestehend aus den Strategien der Agenten $a \in A$.

Während der Laufzeit kann sich die Strategie des Multiagentensystems S_A verändern und ist zeitlich variabel. Ebenso ist die Anzahl der Agentenstrategien $|S_A|$ variabel, wenn Agenten aus dem System entfernt oder hinzugefügt werden.

3.1.3.2 Funktion

Die Funktion des Multiagentensystems ist eine Komposition der Transferfunktionen der Agenten. Wenn eine Funktion in einem Netzwerk aus 'einfacheren' Transferfunktionen φ berechnet wird, fließt Information durch die gerichteten Kanten eines Funktionsgraphs, der von den Agenten zur Laufzeit aufgespannt wird (siehe Abbildung 3.4).

Einige Agenten kommunizieren, indem ihre Aktionen von den Sensoren anderer Agenten wahrgenommen werden. Diese Ergebnisse fließen in die Transferfunktion des nachfolgenden Agenten ein. Einen Schnappschuss einer Komposition von Transferfunktionen zeigt Abbildung 3.4. Der interne Zustand wurde zugunsten der Übersichtlichkeit weggelassen. Hierbei löst die Wahrnehmung sen_{a_1} von Agent a_1 eine Interaktion mit a_2 und a_3 und zwischen a_2 und a_3 aus, in deren Folge Aktion $\varphi_{a_3}(\varphi_{a_1}, \varphi_{a_2}(\varphi_{a_1}(sen_{a_1}))) \mapsto akt_{a_3}$ entsteht.

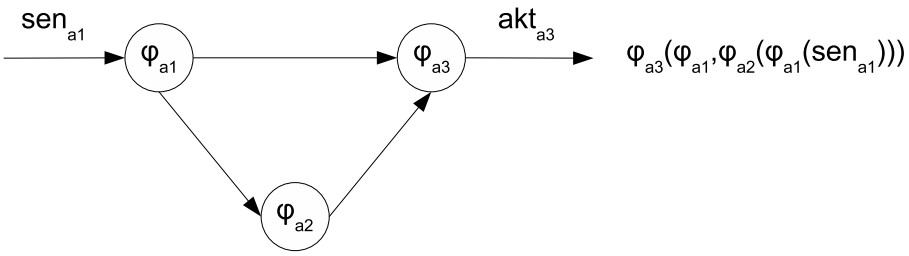


Abbildung 3.4: Funktionskomposition in einem Multiagentensystem

Wenn das Interaktionsnetzwerk Zyklen enthält, hängt die Kommunikation nicht nur von der aktuellen Wahrnemung, sondern auch von den vorangegangenen Wahrnehmungen ab. Dieses Feedback verhält sich ebenso rekursiv, wie die Transferfunktion eines Agenten und seines inneren Zustandes.

Definition 14 (Funktion des Multiagentensystem)**.**
Die Funktion des Multiagentensystems

$$\Phi(t) = (\Phi(t-1), \varphi_{a_1}(t-1), \ldots, \varphi_{a_n}(t-1))$$

ist eine rekursive Funktionskomposition der Transferfunktionen der Agenten $a \in A$.

Ebenso wie die Strategie S_A ist die Funktion variabel in der Anzahl ihrer Parameter in Abhängigkeit von der Anzahl der im System befindlichen Agenten.

3.1.4 Kosten

In Systemen, egal welcher Art, ist der Betrieb und die Aufrechterhaltung der intendierten Funktion mit Kosten verbunden. Kosten resultieren aus unterschiedlichsten Voraussetzungen und Betrachtungsweisen von Systemen. Das Ziel dieses Abschnittes besteht darin, ein generelles und allgemeines Verständnis von Kosten zu etablieren und eine allgemeingültige Definition herauszuarbeiten.

Eigenschaftsgebundene fixe Kosten

Eigenschaftsgebundene Kosten sind vergleichbar mit den *Fixkosten*[214] im betriebswirtschaftlichen Sprachgebrauch. Ziegenbein etwa betrachtet funktionale und nichtfunktionale Eigenschaften [261] eingebetteter Systeme, wie z.B. Stromverbrauch und Speichergröße. Hieraus lassen sich bei geeigneter Modellierung direkt Kosten ableiten durch Aufsummierung der Kosten aller Einzelbestandteile [105]. Ebenso lassen sich für reine Softwarelösungen fixe Kosten ableiten, wie z.B. Speicherbedarf im Arbeitsspeicher bzw. auf einem Speichermedium. Der Speicherbedarf im Arbeitsspeicher wird über die gesamte Lebensspanne eines Agenten nicht unter eine gewisse Schranke fallen. Denkbar wäre auch eine zugesicherte Kommunikationsbandbreite im Sinne einer Flatrate in einer Umgebung mit mehreren Prozessen.

In der Wirtschaft sind fixe Kosten von aktuellen Schwankungen unabhängige Kosten. Diese Leerkosten fallen immer an, auch dann, wenn keine Bearbeitung ansteht. Dazu zählen z.B. Mietkosten, Kosten für laufende Wartungsverträge usw. Die hier betrachtete reine Softwarelösung lässt in diesem Sinne (virtuelle) Fixkosten zu, deren tatsächliche physikalische

Kosten entweder vernachlässigbar gering oder gleich null sind. In der Simulation genutzte Pendants natürlicher Ausprägungen fixer Kosten lassen sich daher mit virtuellen Kosten bemessen.

Als Beispiel sei ein Agent a angeführt, der in einer simulierten Wertschöpfungskette virtuelle Güter an virtuellen Produktionsanlagen produziert und diese in einem virtuellen Lager zwischenlagert. Diese Parameter lassen sich als Teil der Strategie s_a eines generischen Produktionsagenten modellieren: $s_a = (produktionsmenge : 2, lagergroesse : 100)$. Dies würde den so konfigurierten Agenten veranlassen, pro Zeiteinheit zwei Stück eines Gutes zu produzieren und über Lagerplatz von 100 Stück dieses Gutes zu verfügen. Eine Kostenfunktion

$$cost(S) \rightarrow \mathbb{R} \tag{3.3}$$

ordnet einem Strategieelement $s \in S$ eine reelle Zahl zu. Mit $cost(produktionsmenge) = 2.0$ und $cost(lagergroesse) = 1.0$ ergeben sich die fixen Kosten des Produktionsagenten wie folgt:

$$\begin{aligned} cost(S) &= cost(produktionsmenge) \cdot s_a(produktionsmenge) \\ &\quad + cost(lagergroesse) \cdot s_a(lagergroesse) \\ &= 2.0 \cdot 2 + 1.0 \cdot 100 \\ &= 104 \end{aligned}$$

Generell werden die fixen Kosten eines Agenten a wie folgt ausgedrückt:

$$costfix(s_a) = \sum_{s \in s_a} cost(s) * s_a(s) \tag{3.4}$$

Aktionsbasierte und variable Kosten

Variable Kosten verändern sich mit der Verarbeitungsmenge. Hierbei handelt es sich um Kosten in Abhängigkeit der ver- oder bearbeiteten Informationen. Im Unterschied zu den fixen Kosten ist deren Anzahl im Voraus nicht bekannt. Ein Agent, der im Auftrag von Kunden Reiseinformationen zusammenstellt muss hierfür Datenbankabfragen tätigen. Zur Laufzeit wird daher eine gewisse Anzahl von Abfragen in Abhängigkeit der Kundenanfragen pro Zeiteineheit anfallen und im System Kosten verursachen. Diese fallen aus Sicht eines Agenten a in Abhängigkeit durchgeführter Aktionen $akt \in Akt$ an. Kosten werden in Analogie zu Definition 3.3 mittels einer Kostenfunktion definiert, die der einmaligen Ausführung einer Aktion eine relle Zahl zuordnet

$$cost(Akt) \rightarrow \mathbb{R} \tag{3.5}$$

cost ordnet damit jedem Element aus der Aktionsmenge Akt und eine reelle Zahl zu, z.B. Kosten für eine Datenbankabfrage $cost(db\_abfrage) = 0.1$ wobei $db\_abfrage \in Akt$ eine Aktion von a ist. Für alle Datenbankabfragen im Zeitraum t wird folgende Schreibweise genutzt: $cost(db\_abfrage(t))$ und für alle Datenbankabfragen von Agent a in t die folgende $cost_a(db\_abfrage(t))$. Damit ergibt sich die verallgemeinerte Form der variablen Kosten pro Agent: Sei Akt die Menge der verfügbaren Aktionen von a, dann berechnet die Funktion

$$costvar(a,t) = \sum_{akt \in Akt} cost_a(akt(t)) \tag{3.6}$$

eine reelle Zahl aus der Aufsummierung aller Kosten der von a im Zeitraum t durchgeführten Aktionen $akt \in Akt$. Sind die Kosten für die Aktion Datenbankabfrage mit $cost(db\_abfrage) = 0.1$ und den Nachrichtenversand mit $cost(nachricht) = 0.01$ angegeben und a führt im Zeitabschnitt t fünf Datenbankabfragen aus und sendet drei Nachrichten, so belaufen sich seine Kosten auf $costvar(a,t) = 0.53$.

Transaktionskosten

Transaktionskosten werden kostenseitig bereits von den variablen und den fixen Kosten mit abgedeckt. Dennoch lohnt sich eine Analyse von Transaktionen. Unter Transaktionen wird in diesem Zusammenhang die Austauschbeziehung zwischen ökonomischen Einheiten betrachtet. Hierzu liefert die Transaktionskostentheorie [192, 212] einen theoretischen Rahmen zur Bewertung von Transaktionen. Die Höhe von Transaktionskosten wird unter anderem bestimmt von der Unsicherheit und Häufigkeit von Transaktionen.

Es fallen je nach Ausgestaltung der zwischen den Agenten stattfindenen Transaktion unterschiedliche Kosten an. Im elektronischen Zahlungsverkehr und insbesondere bei agentenspezifischen Koordinationsmechanismen fallen je nach Ausgestaltung eine hohe Anzahl an Nachrichten an (siehe Abschnitt 2.2.5), deren Anzahl z.B. im Falle englischer Auktionen nicht nach oben begrenzt sind. Die Allokation transaktionsspezifischer Resourcen ist daher je nach Art der Aktion als höchst unterschiedlich einzustufen. Hierbei lohnt es, nicht ausschliesslich die Optimalität des Ergebnisses sondern insbesondere auch die Transaktionskosten zu betrachten.

Agenten, deren hauptsächliches Koordinationsinstrument die englische Aktion über einen zentralen Auktionator ist, werden daher zunächst unabhängig von der Qualität der Ergebnisse höhere Transaktionskosten durch sehr viele Nachrichten und längere Transaktionsdauern aufweisen als eine Peer-to-Peer artige Struktur von Agenten ein einfaches Anfragemuster beinhaltet.

Systemkosten

Unter Berücksichtigung der agentenspezifischen Kosten lässt sich eine generelle Systemkostenfunktion ableiten. Voraussetzung hierfür ist eine hinreichend genaue Abbildung der agentenspezifischen Kosten. Agenten als proaktive Elemente des Multiagentensystems zeichnen verantwortlich für die Systemfunktion Φ. Da ein MAS ohne Agenten nach Definition 14 keine Funktion erfüllt, werden alle Systemkosten auf Agentenspezifische Kosten umgelegt werden. Die Systemkosten ergeben sich wie folgt

Definition 15 (Systemkosten)**.**
Die Systemkosten $costsys$ eines Multiagentensystems ergeben sich aus den summierten fixen Kosten (siehe Gleichung 3.4) und den variablen Kosten (siehe Gleichung 3.6) aller Agenten in einem Zeitabschnitt t:

$$costsys(MAS,t) = \sum_{a \in A} (costfix(s_a) + costvar(a,t))$$

Ohne Angabe von t entfallen alle variablen Kosten und es werden von $costsys(MAS)$ lediglich die fixen Kosten betrachtet.

3.1.5 Verteiltes Optimierungsproblem - VOP

Nun kann die Optimierung folgendermaßen formuliert werden

Definition 16 (Verteiltes Optimierungsproblem (VOP))**.**

Ein verteiltes Optimierungsproblem ist gegeben durch:

$$\begin{aligned} &\textit{minimiere } costsys(x), \\ &\textit{unter Nebenbedingung:} \\ &\qquad\qquad x \textit{ ist Strategie } S_A \textit{ eines } MAS, \\ &\qquad\qquad \textit{Kostenfunktion } cost, \\ &\qquad\qquad \gamma_i(MAS) \leq 0, \forall i \in \{1, \ldots, q\} \end{aligned} \tag{3.7}$$

wobei X den *Lösungsraum* oder *Suchraum* und $x \in X$ eine Lösung darstellt. $cost(X) \rightarrow \mathbb{R}$ ist die Zielfunktion, welche in den Raum der reellen Zahlen $\mathbb{R}$ abbildet.

Der Lösungsvektor x ist ein Multiagentensystem MAS, für welches dessen Strategie S_A als konstituierend für MAS angesehen wird (vgl. dazu Abschnitt 3.1.2.4 und 3.1.3.2). Die Kostenfunktion *cost* ordnet den Eigenschaften (Gleichung 3.3) und den Aktionen (Definition 3.5) des Multiagentensystems Kosten zu. Die Strategie S_A und die Funktion Φ des MAS ist verteilt über die Agenten, daher ist *costsys* innerhalb des Systems aufgrund der Sichtbarkeitsfunktion *vis* nicht bekannt. Die Berechnung von *costsys* ist als Eigenschaft von MAS aufzufassen, die innerhalb von MAS nicht komplett berechenbar sein muss. Einem Agent sind lediglich seine selbst verursachten Kosten bekannt.

Durch zeitliche Veränderung des Systems aufgrund von Anpassungen der Agenten oder dem Hinzufügen neuer bzw. dem Entfernen von Agenten ist das Gesamtsystem MAS, die Systemstrategie S_A und auch die Systemfunktion Φ variabel. Es gelten die durch γ definierten Nebenbedingungen auf MAS, zum Beispiel die Kapazität der Umgebung U.

3.1.6 Ökonomische Perspektive

Gegenüber den klassischen Methoden der Dekomposition ist vor allem die Einfachheit ökonomischer Modelle [126] bemerkenswert. Jeder Agent kontrolliert dabei lediglich einen Teil der vorhandenen Ressourcen des Gesamtsystems: Um ein gemeinsames (System)Ziel zu erreichen, ist Zusammenarbeit notwendig. Es ist sehr wahrscheinlich, dass ein einzelner Agent die Dienstleistung anderer Agenten in Anspruch nimmt, um seine Strategie zu verfolgen. Auf Basis der vorangegangenen Beschreibung verteilter Systeme als Multiagentensystem lässt sich der Handel von Dienstleistungen und Gütern realisieren. Dies lässt sich am Beispiel einer Wertschöpfungskette veranschaulichen. Ein Produzent benötigt Rohstoffe oder vorgefertigte Teile von Zulieferern zur Herstellung seines Gutes, welches wiederum an andere Agenten verkauft wird.

3.1.6.1 Geld

Im Falle der Wertschöpfungskette kann der Ausgleich zwischen den beteiligten Agenten über deren spezifische Dienstleistungen erfolgen. Dies ist jedoch aus ökonomischer Sicht uneffizient, da hierdurch höhere Transaktionskosten zur Ausgestaltung und Durchführung anfallen [192]. Die Verwendung von Geld ist eine wesentlich einfachere Option. Die Repräsentation von Geld, Ressourcen, Wissen, (elektronische) Güter und Dienstleistungen wird durch das Konzept der Objekte (siehe Definition 4) abgedeckt. Die Sichtbarkeit und damit Zugriff eines Agenten auf Objekte ist bereits über die Sichtbarkeitsfunktion (siehe Definition 5) und genauer über die Sensoren (siehe Definition 7 definiert. Ein Objekt o das ausschließlich durch einen Agenten a wahrgenommen werden kann, unterliegt dessen Kontrolle, soweit es Manipulationen zulässt. Die Kontrollfunktion gibt an, ob ein Agent alleinige Kontrolle über bestimmte Objekte besitzt:

$$control_a(o) = \begin{cases} wahr, & o \in Sen_a \wedge \forall a' \in \{A \setminus a\}\ \neg\exists o \in Sen_{a'} \mid a \neq a' \\ falsch, & \text{sonst} \end{cases} \tag{3.8}$$

Im Fall von Geld-Objekten handelt es sich damit um Geld im Besitz von a. Die Kontrollfunktion $control_a(o)$ definiert die ausschließliche Sichtbarkeit von o durch a. Damit lässt sich das einem Agenten a zur Verfügung stehende Kapital wie folgt definieren:

$$funds(a) = \sum_{o' \in O} o' \mid control_a(o') = wahr \wedge o' \text{ ist Geldobjekt}\} \tag{3.9}$$

3.1.6.2 Umgebung

Geld kann von Agenten niemals selbst erzeugt werden. Es ist lediglich erlaubt, den Besitzer im Zuge einer Transaktion zu wechseln. Hierdurch wird sichergestellt, dass Geld nicht verschwinden bzw. erzeugt werden kann. Im System ist eine initiale Menge an Geldobjekten vorhanden und zusätzlich kann durch die Umgebung neues Geld zur Verfügung gestellt werden.

Hierdurch ist die Menge von Geld im System begrenzt. Während Transaktionen zwischen den Agenten bzw. von Agenten mit dem System kann Geld zwischen den Agenten wechseln. Die entstehende Verteilung und der Fluss von Geld im System zeigt Abbildung 3.5. Die jeweiligen Flüsse von Informationen und Geld (gestrichelte Linie) bilden einen Geldfluss in entgegengesetzter Richtung zum Informationsfluss.

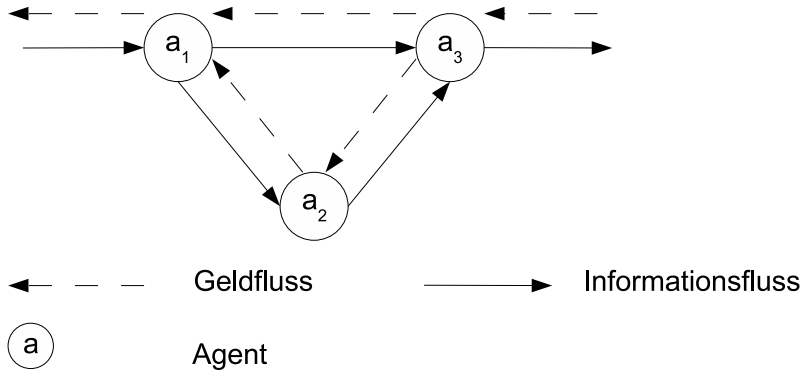


Abbildung 3.5: Floss von Geld innerhalb des Systems: Agent zu Agent und Agent Umgebung

3.1.6.3 Steuern

Die in Kapitel 3.1.4 vorgestellten Kostenfunktionen für fixe Kosten (Gleichung 3.3) und variable Kosten (Gleichung 3.5) lassen sich in Form von Steuern erheben. Ein Agent muss an die Umgebung regelmäßig oder bei Nutzung umgebungsrelevanter Ressourcen Steuern zahlen. Die Höhe der Steuern richtet sich nach der Kostenfunktion. Für genaue Regulierung der Kosten kann pro virtueller Umgebung $v \in V$ eine eigene Kostenfunktion $cost_U$ existieren.

Dies erlaubt eine feingranulare Steuerung der zur Verfügung stehenden Hardware, z.B. in Rechenzentren. Als Beispiel sei eine heterogene Hardwarelandschaft aufgezeigt, in welcher zwei Hardwareknoten A und B laufen. A hat ein angeschlossenes *RAID* (engl. *Redundant Array of Independent Disks*) zur Verfügung, während B über mehrere CPU's verfügt. Hier lassen sich die Kosten für Speicherplatz und CPU den Gegebenheiten anpassen. Während bei A die Kosten für Speicherplatz geringer anzusetzen sind im Vergleich zu B, hat im Gegenzug B geringere Preise für Nutzung der CPU. Ein wesentlicher Effekt hierbei ist eine bessere Aufteilung von Aufgaben anhand ihres Nutzungsprofils (CPU- vs. Speichernutzung).

Die Steuern $\mathcal{T}$ basieren auf der (virtuellen) umgebungsspezifischen Kostenfunktion *cost* und ergeben sich pro Agent zum Zeitpunkt t wie folgt:

$$\mathcal{T}_a(t) = costfix(s_a) + costvar(a, t) \tag{3.10}$$

Mit Hilfe von Steuern kann der 'Designer' vor und währen der Laufzeit die Rahmenbedingungen ändern. Dies ist nicht an einen konkreten Designer gebunden, sondern kann und sollte für die vorhandene Hardware jeweils unterschiedlich entworfen werden. Hiermit kann indirekt die Entwicklung des Systems im Sinne der Mechanismus Design Theorie [13] beeinflusst werden. Die im System vorhandene heterogene Hardware kann damit durch deren jeweilige Eigentümer im Sinne optimaler Nutzung besteuert werden. Auch wenn die Eigentümer jeweils selbst nutzenmaximierend handeln, werden die Steuern so aufgebaut sein, dass Agenten auf der jeweiligen virtuellen Umgebung Aktionen ausführen können.

Eine einfache und universelle Form der Besteuerung ist die kostenbasierte Nutzung von Systemressourcen. Hierbei werden Kosten auf die Nutzung und den Verbrauch von CPU-Zyklen, Speicher und Bandbreite erhoben. CPU und Bandbreite stellen variable Kosten dar, die nur bei Bedarf anfallen. Demgegenüber steht die Nutzung von Speicherplatz, z.B. in Form genutzten Arbeitsspeichers während der Laufzeit eines Agenten. Hier existiert ein fixer Anteil, der nach dem Programmstart des Agenten immer benötigt wird. Während der Ausführung von Aktionen fällt bedarfsgetrieben weiterer variabler Speicherbedarf an. Mit Hilfe dieser Form lassen sich universell alle Aktionen von Softwareagenten besteuern, da jede Anweisung (z.B. Nachrichtenversand, Sortieren, Persistieren von Daten) Ressourcen in den drei Dimensionen CPU, Speicher und Bandbreite benötigt. Im weiteren Verlauf der Arbeit werden jedoch zur Veranschaulichung ausschließlich höherwertige Aktionen oder Eigenschaften besteuert.

3.1.6.4 Ökonomischer Agent

Mit Hilfe des in Geldflusses innerhalb des Multiagentensystems lässt sich der Profit eines Agenten zum Zeitpunkt t $\pi_a(t)$ aus Sicht eines Agenten angeben durch:

$$\pi_a(t) = \mathcal{R}_a(t) - \mathcal{P}_a(t) - \mathcal{T}_a(t) \tag{3.11}$$

wobei $\mathcal{P}_a(t)$ Zahlungen von a an andere Agenten bezeichnet und $\mathcal{T}_a(t)$ Steuern an die Umgebung v ausdrückt. Eingehende Zahlungen im Zeitraum t werden durch $\mathcal{R}_a(t)$ ausgedrückt.

In diesem Sinne handelt ein ökonomischer Agent rein nach seinem eigenen Vorteil und versucht seinen Nutzen zu maximieren. Der einzige Wert für Agenten besteht im Geld bzw. in Informationen oder Dienstleistungen, die sich in der Zukunft in Profit π transformieren lassen.

Die Strategie s_a eines Agenten hat entscheidenden Einfluss auf den zu erwartenden Profit π_a weil dies die einzige Differenzierungsmöglichkeit zwischen den Agenten ist. Eine Strategie s_a, welche mit einer einzigen (kostenpflichtigen) Aktionen eine Dienstleistung erbringt ist damit aus ökonomischer Sicht Strategie s_b vorzuziehen, die mehr als eine Aktion für eine vergleichbare Dienstleistung benötigen.

Definition 17 (Ökonomischer Agent)**.**
Ein ökonomischer Agent a ist ein Agent nach Definition 11, dessen Transferfunktion eine nutzenmaximierende Kostenfunktion realisiert. Ziel ist die Maximierung der Werte von Objekten unter seiner Kontrolle:

$$maximiere\ control_a(o) = wahr$$

Die einzigen Objekte mit wirklichem Wert für ökonomische Agenten sind Geldobjekte $funds(a)$. Daher werden andere Objekten, wie z.B. Informationen, virtuelle Güter oder Dienstleistungen produziert oder gehandelt, um sie zu einem zukünftigen Zeitpunkt in Geld zu überführen.

3.2 Optimierung

Dieser Abschnitt beschäftigt sich mit der Adaption und Optimierung von Systemen auf Grundlage lokaler evolutionärer Interaktionen vor dem Hintergrund der eingangs gestellten Forderungen im Spannungsfeld Verteilung, Heterogenität und Dynamik. Diese generellen Anforderungen werden zunächst näher beleuchtet und weiter in abgeleitete und speziellere Anforderungen überführt (Abschnitt 3.2.1). Darauf aufbauend wird das eigentliche Optimierungsverfahren in 3.2 vorgestellt und anschließend in den Abschnitten 3.3.1-3.3.2 emergente Merkmale abgeleitet und analysiert.

3.2.1 Anforderungen

Dieser Abschnitt untersucht die Randbedingungen, welche in den Abschnitten 1.2, insbesondere in 1.2.3 festgelegt wurden. Hierbei werden die Anforderungen zunächst heruntergebrochen auf eine für maschinelle Optimierung geeignetes Abstraktionsniveau. Weiterhin werden gegebenenfalls weitere Anforderungen daraus abgeleitet.

3.2.1.1 Verteilte Lösung

Verteilte Probleme ergeben sich aus einer Reihe von Gründen, beispielsweise aus inhärent verteilten (räumlich) Problemen, geschützten Information und auch verteilten Ressourcen zur Problemlösung. Zusätzlich kann aufgrund der Beschränkung von Ressourcen eine zentrale Verarbeitung ebenfalls nicht möglich sein (siehe Abschnitt 1.2.1). Diese Faktoren können dazu führen, dass die Zusammenführung notwendiger Informationen, im Rahmen dieser Arbeit durch Informationsobjekte O repräsentiert, gänzlich verhindert bzw. nur teilweise gegeben ist. Formal realisiert diesen Umstand die Sichtbarkeitsfunktion vis. Die Informationen sind

damit über die Menge der zur Verfügung stehenden Berechnungsressourcen verteilt. Im agentenorientierten Ansatz ist dies die Menge der Agenten A.

Informationsobjekte werden von außen an unterschiedlichen Stellen als Teile des zu lösenden Problems verfügbar gemacht und von unterschiedlichen Agenten wahrgenommen. Hierdurch wird das Problem einmalig oder dauerhaft in das System eingebracht. Abstrakt lässt sich damit das verteilte (Optimierungs)Problem VOP auf einem verteilten (Multiagenten)System MAS modellieren. Wird die Lösung des VOP im System selbst erbracht, stellt der Zustand des Systems auch gleichzeitig die Lösung dar. Hierbei 'liest' das System das Problem und reagiert, indem eine Problemlösung zustande kommt. Interessanterweise existiert die Lösung nicht in Form eines Masterplanes an einer Stelle im System, sondern verbleibt vielmehr als impliziter Teil des Systems ebenfalls verteilt.

Das verteilte System ist die verteilte Lösung für ein verteiltes Problem. Beispiele hierfür finden sich z.B. im Supply-Chain Management [97, 115, 238]. Unternehmen arbeiten global zusammen, um logistische Probleme (Lagern, Transportieren, Umschlagen nach [191]) zu lösen. Dabei werden lediglich minimale Informationen über Schnittstellen ausgetauscht. Jedoch werden bei Weitem nicht alle Informationen geteilt, die zur zentralen Bearbeitung notwendig sind. Ähnlich verhält es sich mit Ameisenpopulationen, die das Problem der verteilten Futtersuche, Verteidigung und des Nestbaues bewerkstelligen ohne Masterplan. Auch hier stellt die Gesamtpopulation eine verteilte Lösung dar. In beiden Fällen sorgen a priori festgelegte lokale Verhaltensmuster der Akteure (=Agenten) für die gewünschte globale Lösung. Nur das Zusammenspiel der beteiligten Agenten führt zur (emergenten) Lösung. Im Vergleich zu populationsbasierten Ansätzen, die mit jedem Individuum eine gesamte Lösung repräsentieren, fordert der vorliegende Ansatz Toleranz gegenüber verteilten Lösungen.

3.2.1.2 Nachfrageorientierung

Aufgrund der dynamischen Umwelt ist es aus Systemsicht wichtig, hinreichend schnell auf externe Veränderungen reagieren zu können. Ein wichtiger Punkt hierbei ist die Anpassung der Optimierungsprioritäten anhand der Nachfragepriorität. Die Schwierigkeit in komplexen Systemen besteht in dieser Hinsicht darin, einen generellen Ansatz zur Informationsverteilung zu finden. Hierfür existierende Ansätze algorithmischer Natur setzen immer die Fähigkeit voraus, diesen Algorithmus zu implementieren - im heterogenen Umfeld interorganisationaler Systeme ein nicht zu unterschätzendes Hindernis. Zusätzlich induzieren viele Verfahren Engpässe in das System.

An dieser Stelle wird daher ein marktbasiertes Verfahren vorgeschlagen. Zum Einen zeichnen sich diese Verfahren durch hohe Skalierbarkeit aus [247], zum Anderen ist damit ein dezentraler Koordinationsmechanismus gegeben [64].

3.2.1.3 Lernen

Die geforderte Adaptivität hängt im Wesentlichen vom konkreten Problem ab und lässt sich daher nicht ohne weiteres quantifizieren. Idealerweise passt sich das System sofort an veränderte Bedingungen an, ohne jedoch hierfür extra Ressourcen zu benötigen. Diese beiden Zielstellungen widersprechen sich jedoch und generieren inkonsistente Vorgaben hinsichtlich der Systemstrategie. Ein Beispiel, das in dieser Arbeit an verschiedenen Stellen wieder aufgegriffen wird, ist der Konflikt zwischen Nachfrageschwankungen und der Entscheidung zur Einbringung bzw. Entfernung von Ressourcen in das System. Die generelle Ausrichtung der

Suchstrategie spielt hierbei eine entscheidende Rolle. Charakteristisch für eine Suchstrategie sind unter anderem die beiden Extreme *Exploration* vs. *Exploitation*. In nachfrageorientierter Optimierung kommt dauerhafter Exploration eine wichtige Rolle zu, denn die Nachfrage kann schwanken (und damit das Problem).

In *Reinforcement Lernverfahren* (dt. *Verstärkendes Lernen*) werden diese unterschiedlichen Extreme auch als *Exploitation-Exploration Dilemma* [228, 236] bezeichnet. Hierbei steht ein lernender Agent vor dem Konflikt, entweder eine nach lokalem Wissensstand optimale Aktion (Exploitation) oder eine Aktion mit unbekanntem Folgezustand (Exploration) auszuführen [156]. Einfache Reinforcement Lernverfahren sind effektiv, wenn der Agent verstärkende Signale richtig identifizieren kann. In vielen komplexen Umgebungen ist die eindeutige Zuordnung schwierig. Eine Aktion muss nicht immer sofort Konsequenzen nach sich ziehen, diese können erst sehr viel später auftreten. Dieses sogenannte *delayed Reinforcement* (dt. *verzögerte Verstärkung*) beinhaltet die Schwierigkeit abzuschätzen, wie vergangene Aktionen die aktuelle Situation herbeigeführt haben. Einen Ansatz bietet die Dualität zur lokalen Selektion [156].

3.2.1.4 Heterogenität

Eine weitere Schwierigkeit besteht, wenn die verteilte Lösung über mehrere Verarbeitungsschritte in unterschiedlichen Agenten entstanden ist. Ähnlich einer Fertigung mit verschiedenen Stufen spielen hier die Interaktionen als Teilgraph der Umgebungsstruktur G_U (siehe Definition 3) eine große Rolle. Insbesondere, wenn die Interaktionen nicht seriell, sondern netzwerkartig in Schleifen oder Zyklen (Rückkopplung) verlaufen. Die entstehenden Interaktionen sind komplex und damit nichtlinear. Solche Interaktionsstrukturen lassen sich aus Sicht einer Optimierungsinstanz schwer analysieren und hinsichtlich des Beitrags einzelner Agenten nicht mehr auswerten (vgl. hierzu Matthies [153]). Im Vergleich zu populationsbasierten Ansätzen, die mit jedem Individuum eine gesamte Lösung repräsentieren, fordert der vorliegende Ansatz auch die Bewertbarkeit einzelner Beiträge zur Gesamtqualität der Lösung.

Eine Bedingung ist daher die Ermöglichung verteilter Lösungsstrategien auf einer möglichst generellen Ebene. Weiterhin besteht die Forderung nach Adaptivität auf lokaler als auch auf Systemebene. Hierbei ist von Interesse, inwieweit Verhaltensmuster für unterschiedliche Probleme vorgegeben sein müssen oder ob diese generisch und adaptiv entstehen können. Weiterhin sollte eine Leistungsbewertung einzelner Agenten hinsichtlich ihres Beitrages möglich sein.

Um interorganisationale (Optimierungs)Prozesse und deren Schnittstellen einfach zu gestalten, ist es notwendig, einen Ansatz mit möglichst wenig Aufwand in bestehende Lösungen zu integrieren. Dazu zählt zum Einen der Integrationsaufwand selbst. Zum Anderen sind auch die Berechnungskomplexität und der Realisierungsaufwand des Verfahrens entscheidend. Akzeptanzfördernd sind generelle und einfache Verfahren.

3.2.1.5 Zusammenfassung

Für das Szenario eines komplexen Systems in komplexer Umwelt ist für die Optimierung ein dezentraler Ansatz wesentlich besser geeignet, als zentrale Methoden. Anhand abgeleiteter Anforderungen, basierend auf den Anforderungen der Problemstellung (Abschnitt 1.2) lassen sich weitere Eigenschaften zusammenfassen:

Exploration ist in dynamischer Umwelt ein wesentlicher Faktor der Suchstrategie. Hiermit wird eine konstante Evaluation der Umwelt sichergestellt.

Lokale Selektion ist aufgrund von zwei Argumenten für die Realisierung geeignet. Ein zentraler Selektionsmechanismus wird in der Problemstellung explizit ausgeschlossen. Lokale Selektion ist auf zentrale Operationen nicht angewiesen. Ein weiterer Grund ist die Dualität von Lokaler Selektion und Reinforcement Learning. Lokale Selektion erlaubt Populationen die Anpassung an eine Umgebung über Generationen hinweg, während Reinforcement Learning die Anpassung während die Lebenszeit eines Agenten realisiert [156].

Marktbasierte Verfahren eignen sich nach Eymann [64, 234] als dezentrale Koordinationsverfahren. Hierfür ist keine Zentrale Steuerungsinstanz notwendig und es bestehen daher keinerlei Skalierungshindernisse [247].

Verteiltes Lösen eines Problems muss vom Verfahren unterstützt werden. Im Gegensatz zum Task- oder Resultsharing, wobei die Aufgabe oder das Resultat zusammengeführt wird, existiert kein Agent mit dem Gesamtwissen zum Bearbeiten des Problems.

Lokale Bewertung von Individuen spielt eine entscheidende Rolle und muss ermöglicht werden, da keine externe Instanz Agenten bewerten und vergleichen kann. Diese Forderung dient neben der Bewertung lokaler Entscheidungen auch dem Vergleich von Agenten untereinander (soweit die Informationen verfügbar sind). Lokale Bewertung der Fitness wird auch als endogene Fitness bezeichnet [156].

Generisch Im Unterschied zu vorgegebenen Verhaltensmustern soll der Ansatz generisch auf unterschiedliche Probleme anwendbar und möglichst einfach in heterogene Systeme mit heterogenen Akteuren integrierbar sein.

Effiziente Realisierung durch ressourcenschonenden Einsatz von bereitstehenden Mitteln. Für dynamische Umgebungen sind z.B. Regelbasierte Ansätze zu teuer, da hier alle Entscheidungsalternativen vorgehalten werden müssen. Lokale Entscheidungen sollen möglichst effizient bei zu betrachtetem Aufwand/Nutzen sein und gleichzeitig adaptiv realisiert werden.

Diese abgeleiteten Anforderungen gelten als Entscheidungsgrundlage für den Ansatz und werden zur Evaluation herangezogen.

3.2.2 Evolutionärer Agent

Ziel dieses Abschnittes ist die Erweiterung des ökonomischen Agenten aus Definition 17 (Abschnitt 3.1.6.4) um die Fähigkeit zur Reproduktion. Eine solche Erweiterung beinhaltet sowohl Anforderungen an ein Agentensystem (siehe Abschnitt 4.1), als auch an das formale Agentenmodell.

Definition 18 (Evolutionärer Agent)**.**
Ein evolutionärer Agent a ist ein ökonomischer Agent mit der Fähigkeit zur evolutionären Reproduktion. Hierfür wird die Definition des ökonomischen Agenten (Definition 17) erweitert um die Reproduktionsschwelle θ:

$$a = (Akt, Sen, Z, \varphi, s, \theta)$$

θ legt fest, wie viel Geld ein Agent zur Reproduktion mindestens besitzen muss. Die Reproduktion selbst erfolgt lokal. Das bedeutet es sind alle notwendigen Reproduktionsschritte vom Agent lokal durchzuführen und von anderen Agenten bzw. externen Diensten möglichst unabhängig zu machen. Dazu zählen

- Wahl des Reproduktionszeitpunktes
- Selektion
- Rekombination
- Mutation
- Generieren neuer Agenten

Die beiden Punkte Rekombination und Mutation lassen sich sehr einfach lokal durchführen. Hierfür sind lediglich die erforderlichen Operatoren vom Agenten lokal auf die Strategien anzuwenden. Voraussetzung für eine lokale Reproduktion ist das Vorhandensein der eigenen Strategie und der Partnerstrategie. Der erste Fall sollte kein Problem sein, da ein Agent seine Strategie kennt. Im zweiten Fall ist die Selektion für die 'Beschaffung' der Partnerstrategie verantwortlich. Eine detaillierte Beschreibung der lokalen Selektion erfolgt in Abschnitt 3.2.4.

Die Wahl des Reproduktionszeitpunktes ist aus lokaler Perspektive schwierig zu entscheiden und gleichzeitig wichtig für den Erfolg eines Agenten und dessen Strategie. Erfolg eines Agenten wird hier mit der Verbreitung der Strategie oder Teile davon gleichgesetzt. Zwei wesentliche Einflussfaktoren kommen als entscheidend für die Wahl des richtigen Reproduktionszeitpunktes in Frage:

Umgebung: Eine Reproduktionsstrategie kann vom Zustand der Umwelt abhängig gemacht werden. Dazu prüft der Agent, ob genügend Ressourcen in der Umgebung vorhanden sind. Falls diese Prüfung erfolgreich war, findet eine Reproduktion statt, andernfalls wird auf bessere Zeiten gewartet. Damit ist sichergestellt, dass die Nachkommen über genügend Ressourcen verfügen und nicht in Konkurrenz zum Elternagent stehen. Schnelle Reproduktion in 'unbesiedeltem Gebiet' macht immer Sinn und entspricht dem biologischen Pendant der *r-Selektion* [149]. Die 'Warten' Strategie birgt die Gefahr des Aussterbens, ohne die eigene Strategie zu verbreiten. Zusätzlich ist die Umgebung für den Agenten nicht vollständig sichtbar und darüberhinaus dynamisch. Zum Einen ergibt sich damit die Schwierigkeit, eine geeignete Umgebung zunächst zu erkennen. Zum Anderen bietet eine dynamische Umgebung den Vorteil neuer und unbesetzter Nischen. Diese sind unter anderem für Nachkommen mit neuen Strategien interessant. Der Erfolg einer neuen Strategie lässt sich aus Sicht eines Agenten allerdings weder herleiten noch vom Zustand der Umgebung ableiten. Daher bleibt der Faktor Umgebung zu ungenau, um als verlässlicher Indikator zu dienen. Die Strategie 'Warten' birgt eher den Nachteil, die eigene Strategie nicht zu verbreiten und selbst aufgrund zukünftiger widriger Umstände ohne Reproduktion aus dem System auszuscheiden. Zusätzlich müssen zum Zeitpunkt der Reproduktion für den Agent und seine Nachkommen genügend Ressourcen vorhanden sein, um nach der Reproduktion entsprechend handlungsfähig zu bleiben. Dies lässt sich jedoch immer lokal feststellen.

Zustand des Agenten: Dieser bietet wesentlich mehr Möglichkeiten, einen geeigneten Zeitpunkt abzuleiten. Der Agent hat die volle Kontrolle über den Zustand und damit alle Informationen zur Verfügung. Ein Indikator ist der Wert der Objekte, die der ökonomische Agent (siehe Abschnitt 3.1.6.4) a kontrolliert $control_a(o)$. Dies können beliebige Objekte sein. Mindestens zählen dazu jedoch Geldobjekte ($funds(a)$), die dem Agenten erlauben, Aktionen durchzuführen. Andere Objekte sind dagegen nur dann von Wert für den Agenten, wenn sie in Geldobjekte transformierbar sind. Je mehr ein Agent davon besitzt, desto höher ist seine Fitness im Sinne von Aktionen, die der Agent durchführen kann. Ein Agent mit unbegrenzter Anzahl an Geldobjekten könnte theoretisch unbegrenzt lange in seiner Umgebung existieren. Daher wird als Indikator ähnlich den Ressourcen des Echo-Systems von Holland [113, 114] bzw. dem Level von Energie bei Menczer [156] eine ausreichende Menge an Geldobjekten vorausgesetzt. Diese *Reproduktionsschwelle* sei mit θ bezeichnet. Betrachtungen zur Bestimmung von θ finden sich im folgenden Kapitel 3.2.3.

Die lokale Selektion erfolgt aus Agentenperspektive. Die Reproduktion soll gegebenenfalls auch ohne Partner realisiert werden können. In diesem Fall erfolgt eine einfache Replikation. Andernfalls würde ein reproduktionsbereiter Agent unter ungünstigen Umständen keinen geeigneten Partner finden und unendlich lange suchen. Das dieser Fall durchaus nicht die Ausnahme darstellt, lässt sich z.B. durch den Start eines Systems mit nur einem Agent oder bei Netzwerkausfällen mit einem Agent a pro virtueller Umgebung v beobachten. In beiden Extremfällen existiert lediglich ein einzelner Agent abgeschottet in seiner Umgebung. Zumindest im ersten Fall würde ein Agent unendlich lange suchen müssen. Bei Netzwerkausfällen währt die Suche solange, bis die Netzwerkinfrastruktur wieder funktioniert. Einen Vorteil im Sinne der Flexibilität bietet eine asynchrone und nicht seriell realisierte Abfolge von Reproduktion und Selektion. Die Selektion erfolgt zeitlich entkoppelt von der eigentlichen Reproduktion. Hierbei wird lokal die Strategie(n) des gewählten Partners bis zur Reproduktion zwischengespeichert. Algorithmus 1 zeigt die Hauptschleife eines Agenten als Pseudocode:

Algorithmus 1 Evolutionaerer Agent (Hauptschleife)

Vorbedingung: Strategie s und Geldobjekte $funds(a)$

```
1:  Initialisiere Zustand z, Transferfunktion φ
2:  Starte lokale Selektion /* siehe Algorithmus 3 */
3:  loop
4:    Sensorinput Sen_a auswerten
5:    Transferfunktion φ(Sen_a, z) ausführen
6:    Aktionen ausführen akt ← φ
7:    Statusupdate z ← φ
8:    if funds(a) ≥ θ_a then
9:      Reproduktion /* siehe Algorithmus 2 */
10:   else if funds(a) < 0 then
11:     exit /* Tod */
12:   end if
13: end loop
```

Die Hauptschleife des Agenten ist einfach. Zunächst werden die verfügbaren Sensorinformationen ausgewertet (Zeile 4) und zusammen mit dem initialen Zustand z an die Trans-

ferfunktion φ als Input übergeben (Zeile 5). Die resultierenden Aktionen werden ausgeführt (Zeile 6). In Abhängigkeit des verfügbaren Geldes $funds(a)$ erfolgt die Reproduktion (Zeile 9), der Tod des Agenten (Zeile 12) oder keine der beiden Alternativen $0 \leq funds(a) < \theta$.

Beispiel 1
Agent a übernimmt die Rolle eines Lageragenten mit einer Lagergröße von $s_a(lagergroesse) = 10$ in einer Supply-Chain. Er besitzt ein Geldvermögen von $funds(a) = 95$ und seine Reproduktionsschwelle sei $\theta_a = 100$. Ferner besitzt a die virtuellen Güter $control_a(p_1, \cdots, p_{10})$. Er empfängt zum Zeitpunkt t eine Nachricht zur Order über 10 Güter mittels Sen_a.

Per Transferfunktion φ wird die Aktion 'Verkauf von Gütern' ($verkaufen \in Akt$) zum Stückpreis von 1 Geldeinheit ausgewählt. Die Transaktionskosten für die Aktion $verkaufen$ sind festgelegt mit $cost(verkaufen) = 0.05$ (z.B. Nachrichtenversand) und die Kosten für die Aktion Lagern beträgt pro Stück $cost(p) = 0.09$. Damit belaufen sich die variablen Kosten (siehe 3.6) auf

$$\begin{aligned} costvar(a,t) &= cost(verkaufen) \\ &\quad + cost(p) \cdot 10 \quad (10 \text{ Stück von } p \text{ im Lager}) \\ &= 0.05 + 0.09 \cdot 10 \\ &= 0.95 \end{aligned}$$

Zusätzlich wird für den Lagerplatz ein Fixkostenbetrag von $cost(lagerplatz) = 0.005$ erhoben (siehe Definition 3.4):

$$\begin{aligned} costfix(s_a) &= cost(lagerplatz) \cdot s_a(lagergroesse) \\ &= 0.005 \cdot 10 \\ &= 0.05 \end{aligned}$$

Die Gesamtkosten an Steuern (siehe Gleichung 3.10) belaufen sich damit auf

$$\mathcal{T}_a(t) = costfix(s_a) + costvar(a,t) = 0.95 + 0.05 = 1$$

und stehen den Erlösen von $\mathcal{R}_a(t) = 10$ für den Verkauf der Güter gegenüber. Zahlungen an andere Agenten wurden von a im betrachteten Zeitraum nicht geleistet ($\mathcal{P}_a(t) = 0$). Das Vermögen von a beträgt damit nach Durchführung der Aktionen t':

$$\begin{aligned} funds(a)(t+1) &= funds(a)(t) + \mathcal{R}_a(t) - \mathcal{P}_a(t) - \mathcal{T}_a(t) \\ &= 95 + 10 - 0 - 1 \\ &= 104 \end{aligned}$$

Damit verfügt a über genügend Geld zur Reproduktion $funds(a)(t') = 104 > \theta_a$. Ein neuer Agent c wird erzeugt und das Geld zwischen a und c zu gleichen Teilen aufgeteilt: $funds(a)(t+1) = funds(c)(t+1) = 52$.

Zu Beginn wird getrennt von der Hauptschleife die lokale Selektion gestartet. Diese kann somit asynchron arbeiten. Von Vorteil ist dabei: die Selektion kann langsamer arbeiten (ressourcenschonender) und Verzögerungen werden vermieden. Durch lokale Selektion wird eine explorative Ausrichtung der Suchstrategie angestrebt.

Die Kosten für Aktionen sind von der jeweiligen Umgebung v des Agenten festgelegt und beeinflussen damit wesentlich den Selektionsdruck. Je nach Höhe der Kosten für Lagerplatz ($cost(lagerplatz)$) geraten Agenten mit großem Lagervolumen stärker ins Hintertreffen gegenüber Agenten mit geringem Lagerplatz. Ist dieser Parameter durch die Strategie festgelegt und über die Lebensdauer eines Agenten hinweg konstant, verschiebt sich nach einigen Agentengenerationen die Lagergröße in Richtung kleinerer Läger. Ein ökonomisch effizientes Verfahren entsteht. Hierdurch lässt sich im Sinne der Mechanismus-Design Theorie [13] optimales Verhalten auf Agentenebene erzielen. Wird Lagerplatz mit Speicherplatz gleichgesetzt, so wird die bereitgestellte Hardware implizit effizient genutzt. Beispiel 2 zeigt effiziente Verwendung heterogener Hardware durch unterschiedliche Agententypen.

Beispiel 2 (Adaptives Hosting)
Zwei Rechner A und B seien als Hardware für ein verteiltes MAS vorhanden. A verfügt über viel Speicher und wenig CPU Kapazitäten wohingegen B über viel CPU und wenig Speicher. Die Aufgabe für das Multiagentensystem besteht darin, Daten zu speichern und zu analysieren. Es stehen entsprechend zwei Typen von Agenten zur Verfügung: Lageragent zur Speicherung von Daten und der Typ Miningagent zur Analyse. Entsprechend ihrer Aufgabe benötigt der Lageragent Speicher und der Miningagent Rechenzeit.

Jeder Rechner stellt eine virtuelle Agentenumgebung (v_A, v_B) dar und definiert eigene Kosten für die Aktionen $AKT = \{speicher, cpu\}$ (Abb. 3.6). Die Besteuerung der Agenten erfolgt nach der jeweiligen Kostenstruktur der Umgebung, in der sie gehostet sind. Der hierdurch induzierte differenzierte Selektionsdruck führt damit implizit zu adaptivem Hostingverhalten: Lageragenten breiten sich in Umgebung v_A aus und die Miningagenten entsprechend umgekehrt in Umgebung v_B.

Aufgrund des Dualismus von lokaler Selektion und Reinforcement learning wird für das adaptive Hosting in Beispiel 2 keine lokale Lernstrategie benötigt. Stattdessen geschieht diese Anpassung effizient in evolutionären Maßstäben. Die interne Logik eines Agenten (bzw. dessen Transferfunktion φ) kann daher auf das nötigste beschränkt werden. Eine Anwendung dieses Ansatzes kann somit auch in Umgebungen mit stark begrenzten Ressourcen erfolgen.

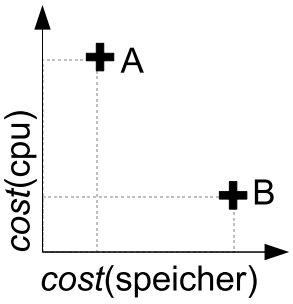


Abbildung 3.6: Kosten der physikalischen Ressourcen

Die Reproduktion eines Agenten ist aufgrund von Effizienzgründen zunächst entkoppelt von der Selektion und kann notfalls auch als Replikation verlaufen. Algorithmus 2 zeigt die Vorgehensweise in Pseudocode. Zwingend verfügbar muss dabei die eigene Strategie, sowie

benötigte Reproduktionsparameter sein (im Beispiel exemplarisch die Mutationsrate). Parameter können allerdings auch über die Strategie s_a verfügbar gemacht werden und damit *selbstadaptiv* (engl. *self-adaptive*) verwendet werden.

Algorithmus 2 Reproduktion

Vorbedingung: Strategie s_a, Mutationsrate mr, (Partnerstrategie s_p)

1: **if** Partnerstrategie s_p = null **then**
2: /* Replikation */
3: $s_p := s_a$
4: **end if**
5: /* Erzeuge offspring s_c */
6: $s_c :=$ recombine(s_a,s_p)
7: $s_c' :=$ mutate(s_c, mr)
8: /* Erzeuge neuen Agent mit Strategie s_c' */
9: $a_c :=$ newAgent(s_c')
10: /* Transfer von Geld an a_c */
11: $a_c \leftarrow \frac{funds(a)}{2}$

3.2.3 Lokale Fitness

Nach Mitchell und Forrest [163] ist der aus biologischer Sicht unrealistischste Aspekt evolutionärer Algorithmen die externe Berechnung der Fitness. Dieser auf externe zentrale Selektion angewiesene Ansatz wird hier nicht weiter verfolgt. Das Konzept *endogener Fitness* bzw. *lokaler Fitness* [156] wird stattdessen verwendet und nachfolgend analysiert.

Ein wesentliches Merkmal lokaler Selektion ist die lokale Bewertung der Fitness mittels $funds(a)$ und θ. Hierbei erweisen sich zwei Sachverhalte in [156, 221] als momentan nicht ausreichend gelöst:

1. Wie wird θ lokal festgelegt?
2. Wie ist das aktuelle Vermögen anderer Agenten richtig einzuschätzen?

Abbildung 3.7 zeigt den Verlauf von $funds(a)$ und π_a resultierend aus den Profiten über einen Zeitraum von $0 \leq t \leq 93$. Die $funds(a)$ Kurve ist das Integral über den Profit (mit Ausnahme der Reproduktionsereignisse). Agent a startet mit $funds(a) = 1$ und erzeugt in $t = 35$ und $t = 58$ jeweils einen neuen Agenten. Die Reproduktionsschwelle ist $\theta = 2$. Gegen Ende seiner Lebensspanne verliert der Agent Geld aufgrund negativer Profite und stirbt schließlich bei $t = 93$. Die Verluste können zum Beispiel durch verstärkten Konkurrenzdruck, fehlende Nachfrage oder erhöhte Steuern verursacht werden.

Bezüglich Frage 1 lässt sich anhan Abbildung 3.7 feststellen, dass sehr kleine Theta problematisch für Agenten sind. Beispielsweise wenn $\frac{\theta_a}{2} < average(\mathcal{P}_a + \mathcal{T}_a)$, dann würde der Agent sehr wahrscheinlich kurz nach seiner Erzeugung sterben. Der Grund liegt im *Startkapital*, welches vom *Elternagent* bereitgestellt wird. Dieses beträgt $\frac{\theta_a}{2}$ und der Agent wird zunächst Informationen oder Produkte kaufen müssen und Steuern zahlen. Wird dabei das aktuelle Budget überstiegen, so ist der Agent bankrott bevor eine Aktion Gewinn abwerfen kann. Eine Möglichkeit der Überbrückung finanzieller Durststrecken besteht über finanztechnische

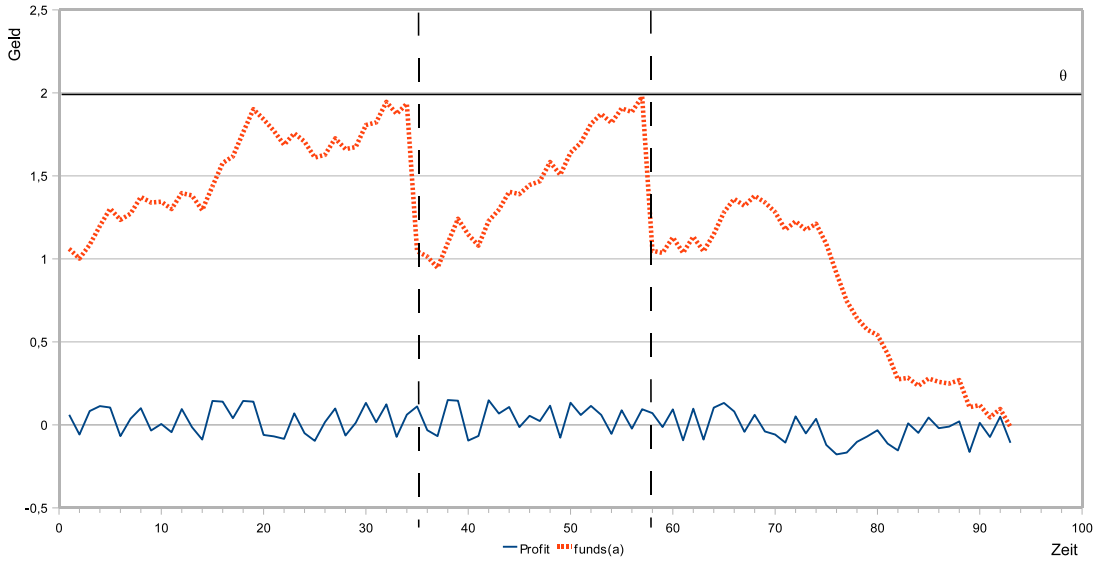


Abbildung 3.7: Beispielhafter Verlauf von $funds(a)$ und π_a eines Agenten über seine Lebensspanne. Die beiden senkrechten Linien markieren Reproduktionszeitpunkte.

Maßnahmen, wie beispielsweise Darlehen. Dieser Ansatz wird jedoch nicht weiter verfolgt, da hierdurch die Komplexität zunimmt und der dezentrale Ansatz analytisch verzerrt wird.

Andererseits kann evolutionärer Stillstand erzeugt werden, wenn die (Reproduktionsschwelle sehr viel größer als der Profit ist ($\theta_a \gg \pi_a$). In diesem Fall findet Reproduktion erst sehr spät oder gar nicht statt. Ein solches System ändert sich ausgehend vom Initialzustand sehr wenig Infolgedessen wird der Suchraum nicht oder nur unzureichend exploriert. Eine ausgewogene Balance von θ ist daher ausschlaggebend für die gewünschte Systemadaptivität und -stabilität.

Einen Ansatz zeigt das *ECHO*-System von Holland [113]. Agenten bestehen dort aus unterschiedlichen Ressourcen und können sich nur reproduzieren, wenn über alle benötigten Ressourcen doppelt verfügt wird. Dieser Ansatz kann in abgewandelter Form an das bestehende Verfahren adaptiert werden. Hierbei ist das Konzept der Ressource durch das abstraktere Konzept des Geldes zu ersetzen. Die tatsächliche Höhe von θ kann damit wie auch andere evolutionäre Parameter *empirisch-simulativ* oder selbst-adaptiv justiert werden. Eine weitere Möglichkeit bietet sich durch den rationalen Ansatz. Während der Erstellung des Agenten werden bereits Kriterien festgelegt, die später zur Laufzeit eine dynamische Abschätzung von θ ermöglichen. Damit stehen folgende Möglichkeiten zur Verfügung:

Rational: A priori festgelegtes Verfahren. Vorteil: Kann in bestimmten Fällen auf einfache Weise sehr gute Resultate erzielen. Nachteil: Nicht immer möglich

Empirisch: Aufgrund von Erfahrungswerten oder Simulationen und Messreihen bestimmte Reproduktionsschwelle. Vorteil: empirisch unterlegte Wirksamkeit. Nachteil: Zum Teil sehr aufwändig. In vielen Fällen muss das aufwändige Verfahren für pro Agententyp und Umgebung durchgeführt werden. Eine allgemeingültige hinreichend optimale Wahl der Reproduktionsschwelle existiert nicht (Im Gegensatz zur Mutationsrate bei evolutionären Verfahren).

Selbst-adaptiv: θ wird als Teil der Agentenstrategie s in den Explorationsprozess aufgenommen und damit selbständig an veränderte Umgebungsbedingungen angepasst. Vorteil: Keine aufwändige Suche bzw. Bestimmung mehr notwendig. Nachteil: Startwerte müssen hinreichend gut gewählt werden, ansonsten droht Zusammenbruch bzw. Starre des Systems.

Beispiel 3 (Rationale θ-Bestimmung für Lageragenten)
a sei ein Lageragent mit Lagergröße $s_a(lagergroesse)$. Ein solcher Agent sollte vom verfügbaren Geld sein Lager komplett füllen können. Damit nach einer Reproduktion beide Agenten problemlos ihre Lager füllen können, sollte θ mindestens doppelt so hoch sein, wie alle Kosten zur vollständigen Transaktionsentwicklung über einen bestimmten Zeitraum Δt. Dazu zählen der Einkauf von Produkten $P = \{p_1, \ldots, p_{s_a(lagergroesse)}\}$, um das gesamte Lager vollständig zu füllen $\mathcal{P}_a$, die Transaktionskosten für Einkauf, Verkauf und Lagerung der Produkte während Δt:

$$costvar(a, \Delta t) = cost(buy) + cost(sell) + cost(p) \cdot s_a(lagergroesse) \cdot \Delta t$$

Die Fixkosten für den Betrieb des Lagers im Zeitraum Δt sind:

$$costfix(s_a, \Delta t) = cost(lagerplatz) \cdot s_a(lagergroesse) \cdot \Delta t$$

Damit ergibt sich eine rationale Abschätzung der benötigten Reproduktionsschwelle θ_a für a, um seinen Dienst für Δt ohne Einkünfte $\mathcal{R}_a(\Delta t)$ aufrecht zu erhalten:

$$\theta_a = (2 + \Delta_r) \cdot (costfix(s_a, \Delta t) + costvar(a, \Delta t) + \mathcal{P}_a(\Delta t))$$

Δ_r ist ein Skalierungsfaktor in Abhängigkeit des vorgesehenen Umgebungskontextes. In Abhängigkeit von Δt und Δ_r ergibt sich damit eine konservative bis optimistische Schätzung für θ_a.

Beispiel 3 berücksichtigt nicht eine starke Vergrößerung des Lagers des Kindagenten c gegenüber dem *Elternagenten* a ($s_a(lagergroesse) \ll s_c(lagergroesse)$), da $s_c(lagergroesse)$ bei der Entscheidung zur Reproduktion nicht bekannt ist (siehe Algorithmus 1 Zeile 8). Hierfür kann z.B. Δt bzw. Δ_r entsprechend konservativer ausgelegt werden. Ist z.B. mit starker Dynamik oder längeren Zeiträumen ohne Nachfrage zu rechnen, müssen diese Faktoren entsprechend höher angesetzt werden.

Externe lokale Fitness

Die zweite zu Anfang dieses Abschnittes aufgeworfene Frage beschäftigt sich mit der externen Bewertung des Faktors $funds(a)$. Die Auswertung und Interpretation von $funds$ im Hinblick auf θ lässt sich externalisiert nicht einfach als Fitnessindikator verwenden. Zunächst ist festzuhalten, dass für eine lokale Selektion aus Sicht eines Agenten gegenüber anderen Agenten Details sichtbar gemacht werden müssen. Die Verwendung von Interpretation von $funds$ ist in diesem Zusammenhang allerdings problematisch.

Ein Agent a mit einem im Idealfall konstanten positiven Profit $\pi_a(t) = const > 0$ zeigt in seinem zeitlichen Verlauf von $funds(a)$ ein *Sägezahnmuster*, welches zwischen $\frac{\theta}{2}$ und θ pen-

delt. Genauer gesagt, steigt der Verlauf ausgehend von $\frac{\theta}{2}$ bis θ an und fällt bei Reproduktion zurück auf $\frac{\theta}{2}$. Sendet Agent a die Information $funds(a)$ an Agent b, so sind dessen Interpretationen mehrdeutig bereits unter der Annahme, dass $\theta_a = \theta_b$. b kann daraus nicht ableiten, ob a konstant positiven Profit erwirtschaftet oder ob $\frac{\theta_a}{2} \leq funds(a) \leq \theta_a$ um den aktuellen Wert oszilliert. Dies kann z.B. der Fall sein, wenn θ_a im Durchschnitt 0 beträgt. Ebenso kann umgekehrt zum Zeitpunkt der externen Interpretation für kurze Zeit $\pi_a < 0$ sein und damit einen ansonsten profitablen Agenten als wenig erfolgreich erscheinen lassen.

Zusätzlich ist davon auszugehen, dass zwei unterschiedliche Agenten a, b unterschiedliche Reproduktionsschwellen verwenden $\theta_a \neq \theta_b$. Damit lässt sich aus externer Sicht nicht beurteilen, ob für einen beobachteten Agent a gilt:

$$\frac{\theta_a}{2} \leq funds(a) \leq \theta_a$$

und somit ist ebenfalls keine Aussage über dauerhaft positiven oder negativen Profit möglich. Zudem ist zu einen bestimmten Zeitpunkt t die Informationsmenge zu gering, um gesicherte Rückschlüsse zu erlauben. Zwar erfolgt die Fitnessbewertung ebenso zu einem bestimmten Zeitpunkt in generationenbasierten Verfahren, jedoch sprechen einige Gründe im vorliegenden Modell gegen diese Art der Bewertung:

Abweichungen Erfolgt die Weitergabe der lokalen Fitness von a zu einem Zeitpunkt t mit $funds(a,t) < \theta$ und $funds(a,t') > 0 \quad |t \neq t'$, so ist eine Fehlinterpretation wahrscheinlich. Dasselbe gilt für den umgekehrten Fall mit $funds(a,t) > \theta, funds(a,t') < 0$.

Stagnationen Bleibt $\frac{\theta_a}{2} < funds(a) < \theta_a$ konstant, so ist hierbei die Fitness tatsächlich sehr gering, da durch $\pi = 0$ lediglich die Einnahmen die Ausgaben decken. Ein Vergleich mit einem profitablen Agenten b nach dessen Reproduktion in t_1 ergibt bis zum Zeitpunkt t_2 mit $funds(a,t_2) < funds(b,t_2)$ ein gegenteiliges Bild.

Zeitpunkt Die zufällige Wahl des 'richtigen' Zeitpunktes hat mehr Einfluss als die tatsächliche Höhe von $funds$ innerhalb der Grenzen $[\frac{\theta_a}{2}, \theta]$. Folgt $funds$ bei zwei Agenten dem Sagezahnmuster, so entscheidet der Zufall, welcher Agent bei t mehr Geld besitzt.

Zur lokalen Selektion ist die externe Einschätzung der Fitness eines Agenten notwendig, die invariant gegenüber kurzfristigen Schwankungen ist. Ein Maß für den Erfolg ist daher die Durchschnittsbildung über den Profit der letzten Δt Zeitpunkte:

$$extLocFitness_a = \sum_{t \in \Delta t} \pi_a(t) \tag{3.12}$$

Zusammenfassend lässt sich daher sagen, dass die lokale Fitness $funds$ ein sehr gutes und effizientes Maß für die interne Bewertung und Ableitung des Reproduktionszeitpunktes darstellt. Es stehen zur Bestimmung von θ unterschiedliche Methoden bereit (z.B. Empirie). Diese sind zum Teil äquivalent zur Festlegung anderer evolutionärer Parameter (z.B. Mutationsrate). Zur externen Bewertung im generellen Fall hingegen ist $funds$ nur bedingt geeignet. Unter der Annahme, dass θ für alle Agenten gleich ist, lässt die Auswertung zum Teil richtige Rückschlüsse zu. Der vorliegende Ansatz basiert jedoch auf einer kontextsensitiven Reproduktionsschwelle. Eine valide externe Interpretation ist damit nicht mehr gegeben. Zudem sagt $funds$ zu einem bestimmten Zeitpunkt nichts über den Verlauf aus, sondern fokussiert allein auf einen (unrepräsentativen) Snapshot. Es ist daher nicht empfehlenswert, die lokale Fitness

$funds$ gleichzeitig als Maß für eine externe Bewertung durch andere Agenten heranzuziehen. Zur Lösung wurde stattdessen die Durchschnittsbildung des Profites $extLocFitness_a$ über einen bestimmten Zeitraum in die Vergangenheit vorgeschlagen.

3.2.4 Lokale Selektion

Die lokale Selektion sucht nach geeigneten Partnern und macht deren Strategie s_p zur späteren Reproduktion verfügbar. Agenten, die über lokale Selektion direkt angesprochen werden, muss sichtbar für den Agenten sein (siehe Sichtbarkeit in Kapitel 3.1.1.3). Mit dem in Algorithmus 3 vorgestellten lokalen Selektionsverfahren wird innerhalb des Intervalls ΔLS eine sogenannte *Strategienachricht* an einen zufälligen und geeigneten Agenten $a_p \in A'$ verschickt (Zeile 10). Strategienachrichten enthalten die eigene Strategie s_a und eine im weiteren Verlauf definierte externe lokale Fitness ($extLocFitness_a$). Ein Agent, der eine Strategienachricht empfängt, antwortet ebenfalls mit einer Strategienachricht, die seine eigene Strategie und lokale Fitness enthält (nicht abgebildet). Beim Empfang einer Antwort wird die empfangene lokale Fitness des potentiellen Partners $extLocFitness_p{}'$ mit der aktuell gespeicherten lokalen Fitness $extLocFitness_p$ verglichen (Zeile 14) und ggf. die empfangene Strategie s_p gespeichert (Zeile 16). Daher existiert in diesem Zusammenhang der Begriff Generation nicht. Für jeden Agenten ergibt sich stattdessen eine Lebensspanne. Mit der abgebildeten Selektionsstrategie von Algorithmus 3 werden daher zwei Ziele verfolgt:

Verbreitung eigener Strategie wird realisiert, indem die eigene Strategie an bekannte Agenten verschickt wird (Zeile 7).

Suche geeigneter Strategien wird realisiert, indem die aktuell fitteste bekannte Strategie s_p mit der jeweiligen empfangenen Strategie $s_p{}'$ abgeglichen und gegebenenfalls gespeichert wird (Zeilen 9-14).

Algorithmus 3 Lokale Selektion

Vorbedingung: Strategie s_a

1: $extLocFitness_p := 0$
2: **loop**
3: /* Senden Strategienachricht */
4: $extLocFitness_a \leftarrow$ update externe lokale Fitness
5: $a_p \leftarrow$ wähle sichtbaren Agenten aus $A' \leftarrow vis(a)$ (siehe vis_a, Kapitel 3.1.1.3)
6: **if** $a_p = null$ **then**
7: warten ΔLS
8: start next loop
9: **end if**
10: sende Nachricht mit Strategie s_a und $extLocFitness_a$ an a_p
11: /* Empfangen Strategienachricht */
12: $message_p^s :=$ receive()
13: $extLocFitness_p{}' := message_p^s.get(extLocFitness)$
14: **if** $extLocFitness_p{}' > extLocFitness_p$ **then**
15: $extLocFitness_p := extLocFitness_p{}'$
16: $s_p := message_p^s.get(s_p)$
17: **end if**
18: warten ΔLS
19: **end loop**

Die beiden Ziele sind als Aspekte einer *aktiven* bzw. *passiven lokalen Selektion* aufzufassen, welche die proaktiven und reaktiven Eigenschaften der Agentendefinition 11 widerspiegeln. Die beiden sich ergänzenden Bestandteile Senden und Empfangen müssen nicht zwangsläufig synchron realisiert werden, sondern sind der Übersichtlichkeit halber zusammengefasst. Als Beispiel sei das Senden einer Nachricht an einen Agenten angeführt, der nicht erreichbar ist oder nicht mehr existiert. Falls das Nachrichtensystem keine negative Rückmeldung gibt, würde der Agent auf eine Antwort für immer warten. Für Implementierungsdetails sei auf Abschnitt 4.1 verwiesen. Für diese Art der externen lokalen Fitness wird gegenseitiges Vertrauen vorausgesetzt. Die sichtbaren Agenten ergeben sich aus der Umgebungsstruktur v_a und aus der Sichtbarkeitsfunktion $vis(a)$, assoziiert mit Agent a. Die Interaktionen richten sich damit nach der Netzwerktopologie, die als die 'geographische' Umgebung der Agentenpopulation angesehen werden kann.

Durch die Erzeugung von Kindagenten bzw. gegebenenfalls Split + Mutation ist die generelle Suchstrategie im Vergleich zu *steady-state* Verfahren mehr in Richtung Exploration verschoben. Die Population kann sich damit an die Nachfrage und an die in der Umgebung vorhandenen Ressourcen anpassen. Exploratives Verhalten tritt insbesondere dann auf, wenn die Umgebung mehr Ressourcen bereithält, als durch die Agentenpopulation 'verbraucht' bzw. benötigt wird. Startet ein System, bestehend aus einem Agenten, so erfolgt zunächst eine Phase hoher Exploration. Es werden mehr Agenten erzeugt, als sterben. Sobald die Population in etwa soviel Geld benötigt, wie durch die Umgebung bereitgestellt wird, verschiebt sich die Suchstrategie mehr in Richtung Exploitation, ohne jedoch den grundlegenden Explorationscharakter zu verlieren. Ein solches 'organisches Wachstum' ist gewünscht im Sinne der Nachfrageorientierung. Ebenso kann damit auf natürlicher Weise eine Adaptivität bezüglich der Umgebung erreicht werden.

Solange das Geld zwischen 0 und θ_a schwankt, erfolgt auf einen Agenten lokal kein Selektionsdruck. Die Weitergabe der Partnerstrategien, vorausgesetzt die Agenten sind nicht voneinander isoliert, ist ähnlich zur deterministischen Tournament Selektion [159]. Zu jedem Reproduktionszeitpunkt wird die bis dato beste empfangene Partnerstrategie s_p verwendet. Die Tournamentgröße k_a ist eine agentenspezifische emergente Variable. Sie ergibt sich Agenten-individuell und indirekt durch die Intervalle der Selektion ΔLS und Reproduktion ΔR und die sichtbaren Agenten $vis(a) \rightarrow A'$. Die obere Schranke für k_a eines Agenten a ist gegeben durch:

$$k_a = min\left(\left\lfloor \frac{\Delta R}{\Delta LS} \right\rfloor, |A'|\right) \tag{3.13}$$

k_a kann nicht größer sein als die Anzahl der durchgeführten Selektionen im Intervall ΔR und ist gleichzeitig begrenzt durch die Anzahl der für a sichtbaren Agenten $|A'|$.

Es besteht durchaus die Möglichkeit, gleichzeitig eine Strategienachricht an alle bekannten Agenten zu schicken und darüber eine Auswahl zu treffen. Dies würde jedoch das systemweite Nachrichtenaufkommen um den Faktor $|A'|$ pro Agent steigern. Zusätzlich ergibt sich dadurch jederzeit der maximale Selektionsdruck durch maximale Tournamentgröße k_a pro Agent bei deterministischer Auswahl des Partners. Alternativ bleibt hier eine stochastische Auswahl, die einen etwas höheren Aufwand verursacht. Daher wird für eine Umsetzung die in Algorithmus 3 dargestellte Selektion vorgeschlagen. Eine weitergehende Analyse des hierdurch entstehenden variablen Selektionsdruckes erfolgt in Abschnitt 3.3.3.

Damit erweitert die in dieser Arbeit verwendete lokale Selektion in wesentlichen Punkten vorhandene Ansätze bzw. vereinigt deren Eigenschaften:

Proaktivität und Lokalität der Selektion finden direkt im Agenten statt. In panmiktischen Verfahren, cEA und dEA übernimmt nicht der Agent die Selektion über Individuen in der lokalen Nachbarschaft, sondern der evolutionäre Algorithmus, welcher die Individuen verwaltet [18, 207].

Exploration und Exploitation wird gleichermaßen als Suchstrategie umgesetzt. Im Gegensatz zu Menczer in [156]

Adaptivität der Population wird durch eine ökonomische Populationsverwaltung erreicht. Elternagenten sterben nicht ab, sondern verbleiben in der Population. Damit wird eine Adaptivität der Populationsgröße im Gegensatz zum steady-state Verhalten [221] erreicht. Eine weitere Untersuchung erfolgt in Abschnitt 3.3.1.

Variable Reproduktionsschwelle erlaubt im Gegensatz zu Menczer [156] unterschiedliche Typen von Agenten bzw. eine Selbstadaption der Reproduktionsschwelle auf Agentenebene.

Insgesamt wird durch die vorgestellten Punkte eine wesentliche Erweiterung der bestehenden Verfahren in den Bereichen Lokalität, Suchstrategie (Exploration vs. Exploitation), Anpassung der Populationsgröße und Reproduktionszeitpunkt erreicht. Hierdurch werden wesentliche Aspekte der Adaptivität des vorgestellten Ansatzes bereits sichtbar. Eine mathematische Analyse der genannten Punkte erfolgt im nächsten Kapitel 3.3.

Ein weiteres theoretisches Konzept ist das Altern der gespeicherten Fitness s_p. Dieses Konzept wird in der weiteren Arbeit nicht betrachtet. Es stellt jedoch eine Möglichkeit der Weiterentwicklung dar und wird daher kurz vorgestellt.

Ist die Lebensspanne eines Agenten a sehr viel länger als ΔR, so erfolgt mehrmalige Reproduktion. Erhält a zu Beginn dieser Lebensspanne eine Strategie s_p mit extrem hoher externer Fitness, so kann diese Strategie möglicherweise die gesamte Lebensdauer als gespeicherte Partnerstrategie überdauern. Dies kann zu ungünstigen Selektionseffekten führen, die ähnliche Auswirkungen wie das Skalierungsproblem bei Fitness-basierter Selektion haben. Hierbei erfolgt eine ungewollte Minderung der Diversität der produzierten Nachkommen, was zu vorzeitiger Konvergenz führen kann. Eine Möglichkeit der Abhilfe besteht in der sogenannten *Alterung* der gespeicherten Fitness $extLocFitness_p$. Aus der Menge der denkbaren unterschiedlichen Methoden seien hier zwei kurz vorgestellt:

Alterung der externen lokalen Fitness $extLocFitness_p$ kann durch eine schrittweise Verringerung des in $extLocFitness_p$ gespeicherten Fitnesswertes erreicht werden . Nach einer vorgegebenen Zeit (z.B. ΔR) wird $extLocFitness_p$ um einen prozentualen oder absoluten Betrag $\Delta_{extLocFitness}$ verringert:

$extLocFitness_p \mathrel{-}= \Delta_{extLocFitness}$

Hierdurch wird die gespeicherte Fitness allmählich 'entwertet'. Erforderlich ist jedoch ein weiterer Parameter $\Delta_{extLocFitness}$, was aus administrativer Sicht neue Komplexität mit sich bringt.

Löschen der Partnerstrategie nach bestimmter Zeitspanne bzw. nach der Reproduktion stellt eine weitere Möglichkeit dar. Das Löschen von s_p und $extLocFitness_p$ nach erfolgter Reproduktion spart zusätzliche Parameter. Ist $\Delta R < \Delta LS$, so kann ein Rückfall auf einen einfachen Split ohne Rekombination erfolgen. Daher kann die Zeitspanne zum Löschen auch erweitert werden auf ein Vielfaches von ΔR bzw. das Löschen erst erfolgen, wenn eine neue Strategie vorhanden ist.

Vertrauen wird in dieser Arbeit bei der lokalen Selektion vorausgesetzt. In realen Anwendungen ist davon auszugehen, dass hierzu geeignete Maßnahmen zur Abhilfe erforderlich sind (z.B. Blacklists). Um die Effekte lokaler Selektion zu untersuchen und nicht durch vertrauensbildende sowie erhaltende Maßnahmen zu verzerren wurde auf eine diesbezügliche Untersuchung verzichtet.

3.3 Analyse der Eigenschaften

Das vorgestellte formale Modell sowie der Optimierungsansatz sind Bottom-Up Verfahren. Hierbei wurde der Ansatz aus Sicht der problemlösenden Agenten erstellt. Dieser Abschnitt beschäftigt sich mit der Analyse entstehender emergenter Effekte, die bei der Anwendung zu erwarten sind. In den nächsten Abschnitten erfolgt eine Ableitung von Systemeigenschaften aus dem gegebenen Modell.

3.3.1 Adaption der Populationsgröße

Systeme in ressourcenbegrenzter Umgebung tendieren zu einer Stabilisierung aufgrund ihrer *Tragfähigkeit* (engl. *carrying capacity*) [61, 149, 156]. Dieses Verhalten ist einer Vielzahl von Systemen zu eigen, insbesondere sind hier biologische (z.B. Fischpopulationen in Seen) und ökonomische Systeme (Angebot und Nachfrage) zu nennen. Im Kontext dieser Arbeit bestimmen monetäre Zu- und Abflüsse die Tragfähigkeit und damit auch die Populationsgröße.

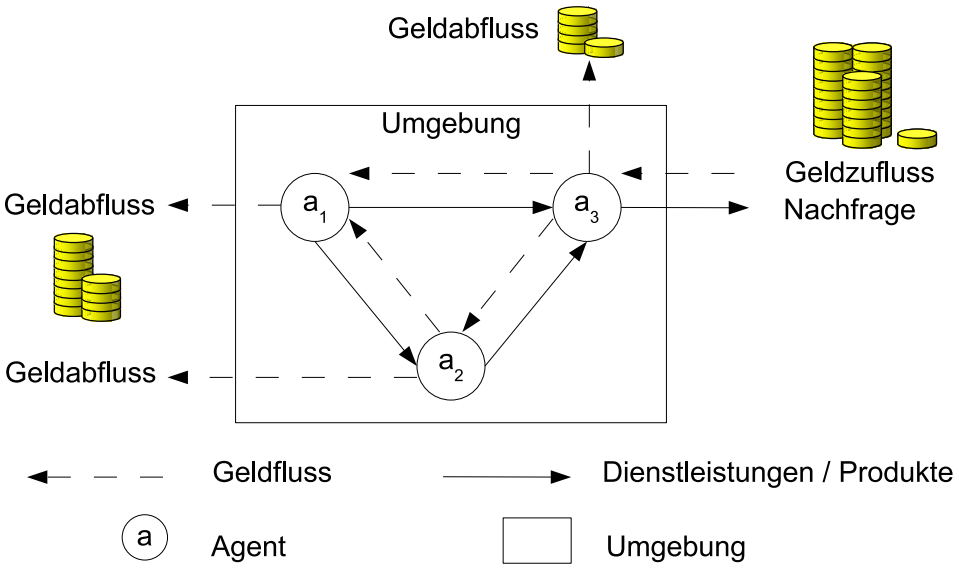


Abbildung 3.8: Geld- und Informations- bzw. Dienstleistungsflüsse zwischen evolutionären Agenten und Umgebung

Abbildung 3.8 zeigt schematisch eine makroökonomische Aufnahme eines Multiagentensystems und dessen Geld- und Warenflüsse. Zunächst stellt die Umgebung per Nachfrage $\mathcal{D}(t)$ eine bestimmte Menge Geld pro Zeiteinheit t zur Verfügung. Dieses Geld wird als Ausgleich für erbrachte Dienstleistungen über Zahlungen an eine (Teil)Menge der Agenten A transferiert. Dazu wird zunächst Menge aller eingehenden Zahlungen eines Multiagentensystems pro Zeiteinheit t betrachtet:

$$\mathcal{R}_A(t) = \sum_{a \in A} \mathcal{R}_{a(t)} \tag{3.14}$$

Hiervon entfällt eine Teilmenge auf die äußere Nachfrage $\mathcal{D}(t) \subseteq \mathcal{R}_A$, da sich die Agenten gegenseitig für Dienstleistungen bezahlen.

Zur Vereinfachung der Analyse wird eine konstante Nachfrage $\mathcal{D}$ und eine konstante Reproduktionsschwelle θ vorausgesetzt. Weiterhin und ohne Beschränkung der Allgemeinheit wird eine kontinuierliche Populationsgröße $|A|$ anstelle einer schrittweise angenommen. Für einen Agenten a stellt das Verhältnis $\frac{\pi_a(t)}{\theta}$ die anteilige Erzeugung eines neuen Agenten bzw. den Tod von a zum Zeitpunkt t dar. Ist $\pi_a(t) = \pm\theta$, so verlässt $funds(a,t)$ das Interval $[0,\theta]$ und bewirkt sofortigen Selektionsdruck (siehe dazu Abschnitte 2.3.5, 3.2.4) in Form der Erzeugung eines Agenten bzw. der Tod von a. Aufsummiert auf die gesamte Population ergibt sich analog der gesamten Zahlungseingänge aus Gleichung 3.14 der Gesamtprofit als Summe der Agentenprofite ($\pi_a(t)$, siehe Gleichung 3.11) innerhalb des MAS als

$$\Pi_A(t) = \sum_{a \in A} \pi_a(t) \tag{3.15}$$

In Abhängigkeit der Verteilung von $\Pi_A(t)$ in der Agentenpopulation werden neue Agenten erzeugt bzw. unprofitable Agenten sterben. Wird der Gesamtprofit durch θ geteilt, ergeben

sich statistisch die Höhe der Zu- und Abgänge innerhalb der Population und damit die Populationsgröße als logistische Gleichung:

$$|A(t+1)| = |A(t)| + \frac{\Pi_A(t)}{\theta} \tag{3.16}$$

Ist $\frac{\Pi_A(t)}{\theta} > 0$ so erhöht sich die Anzahl der Agenten, ist $\frac{\Pi_A(t)}{\theta} < 0$ so wid sie geringer. Daher strebt ein System aus evolutionären Agenten immer ins Gleichgewicht mit seiner Umgebung. Für konstante Umgebungsbedingungen ($\mathcal{D} = const$) lässt sich daher das folgende Theorem formulieren:

Theorem 1. *Die Populationsgrößenadaption führt dazu, dass der Durchschnittsprofit des Multiagentensystems MAS gegen null tendiert* $\Pi_0 = 0$.

Die Erklärung: solange systemweit positive Profite erzielt werden, führt dies zur Vergrößerung der Population. Eine größere Population führt zu höheren Steuern $\mathcal{T}$ und Zahlungen $\mathcal{P}$. Die Nachfrage $\mathcal{D}$ bleibt jedoch konstant und verringert damit zukünftige Profite und das Populationswachstum ebenso. Der umgekehrte Fall gilt analog. Der Beweis dazu findet sich in [184]. Langfristig pendelt damit die Agentenpopulation um eine bestimmte maximale Populationsgröße A_{max}.

Zur Quantifizierung der tatsächlichen Tragfähigkeit (=theoretische Maximalgröße) und damit zur Ermittlung der maximalen Populationsgröße gibt die Nachfrage $\mathcal{D}$ in Relation zu θ eine grobe Nährung:

$$|A_{max}| = \frac{\mathcal{D}}{\theta}$$

$|A_{max}|$ gilt jedoch nur, wenn hypothetisch die gesamte Nachfrage in Profit umgesetzt werden kann. Für eine bestehende Population, lassen sich deren aktuelle Daten verwenden, um eine genauere Abschätzung bei ansonsten konstanter Umgebung zu erhalten:

$$\lim_{t\to\infty} |A(t)| = \frac{\mathcal{D}(t) - \mathcal{T}_A(t) - \tilde{\mathcal{P}}_A(t)}{\theta} \tag{3.17}$$

Hierzu wird zunächst die Profitberechnung des MAS (siehe Gleichung 3.15) auf systemweite Geldzu- und -abflüsse beschränkt. Daher wird als Zufluss $\mathcal{R}_A(t)$ durch $\mathcal{D}$ ersetzt und die Abflüsse durch $\mathcal{T}_A(t) - \tilde{\mathcal{P}}_A(t)$ modelliert. $\mathcal{T}_A(t) = \sum_{a\in A} \mathcal{T}_a(t)$ bezeichnet die Steuerzahlungen von A zum Zeitpunkt t. Mit $\tilde{\mathcal{P}}_A(t)$ werden lediglich die Geldflüsse erfasst, welche das MAS verlassen, etwa als Zahlungen für externe Dienstleistungen. Für alle internen Zahlungen zwischen zwei Agenten a, b gilt: $\mathcal{P}_a(t) = \mathcal{R}_b(t)$. Diese können für eine systemweite Betrachtung an dieser Stelle ausgespart werden. Das Abschätzung 3.17 nur eine Nährung darstellt zeigt die Tatsache, dass jeder neue bzw. entfernte Agent die beiden Terme $\mathcal{T}_A(t)$ und $\tilde{\mathcal{P}}_A(t)$ geringfügig verändert.

Zusammenfassend lässt sich feststellen, dass eine Population aus evolutionären Agenten auf Nachfrageüberhänge und -veränderungen reagiert. Hierbei erfolgt eine emergente Angebotsanpassung als direkte Folge der Populationsgrößenänderung. Ohne zentrale Koordinierungsinstanz lässt sich damit eine intrinsische Populationsadaption realisieren. Die Adaption erfolgt immer vor dem Hintergrund der Kosten und damit verbundenen Steuern $\mathcal{T}$ der jeweiligen virtuellen Umgebungen, in denen die Agenten agieren. Ein solches Verhalten kann in realen System beobachtet werden, wenn Akteure in Märkte eintreten bzw. diese verlassen [97].

3.3.2 Adaption durch Verbreitung erfolgreicher Strategien

Dieser Abschnitt beschäftigt sich mit der Strategieverteilung innerhalb der Agentenpopulation A. Da keine globale Selektionsinstanz existiert, erfolgt der Vergleich der externen lokalen Fitness ausschließlich lokal (siehe Algorithmus 3 in Abschnitt 3.2.4). Ziel ist daher eine Quantifizierung, inwieweit lokales Verhalten zu Anpassung und damit zu Optimierung auf globaler Ebene führt.

Ebenso wie im vorangegangenen Abschnitt wird hier für die Analyse eine kontinuierliche Größe der Agentenpopulation $|A| \in \mathbb{R} \geq 0$ verwendet. Weiterhin wird eine unverfälschte Weitergabe der Strategie vom Elternagent auf den Kindagent $s_c := s_a$ ohne verzerrende Effekte wie Rekombination und Mutation angenommen. Während der Reproduktion (bzw. Split) wird im diskreten Fall eine Kopie von s_a erzeugt, wenn $funds(a) > \theta$. Im kontinuierlichen Fall hingegen wird eine Kopie mit der Wahrscheinlichkeit von $p = \frac{funds(a)}{\theta}$ erzeugt. Die Wahrscheinlichkeit erhöht sich, wenn $\pi_a > 0$. Für die Berechnung der Wahrscheinlichkeit einer Kopie von s_a werden nur Agenten mit positivem Profit $0 < \pi_a^+$ betrachtet, da negativer Profit $\pi_a^- < 0$ langfristig zur Löschung von s_a führt. Auf MAS-Ebene wird der positive Systemprofit zum Zeitpunkt t mit

$$\Pi_A^+(t) = \sum_{a \in A} \pi_a^+(t) \leq \Pi_A(t) \tag{3.18}$$

bezeichnet. Analog bezeichnet $\Pi_A^-(t)$ den gesamten negativen Systemprofit. In t werden daher statistisch

$$A^+(t) = \frac{\Pi_A^+(t)}{\theta} \quad \text{bzw.}$$
$$A^-(t) = \frac{\Pi_A^-(t)}{\theta}$$

$|A^+|$ neue Agenten erzeugt bzw. $|A^-|$ entfernt. Generiert ein Agent a positiven Profit, so beträgt der Anteil dessen Strategie s_a an den neu erzeugten Agenten:

$$\frac{\frac{\pi_a^+(t)}{\theta}}{|A^+(t)|} \tag{3.19}$$

Bezogen auf die Gesamtpopulation zum nächsten Zeitpunkt $(t+1)$ beträgt der Anteil von s_a:

$$p_{s_a}(t+1) = \frac{1 + \frac{\pi_a^+(t)}{\theta}}{|(A(t) \cup A^+(t)) \setminus A^-(t)|} \tag{3.20}$$

Dabei wird der Anteil der originalen Strategie s_a als 1 angenommen und der zusätzliche Anteil definiert sich über das Verhältnis Profit $\pi_a(t)$ zu θ. Die entstehende Population setzt sich zusammen aus der Menge der ursprünglichen Agenten A vereinigt mit der Menge neu hinzukommender Agenten $A^+(t)$ abzüglich der entfernten Agenten $A^-(t)$. Nach ausreichend langer Laufzeit des MAS kann Theorem 1 angewendet werden: Der Gesamtprofit $\Pi_A(t)$ geht gegen null und die Anzahl der Agenten bleibt nahezu unverändert, so dass gilt $|A(t)| = |A(t+1)|$ und die Veränderung des Strategieanteils aus Gleichung 3.20 kann vereinfacht werden zu:

$$p_{s_a}(t+1) = \frac{1 + \frac{\pi_a^+(t)}{\theta}}{|A(t)|} \tag{3.21}$$

Damit vergrößert/verringert sich der Anteil einer Strategie s_a proportional mit dem positivem/negativem Profit π_a des Agenten.

Grundlage eines besseren Profites ist die Strategie s_a, definiert in Kapitel 3.1.2.4. Durch andere Parametrisierung verhält sich die daraus gebildete Transferfunktion φ effizienter als bei anderen Agenten im Sinne der Nutzenmaximierung. Letztendlich benötigt ein im Vergleich profitablerer Agent nach Profitgleichung 3.11 weniger Zahlungen $\mathcal{P}_a$ an Dienstleister (andere Agenten, externe Services, ...), zahlt weniger Steuern $\mathcal{T}_a$ (aufgrund weniger Aktionen, geringerer Ressourcenverbrauch, ...) und bekommt höhere Zahlungen $\mathcal{R}_a$ für seine Dienstleistungen (z.B. aufgrund höherer Qualität der Dienstleistungen, bessere Verhandlungsstrategie, ...). All diese Faktoren führen in Summe zur Ausbreitung der erfolgreichen (=profitableren) Strategien und führen auf MAS Ebene zu einer Adaption bzw. Optimierung des Gesamtverhaltens. Dies äußert sich z.B. in geringerem Ressourcenbedarf des Gesamtsystems bei unveränderter Leistung oder gesteigerter Leistung bei gleichem Ressourcenbedarf. Auch hier erfolgt die emergente Adaption auf globaler Ebene aufgrund lokaler inhärenter Verhaltensmuster.

3.3.3 Adaptiver Selektionsdruck

Das in den vorangegangenen Abschnitten beschriebene Verhalten induziert selbstregulierenden Selektionsdruck in Bezug auf selektierte Strategien. In Systemen mit hohen Profiten pro Agent besteht noch genügend Spielraum zur Exploration bzw. Expansion der Population. Hohe Profite führen zu geringen Reproduktionsintervallen auf Agentenebene, da $funds(a)$ schnell $\theta(a)$ erreicht. Ist daher $\Delta R \leq \Delta LS$, so findet maximal eine Selektion vor jeder Reproduktion statt. Die Tournamentgröße k ist daher 0 oder 1 und die Selektion ist eine zufällige Selektion bzw. eine *Selbstselektion* und unterstützt den explorativen Charakter der Suche.

In angepassten Systemen befinden sich die Kosten und das von der Umgebung zur Verfügung gestellte Geld etwa im Gleichgewicht. Das bedeutet für die Profite der Agenten eine langfristige Stabilisierung um $\pi_a \approx 0$. Der Reproduktionszeitraum ΔR_a kann hierbei theoretisch sehr lange dauern. Während dieses Zeitraumes führt jeder Agent pro Zeitintervall ΔLS eine Selektion durch und vergrößert damit die Tournamentgröße k der evaluierten Agenten bis zur maximalen Anzahl bekannter Agenten $|A'|$. Die Wahrscheinlichkeit, dass ein Agent $a' \in |A'|$ bei n-maliger zufälliger Selektion nicht ausgewählt (in den Tournamentpool) wird, beträgt:

$$P(E) = \left(\frac{|A'| - 1}{|A'|}\right)^n$$

Entsprechend ist das Gegenereignis 'bei n-maliger Selektion wird a' mindestens einmal selektiert:

$$P(\overline{E}) = 1 - P(E) = 1 - \left(\frac{|A'| - 1}{|A'|}\right)^n$$

Wird dieses Ereignis mit der gesamten für a sichtbare Agentenpopulation erweitert, so ergibt sich die Wahrscheinlichkeit des Ereignises 'bei n-maliger Selektion wird jeder Agent in A' mindestens einmal selektiert' durch

$$P(A') = \left(1 - \left(\frac{|A'|-1}{|A'|}\right)^{n}\right)^{|A'|}$$

Durch Ersetzung des n-maligen Widerholens durch die entsprechenden Intervalle ergibt sich:

$$P(A') = \left(1 - \left(\frac{|A'|-1}{|A'|}\right)^{\lfloor \frac{\Delta R}{\Delta LS} \rfloor}\right)^{|A'|} \tag{3.22}$$

Wird jeder Agent in A' mindestens einmal vor der Reproduktion von a' selektiert, so bedeutet dies den maximalen Selektionsdruck in a's lokaler Nachbarschaft. Hierbei ist $k_a = |A'|$ und es wird deterministisch die Strategie des besten gefundenen Agenten zur Reproduktion genommen.

Beispiel 4 (Adaptiver Selektionsdruck)
Sei a' ein Agent mit $|A'| = 5$ sichtbaren Agenten in seiner lokalen Nachbarschaft. Damit ergibt sich nach 18 zufälligen Selektionen ($\lfloor \frac{\Delta R}{\Delta LS} \rfloor = 18$) eine Wahrscheinlichkeit von $P(A') = 0.91$, dass $k_a = |A'|$. Nach 21 Selektionen beträgt die Wahrscheinlichkeit $P(A') = 0.95$ und nach 28 Selektionen $P(A') = 0.99$.

Alternativ existiert in jeder lokalen Nachbarschaft die Möglichkeit, z.B. über das *Contract Net Protokoll* (siehe Abschnitt 2.2.5) eine Ausschreibung zu starten. Diese Möglichkeit birgt jedoch im Vergleich eine Reihe von Nachteilen. Zunächst vervielfacht sich die Anzahl der versendeten Nachrichten um den Faktor $|A'|$, da immer an alle Agenten der Nachbarschaft gesendet wird. Weiterhin müssen noch Antworten berücksichtigt werden. Bei einer angenommenen Rücksendequote von $\frac{|A'|}{2}$ beträgt das Verhältnis initial $1 : \frac{|A'|}{2} + |A'|$. Je nach Lebensdauer eines Agenten und Veränderungen in seiner Nachbarschaft A' kann sich das Verhältnis ändern. Schließlich kann angeführt werden, dass bei Reproduktion von a immer mindestens ein erfolgreicher Agent beteiligt ist. Dies ist a selbst, da offensichtlich in der Vergangenheit genügend Profite erwirtschaftet wurden. Es wäre daher zur Vermeidung vorzeitiger Konvergenz kontraproduktiv, immer die besten Individuen auszuwählen.

Damit vereint obiger einfacher lokaler Selektionsmechanismus eine effiziente Umsetzung mit adaptivem Selektionsdruck. Zusätzlich ermöglicht dieses Verhalten ein implizites und damit selbst-adaptives Umschalten von Exploration auf Exploitation. Dies erfolgt implizit bei Erreichung der Tragfähigkeit und bedarf keiner zusätzlichen Kontrolllogik bzw. Überwachung. Damit existiert ein effizienter Mechanismus zur Steuerung der Suchstrategie.

3.3.4 Adaption Strategie und Transferfunktion

Nach dem Verständnis dieser Arbeit passt sich ein System aus evolutionären Agenten an unterschiedliche Aufgaben in veränderlichen Umgebungen an. Die auf diese Weise erzielte Adaptivität stellt den generellen Anpassungsmechanismus auf Systemebene dar. Hierbei wird ausgehend von lokalen Mustern eine globale Systemveränderung durch Einsatz selbstorganisierender evolutionärer Prinzipien erzielt. Beim Entwurf konkreter Agenten existiert bislang

das abstrakte Konzept der Strategieadaption 3.3.2. Dieser Abschnitt beleuchtet verschiedene technische Perspektiven zur Formalisierung generischer adaptiver Verhaltensmuster auf Agentenebene.

Zur Beschreibung der Verhaltensadaptivität ist es zunächst notwendig, das Verhalten eines Agenten φ (siehe Kapitel 3.1.2.4) im Sinne von Berechenbarkeit aufzufassen. Die Transferfunktion $\varphi := (sen, z) \mapsto (akt, z')$ ist eine Funktion, die von n Dimensionen auf m Dimensionen abbildet:

$$\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

wobei $n = |sen|+|z|$ und $m = |akt|+|z'|$. Damit bietet sich aus der Menge äquivalenter Berechenbarkeitsmodelle[1] das neuronale Netzwerkmodell aufgrund der leichten Überführbarkeit von φ auf neuronalen Netzwerke an.

Zur Herleitung prinzipieller Adaptivität dieses Modells muss zunächst die Rolle der Strategie s_a genauer spezifiziert werden. Formal wird φ durch eine geeignete Abbildung $s_a \rightarrow \varphi$ kodiert. In der Literatur existieren eine Vielzahl an Arbeiten zur Optimierung neuronaler Netzwerke mittels evolutionärer Verfahren [117, 202]. Für eine praktische Anwendung ist es jedoch nicht immer sinnvoll, die gesamte Funktionalität von φ durch s_a zu kodieren. Zur Veranschaulichung sei ein Agent mit einer Vielzahl von Funktionalitäten ausgestattet, wovon Teile z.B. zum Nachrichtenhandling benötigt werden. Zumindest diese Teile verbleiben konstant, um Basisfunktionalitäten sicherzustellen. Es ist daher im Sinne der Gesamtfunktionalität sorgfältig abzuwägen, welche Funktionalität adaptiv, d.h. durch Kodierung mittels s_a, und welche Funktionalität statisch zu halten ist.

Beispiel 5
Die Fallunterscheidung in der Logik eines Lagerbestellagenten resultiert vereinfachend in zwei Zuständen: Bestellung / keine Bestellung. Hierbei ist sinnvollerweise für eine Optimierung auf Agentenebene die Wahl des Mindestbestandes ausschlaggebend, d.h. der minimale Lagerbestand, ab welchem eine Bestellung ausgelöst werden kann.

Für Beispiel 5 kann die Transferfunktion bis auf die Anpassung des Mindestbestandes statisch gehalten werden. Der Mindestbestand wird als Schwellenwert in einem Neuron repräsentiert, welches den aktuellen Lagerbestand als Input bekommt und bei Unterschreitung des Schwellenwertes ein Aktionspotential auslöst. Damit ist im obigen Beispiel die Adaptivität der Agentenstrategie auf den fest eingegrenzten Rahmen der Bestellung beschränkt. Flexibler kann z.B. eine Bestellstrategie gehalten werden, die zusätzliche Parameter (z.B. Nachfrage, Saison, Feiertage, etc.) in einem komplexeren Entscheidungsalgorithmus kombiniert. Eine generelle Adaptivität im Sinne von langsam wechselnden Aufgaben lässt sich jedoch auch damit nicht erreichen. Um eine auf neuronalen Netzen basierende Transferfunktion vollständig adaptiv zu gestalten, sind alle Parameter zur Beschreibung des Netzes variabel zu halten.

Prinzipiell ergeben sich daraus unterschiedliche Stufen der Adaptivität auf der Ebene der Transferfunktion:

[1] z.B. μ-Rekursion [178, 213], Turing-Berechenbarkeit [213], Neuronale Netzwerke [219], Zelluläre Automaten [253]

Parameter Einzelne Werte werden per Strategie s_a optimiert und in der ansonsten statischen Transferfunktion φ übernommen. Die Funktionalität von φ ist für den speziellen Einsatzzweck vorgesehen und in engen Grenzen anpassbar auf veränderte Szenarien. Im Rahmen dieser Arbeit wird der Einsatz von Parametern zur Optimierung der Agentenstrategie eingesetzt.

Verhaltensmuster Komplexere Verhaltensmuster oder Algorithmen wie z.B. Verhandlungsprotokolle oder Wegsuche werden per s_a kodiert. Hierbei muss oft eine variable Anzahl genetischer Informationen kodiert werden, da a priori die optimale zur Aufgabe korrespondierende Dimension des Netzes nicht bekannt ist.

Vollständige Funktionalität Alle vom Agenten zu erfüllenden Aufgaben werden mit Hilfe eines neuronalen Netzwerkes realisiert. Hierbei werden sämtliche Kenngrößen des neuronalen Netzwerkes in der Strategie kodiert. Dazu zählen z.B. die Anzahl der Neuronen (Input, Output, Hidden), die Anzahl und Gewichtungen der Verbindungen, die Schwellwerte, das Mapping der Sensoren/Aktoren Sen_a/Akt_a auf die Input- / Outputneuronen, etc. Hierbei ist die offene Realisierung von φ die Grundlage für Offenheit bezüglich resultierender Agentenfunktion. Agenten erfüllen damit nicht eine per Design vorgegebene Aufgabe, sondern entwickeln ihr Problemlösungspotential *koevolutorisch* innerhalb der Population und in ihrer spezifischen Umgebung. Eine völlig offene, adaptive und selbst-optimierende Funktionalität sowohl auf Agentenebene und auf MAS-Ebene ist die Folge. Die Richtung der Optimierung wird gleichsam durch die Umgebung und deren Nachfrage bzw. Steuern vorgegeben (siehe hierzu Abschnitte 3.3.1, 3.3.2).

Für adaptives Verhalten in vorgegebenem Rahmen bieten die Ansätze der parametrisierten bzw. verhaltensbasierten Anpassung gute Möglichkeiten. Hier kann aufgrund gut verstandener und funktionierender Basismöglichkeiten eine weitere Ebene der Adaptivität und daraus resultierender Optimierung auf Systemebene realisiert werden.

Eine Realisierung des vollständig offenen Ansatzes geht jedoch weit über die Intention dieser Arbeit hinaus und bietet enorme Möglichkeiten für weitere Forschungsarbeiten. Die Aspekte der Konvergenz bzw. der Initialisierung spielen für erfolgreiche Anwendungen eine große Rolle. Hinsichtlich der Resistenz gegenüber kleinen Änderungen grundlegender Funktionen, z.B. beim Nachrichtenversand, sind bei direkter Umsetzung eher Fragilität und Brüchigkeit des Gesamtsystems zu erwarten. Damit fällt diese Methode in den Bereich der Grundlagenforschung die realistische Anwendungen nicht in nächster Zukunft erwarten lassen.

3.3.5 Erweiterte Takeover time

Die Ausbreitung von Informationen und der hierdurch induzierte Selektionsdruck werden maßgeblich durch die Netzwerkstruktur und der lokalen Nachbarschaft eines Knotens bestimmt [175, 207]. Dieses Kapitel greift die in Abschnitt 2.3.6 identifizierten Fragestellungen nach der Takeover time schwach vernetzten und dynamischen Netzwerkstrukturen auf. Ziel ist die Untersuchung der Ausbreitung des besten genetischen Materials in statischen und dynamischen schwach vernetzten Graphenstrukturen. In einem ersten Schritt wird in Abschnitt 3.3.5.1 empirisch der Grad der Vernetzung bestimmt. Danach wird die Takeover time in schwach vernetzten statischen Strukturen beschrieben und abschließend in dynamischen Netzwerkstrukturen. Das Update der Individuen erfolgt synchron, d.h. alle Individuen werden gleichzeitig aktualisiert.

3.3.5.1 Zufällige Graphen

Stuart Kauffman beschreibt in [124] das sogenannte *Threads Buttons* Phänomen. Buttons steht hierbei für Knöpfe bzw. Knoten, die durch Kanten bzw. Fäden, sogenannte Threads, verknüpft sind. Wird wiederholt jeweils zwischen zwei zufällig ausgewählten Knoten eine ungerichtete Kante eingefügt, dann beginnen die Knoten mehr und mehr zu verbundenen (Teil)graphen $G_{c_1}, \ldots, G_{c_r}$ zusammenzuwachsen. Dabei gilt:

$$G = \bigcup_{i=1}^{r} G_{c_i}$$

mit $G_{c_i}(V_{c_i}, E_{c_i})$. Abbildung 3.9 zeigt für Graphen $G(V, E)$ verschiedener Größe den Zusammenhang zwischen Anzahl von Kanten zu Knoten $\frac{|E|}{|V|}$ und des jeweils *größten verbundenen Subgraphen* $G_c^{max}(V_c^{max}, E_c^{max})$ $|V_c^{max} \subseteq V, E_c^{max} \subseteq E$ (siehe hierzu Definition 22 im Anhang). Die Simulationen wurden für verschiedene Graphgrößen durchgeführt und jeweils über 50 Iterationen gemittelt, um statistisch valide Ergebnisse zu erzielen. Für jede Graphgröße mit $n = |V|$ Knoten wurde in $2n$ Schritten je eine Kante eingefügt und in jedem Schritt G_c^{max} berechnet.

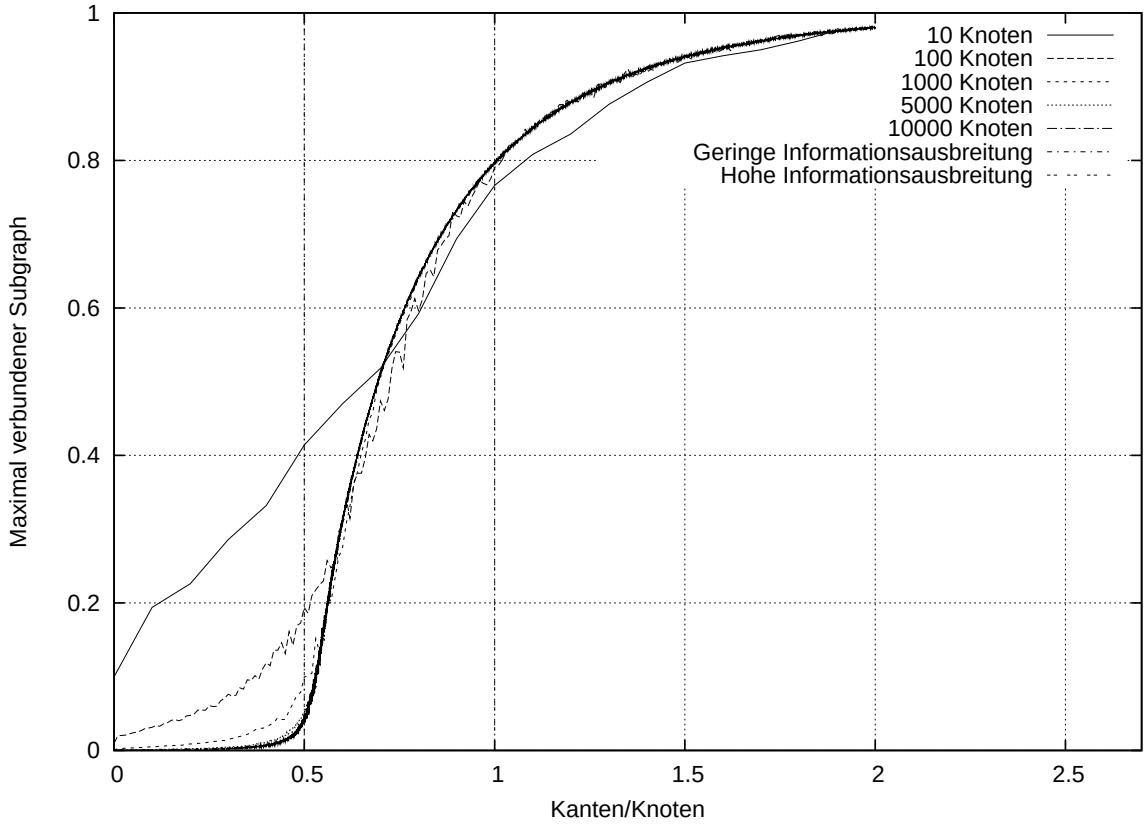


Abbildung 3.9: Verschiedene Graphgrößen im Vergleich. Zusammenhang zwischen dem Verhältnis Kanten zu Knoten und des jeweils größten verbundenen Subgraphen

Der Zusammenhang größter verbundener Subgraph zum Kanten/Knoten Verhältnis wird durch die Funktion

$$tb^{|V|}\left(\frac{|E|}{|V|}\right) = \frac{|V_c^{max}|}{|V|} \tag{3.23}$$

beschrieben. $\frac{|E|}{|V|}$ bezeichnet das Kanten zu Knoten Verhältnis und $|V|$ gibt die Anzahl von Knoten des Graphen an. tb gibt bei gegebener Knotenanzahl und $\frac{|E|}{|V|}$ die Größe des maximal verbundenen Subgraphen $\frac{|V_c^{max}|}{|V|}$ zurück. Wie Abbildung 3.9 zeigt, nähern sich für steigende $|V|$ die Funktionswerte von $tb^{|V|}$ an. Daher wird für hinreichend große Anzahl von Knoten der Einfachheit halber die allgemeine Form ohne Angabe der Graphgröße verwendet:

$$tb\left(\frac{|E|}{|V|}\right) \approx \frac{|V_c^{max}|}{|V|} \tag{3.24}$$

Einfügen von Kanten bis zu einem Verhältnis von $\frac{|E|}{|V|} < 0.5$ bewirkt fast keine Zunahme von tb. Diese Beobachtung ist nachvollziehbar, denn bei $\frac{|V|}{2}$ Kanten gibt es genau $|V|$ Kantenenden. Bei idealer Verteilung der Kanten ist jeder Knoten über genau eine Kante mit einem weiteren Knoten verbunden. Für Graphen mit $\frac{|E|}{|V|} < 0.5$ liegt die Größe des maximal verbundenen Subgraphen bei weniger als 20% für $|V| > 100$ (siehe auch Tabelle 3.1). Die Wahrscheinlichkeit ist gering, dass ein Knoten mit mehreren anderen Knoten verbunden ist. Bemerkenswert ist der Phasenübergang ab einem Verhältnis von Kanten zu Knoten von $\frac{|E|}{|V|} > 0.5$. Dieses Verhältnis wird von Hofstad ([109] S.115) als *kritisches Verhalten* (engl. *critical behavior*) beschrieben. Eine Abschätzung für die Größe des größten Subgraphen wird nach Hofstadt mit $|V|^{\frac{2}{3}}$ angegeben, was ein gute Nährung für die Graphgröße von $|V| = 1000$ Knoten darstellt. Im Vergleich zu Hofstad und den in [124, 125] gezeigten Zusammenhängen wird hier für unterschiedliche Graphgrößen die Entwicklung des größten verbundenen Subgraphen aufgezeigt. Je größer der untersuchte Graph, umso steiler wird der *Phasenübergang* (engl. *phase transition*) ab $\frac{|E|}{|V|} = 0.5$. Die Größe von $|V_c|$ beginnt stark zu steigen. Erdős und Rényi [63] bewiesen für diese Phasentransition die Entstehung eines riesigen verbundenen Subgraphen. Bei idealer zufälliger Verteilung sind in einem Graph mit n Knoten für $\frac{|E|}{|V|} = 0.5$ bereits je zwei Knoten verbunden. Es existieren dann $\frac{|V|}{2}$ Paare verbundener Knoten bzw. Subgraphen. Es gibt maximal $\frac{|V|(|V|+1)}{2} = \binom{|V|}{2}$ Kanten in einem ungerichteten Graphen in welchem *Schlingen* (siehe Anhang D) erlaubt sind. Für jedes bereits verbundene Knotenpaar $\{A, B\}$ gibt es drei Kanten $A-B, A-A, B-B$, deren Einfügen keine Änderung bewirkt. Daher verändern $\frac{3|V|}{2}$ Kanten die Verbindungen des Graphen nicht. Alle restlichen Kanten jedoch verbinden beim Einfügen bis dato unverknüpfte Subgraphen miteinander. Die Wahrscheinlichkeit dafür beträgt:

$$\begin{aligned} p &= 1 - \frac{\frac{3|V|}{2}}{\frac{|V|(|V|+1)}{2}} \\ &= 1 - \frac{3}{|V|+1} \text{ für } |V| \geq 2 \end{aligned} \tag{3.25}$$

Die Wahrscheinlichkeit p, dass die $\frac{|V|}{2} + 1$ Kante zwei bislang unverbundene Subgraphen verbindet, geht für steigende $|V|$ gegen 1:

$$\lim_{|V|\to\infty} p = 1 - \frac{3}{|V|+1} = 1 \tag{3.26}$$

Hierdurch lässt sich der steilere und abrupte Anstieg für größere Graphen erklären. Bemerkenswert ist der nahezu identische Anteil des größten Subgraphen am Gesamtgraph ab $\frac{|E|}{|V|} > 0.6$ (siehe Tabelle 3.1).

	$tb\left(\frac{\lvert E\rvert}{\lvert V\rvert}\right)$ in Relation von Graphgröße		
$\frac{\lvert E\rvert}{\lvert V\rvert}$	100	1000	10000
0.1	0.0314	0.0050	0.0007
0.2	0.0474	0.0089	0.0012
0.3	0.0762	0.0150	0.0026
0.4	0.1190	0.0313	0.0064
0.5	0.1939	0.0997	0.0395
0.6	0.3086	0.2742	0.3139
0.7	0.4740	0.5051	0.5110
0.8	0.5962	0.6385	0.6421
0.9	0.7246	0.7316	0.7315
1.0	0.7906	0.8001	0.7956
1.1	0.8480	0.8423	0.8440
1.2	0.8778	0.8770	0.8785
1.3	0.9074	0.9035	0.9050
1.4	0.9192	0.9254	0.9248
1.5	0.9398	0.9395	0.9406
1.6	0.9506	0.9522	0.9522
1.7	0.9616	0.9622	0.9625
1.8	0.9702	0.9703	0.9692
1.9	0.9786	0.9736	0.9756
2.0	0.9798	0.9807	0.9800

Tabelle 3.1: Größe des maximal verbundenen Subgraphen für unterschiedliche Graphgrößen und ausgewählte Knoten zu Kanten Verhältnissen

Für Graphen mit vielen Kanten und einem Verhältnis von $\frac{|E|}{|V|} \gg 1$ gilt:

$$\lim_{|E|\to\infty} tb\left(\frac{|E|}{|V|}\right) = |V|, \tag{3.27}$$

was Hofstad (vgl. [109] S.115) als *superkritische Ordnung* bezeichnet (engl. *supercritical regime*).

In Abhängigkeit des Kanten zu Knoten Verhältnisses für Graphen mit ($|V| > 100$) lässt sich die Größe des größten verbundenen Subgraphen in drei Stufen einteilen (siehe auch Tabelle 3.1):

Gering: Für $\frac{|E|}{|V|} \leq 0.5$ findet fast keine Vernetzung statt. Für Graphen mit mehr als 100 Knoten beträgt die Größe des größten verbundenen Subgraphen im Mittel weniger als 20% des Graphen.

Mittel: Für $0.5 < \frac{|E|}{|V|} < 1.1$ werden größere Teile des Graphen in Abhängigkeit von tb vernetzt. Der Vernetzungsgrad variiert durchschnittlich zwischen 20% und 80% der Graphgröße.

Groß: Ab $\frac{|E|}{|V|} \geq 1.1$ gehören im Schnitt mindestens 80% des Graphen zum größten verbundenen Subgraphen.

Giacobine et al. [84] geht auch bei schwach schwach vernetzten Strukturen davon aus, dass der gesamte Graph übernommen wird. Diese Annahme erweist sich nach den Ergebnissen aus Tabelle 3.1 als nicht haltbar. Es ist daher anzunehmen, das in Populationen mit schwacher Vernetzung zwischen den Individuen keine vollständige Informationsausbreitung stattfinden kann. Übertragen auf evolutionäre Ansätze bedeutet das eine unvollständige Ausbreitung des fittesten genetischen Erbgutes innerhalb der Population. Es erscheint daher eine Erweiterung der Takeover time im Hinblick auf die Größe übernommener Teilstrukturen sinnvoll, um die Ausbreitung quantitativ abzuschätzen.

Daher werden in den nachfolgenden zwei Abschnitten schwach vernetzte statische bzw. dynamische Graphenstrukturen mit Bezug auf den durch die Takeover time induzierten Selektionsdruck untersucht. Anschließend werden dynamische Strukturen auf ihren Selektionsdruck hin evaluiert.

3.3.5.2 Schwach vernetzte, statische und zufällige Graphen

Für evolutionäre Agenten, deren initiale Umgebung maßgeblich aus maximal einer Verbindung zum Elternagent besteht, ist gerade eine Untersuchung für $tb \leq 1$ notwendig. Anhand Abbildung 3.9 wird deutlich, wie sich Gene des besten Agenten innerhalb einer evolutionären Agentenpopulation in statischen Graphen ausbreiten. Aufgrund der Charakteristika des Threads-Buttons Phänomen ist es sehr wahrscheinlich, dass nur Teile eines derartigen statischen Graphen vom besten Agenten übernommen werden können. Für eine Charakterisierung von Teilübernahmen ist die Takeover time allein nicht aussagekräftig genug, wie Abbildung 3.10 zeigt.

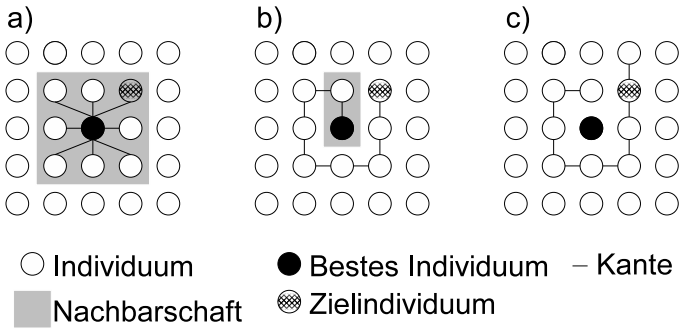


Abbildung 3.10: Auswirkungen unterschiedlicher Graphenstrukturen auf die Takeover time

Die drei Abbildungen in 3.10 zeigen unterschiedliche Graphenstrukturen und verdeutlichen die Auswirkungen auf die klassische Betrachtungsweise der Takeover time. Die Eigenschaften der jeweiligen Graphen sind formal betrachtet gleich: Anzahl der Knoten beträgt $|V| = 25$, Anzahl der Kanten beträgt $|E| = 8$, es existiert jeweils genau ein bester Agent, die Größe des maximal verbundenen Teilgraphen beträgt jeweils $|V_c| = 9$.

Zunächst ist zu bemerken, dass die Takeover time für alle Graphen unendlich ist, da die Graphen jeweils nicht verbunden sind. Trotz gleicher Graphcharakteristika ist eine Aussage über die klassische Takeover time nicht möglich. Wird die Übernahme des erreichbaren Subgraphen betrachtet, so ergeben sich unterschiedliche Werte für die benötigte Anzahl von Schritten und die übernommenen Individuen gemäß Tabelle 3.2. In Abb. 3.10a ist die Übernahme nach einem Schritt abgeschlossen, während dies in Abb. 3.10b 8 Schritte dauert und in Abb. 3.10c keine Übernahme anderer Individuen stattfindet.

	Abb. 3.10a	Abb. 3.10b	Abb. 3.10c
Takeover time (siehe Def 2)	∞	∞	∞
Zeit (Schritte), bis maximale Übernahme erreicht	1	8	0
Erreichter Subgraph (Größe)	8	8	1
Zielindividuum (schraffiert) übernommen	ja	ja	nein

Tabelle 3.2: Charakteristika unterschiedlicher Graphenstrukturen

Es ist daher sinnvoll, die Takeover time für schwach vernetzte Graphen um weitere Aspekte zu ergänzen. Für die *erweiterte Takeover time* (ETT) werden drei Parameter definiert, die detaillierteren Aufschluss über die teilweise Übernahme einer schwach vernetzten Graphenstruktur geben:

Zeit $ETT(T)$: Ist in Analogie zur Takeover time in Populationsnetzwerken (siehe Abschnitt 2.3.6) als Erwartungswert der Zeit definiert. Der Unterschied besteht darin, dass in schwach vernetzten Graphen lediglich ein Teilgraph übernommen werden kann. Daher wird nicht die Zeit für die Übernahme der Gesamtpopulation gemessen, sondern bis ein Stillstand in der Anzahl der fitteren Individuen eintritt. Andernfalls beträgt die Zeit für Knoten, die nicht mit dem besten Individuum über einen Pfad verbunden sind ∞.

Übernommener Subgraph $ETT(G_c)$: Bezeichnet den Erwartungswert für die Größe des übernommenen Subgraphen V_c: $|V_c|, V_c \in G_c, V \in G, G_c \subseteq G$ an.

Wahrscheinlichkeit $ETT(p)$: Gibt den Erwartungswert für Übernahme eines zufällig gewählten Individuums an.

Wesentliche Einflussfaktoren auf die Größe des übernommenen Subgraphen sind die initiale Positionierung des besten Individuums und die Größenverteilung der Subgraphen G_{ci} innerhalb G $|1 \leq i \leq r$. Für die Größenverteilung der Subgraphen in V gilt:

$$P_c(V_{c_i} \in V_c) = \sum_{i \in 1}^{r} \frac{|V_{c_i}|}{|V|} = 1 \tag{3.28}$$

Wobei $|V_{c_i}|, V_{ci} \in G_{c_i}$ die Größe des jeweiligen Subgraphen bezeichnet und P_c die Größe der Subgraphen $G_{c_i} \in G_c$ als Anteil an V zuordnet.

Die Wahrscheinlichkeit, dass das beste Individuum in Subgraph i platziert wird, sei gegeben durch

$$p^{best}_{G_{c_i}} = \frac{|V_{c_i}|}{|V|} \tag{3.29}$$

Ohne Beachtung der Zeitkomponente für $ETT(G_c)$ übernimmt das beste Individuum den kompletten Subgraph G_c, in welchem es platziert wurde, nach endlich vielen Schritten. Die Erwartungswert der Größe von G_c ist die Kombination aus der Größenverteilung der Subgraphen (Definition 3.28) und der Wahrscheinlichkeit, dass das beste Individuum in jedem dieser Subgraphen platziert wird (Definition 3.29).

$$\begin{aligned} ETT(G_c) &= |V_{c_1}| \cdot \frac{|V_{c_1}|}{|V|} + |V_{c_2}| \cdot \frac{|V_{c_2}|}{|V|} + \ldots + |V_{c_r}| \cdot \frac{|V_{c_r}|}{|V|} \\ &= \frac{1}{|V|} \cdot (|V_{c_1}|^2 + |V_{c_2}|^2 + \ldots + |V_{c_r}|^2) \end{aligned} \tag{3.30}$$

Eine detaillierte Abschätzung der Größenverteilung der Subgraphen ist mathematisch schlecht zu beschreiben [84], da genaue Kenntnisse über die Struktur und deren Wahrscheinlichkeit von G notwendig sind. Für $\frac{|E|}{|V|} > 1$ beweist Hofstad [109] und Erdős und Rényi [63], dass der Gesamtgraph statistisch aus einem großen Subgraph der Größe $|V_c^{max}| = |V|$ und einigen kleinen Subgraphen der Größe $|V_{c_i}| = log(|V|), i = 1, \ldots, r-1$ besteht. Mit Hilfe dieser Eigenschaft ist $|V_c^{max}| \gg |V_{c_i}|, V_c^{max} \neq V_{c_i}$ lässt sich die Abschätzung von Definition 3.30 durch den größten Subgraph G_c^{max} approximieren, da die restlichen Subgraphen keinen wesentlichen Einfluss auf das Ergebnis darstellen (siehe 3.32). Durch die Umformung (3.33) lässt sich dann in 3.34 der maximale Subgraph mittels der bekannten Funktion tb substituieren. Hierbei wurde eine unbekannte Graphenstruktur auf ein bekanntes Merkmal zufälliger Graphen zurückgeführt und als Berechnungsgrundlage genutzt.

$$ETT(G_c) = \frac{1}{|V|} \cdot (|V_{c1}|^2 + |V_{c2}|^2 + \ldots + |V_{cr}|^2) \qquad \text{(siehe Def. 3.30)} \tag{3.31}$$

$$\approx \frac{1}{|V|} \cdot |V_{c1}^{max}|^2 \qquad \text{(Approximation)} \tag{3.32}$$

$$\approx |V| \cdot \left(\frac{|V_c^{max}|}{|V|} \right)^2 \qquad \text{(Umformung)} \tag{3.33}$$

$$\approx |V| \cdot \left(tb \left(\frac{|E|}{|V|} \right) \right)^2 \qquad \text{(siehe Def. 3.24)} \tag{3.34}$$

Abschätzung 3.34 liefert erstaunlich genaue Resultate, wie Abbildung 3.11 zeigt. Hier werden die Modellabschätzungen 3.34 mit tatsächlichen Simulationen für $100, 1000, 10000$ Knoten in Bezug auf die Anzahl übernommener Individuen verglichen. Zur besseren Vergleichbarkeit wurden nicht die absolute Anzahl der Individuen dargestellt, sondern deren Anteil an der Gesamtpopulation. Für die Substitution wurden die Resultate einer Simulation mit 10000 Knoten für $tb = tb^{10000}$ für das Modell zugrunde gelegt.

Die Wahrscheinlichkeit für ein einzelnes Individuum lässt sich mit Hilfe von $ETT(G_c)$ leicht ausrechnen. Es ist die Wahrscheinlichkeit, dass ein Individuum Teil des übernommenen Subgraphen ist:

$$ETT(p) = \frac{ETT(G_c)}{|V|} \tag{3.35}$$

Die vollständige Herleitung der Abschätzung folgt prinzipiell denselben Argumenten und Approximationen, wie $ETT(G_c)$. Zunächst kann ein Individuums nur dann übernommen werden, wenn es sich im selben Subgraphen wie das initial beste Individuum zu befindet. In beiden

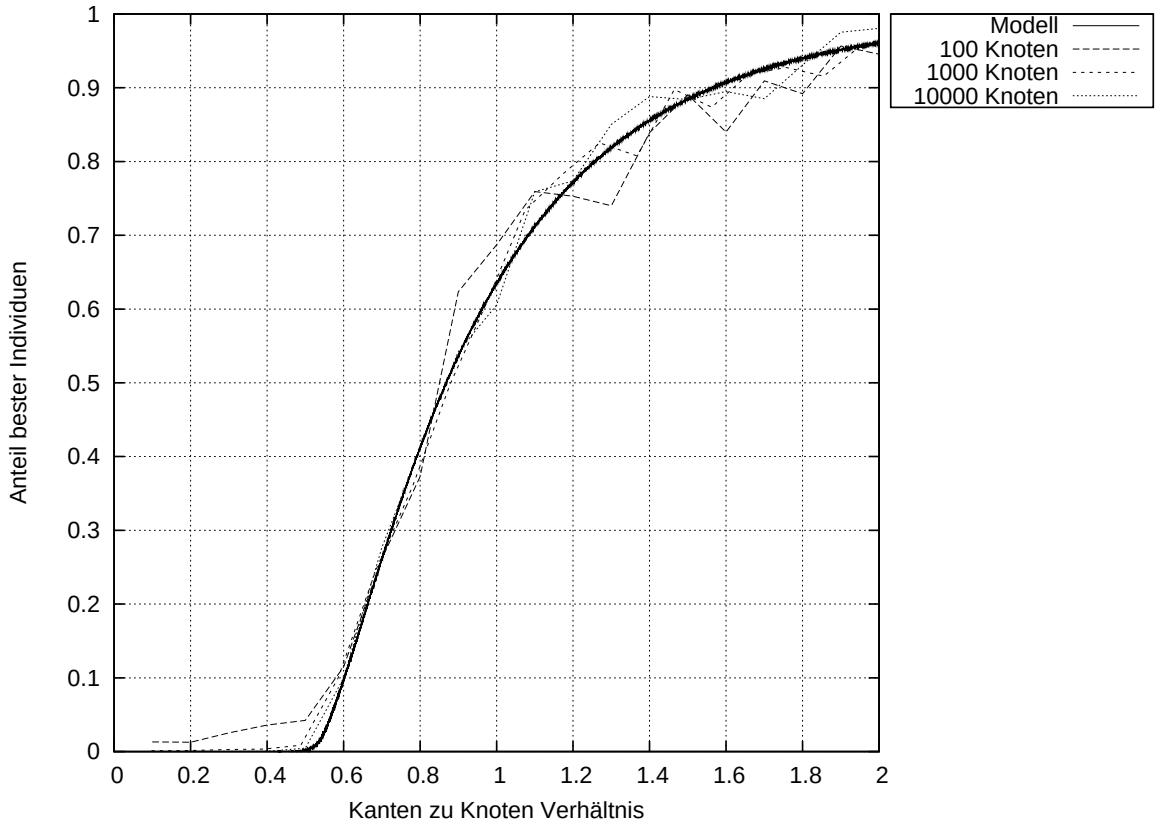


Abbildung 3.11: $ETT(G_c)$

Fällen beträgt die Wahrscheinlichkeit für einen beliebigen Subgraphen $\frac{|V_{c_i}|}{|V|} | 1 \leq i \leq r$. Insgesamt ergibt sich damit die Gesamtwahrscheinlichkeit 3.36. Die unbekannte tatsächliche der Subgraphen G_{c_i} lässt sich approximieren durch die bekannte des größten Subgraphs G_c^{max}, unter Annahme $|V_c^{max}| \gg |V_{c_i}|$.

$$ETT(p) = \frac{|V_{c_1}|}{|V|} \cdot \frac{|V_{c_1}|}{|V|} + \frac{|V_{c_2}|}{|V|} \cdot \frac{|V_{c_2}|}{|V|} + \ldots + \frac{|V_{c_r}|}{|V|} \cdot \frac{|V_{c_r}|}{|V|} \tag{3.36}$$

$$= \left(\frac{|V_{c_1}|}{|V|}\right)^2 \cdot \left(\frac{|V_{c_2}|}{|V|}\right)^2 + \ldots + \left(\frac{|V_{c_r}|}{|V|}\right)^2 \tag{3.37}$$

$$\approx \left(\frac{|V_c^{max}|}{|V|}\right)^2 \tag{3.38}$$

$$\approx \left(tb\left(\frac{|E|}{|V|}\right)\right)^2 \tag{3.39}$$

$$\approx \frac{ETT(G_c)}{|V|} \tag{3.40}$$

Für die erweiterte Takeover time wird exakt der Subgraph G_{c_i} mit Kopien des besten Individuums gefüllt, welcher zu Beginn das beste Individuum enthält ($v_i \in G_{c_i}$). Für $ETT(T)$ wird daher lediglich die Zeit gemessen, bis G_{c_i} übernommen wurde. Daher lässt sich die

Zeitabschätzung zur Übernahme zurückführen auf Untersuchungen zur Übernahme zufälliger Graphen bei bekanntem Vernetzungsgrad (siehe [84]). Dieser lässt sich aus der bekannten Anzahl der Knoten und Kanten ableiten. Der Fall $ETT(T)$ für Graph G lässt sich somit auf den Fall der Übernahme von $ETT(G_{c_i})$ reduzieren. Generell ist zu bemerken, dass Abschätzungen des Durchmessers in zufälligen Graphen lediglich für $\frac{|E|}{|V|} > 2$ in [109] behandelt. Dieses Verhältnis wird hier zunächst nicht betrachtet.

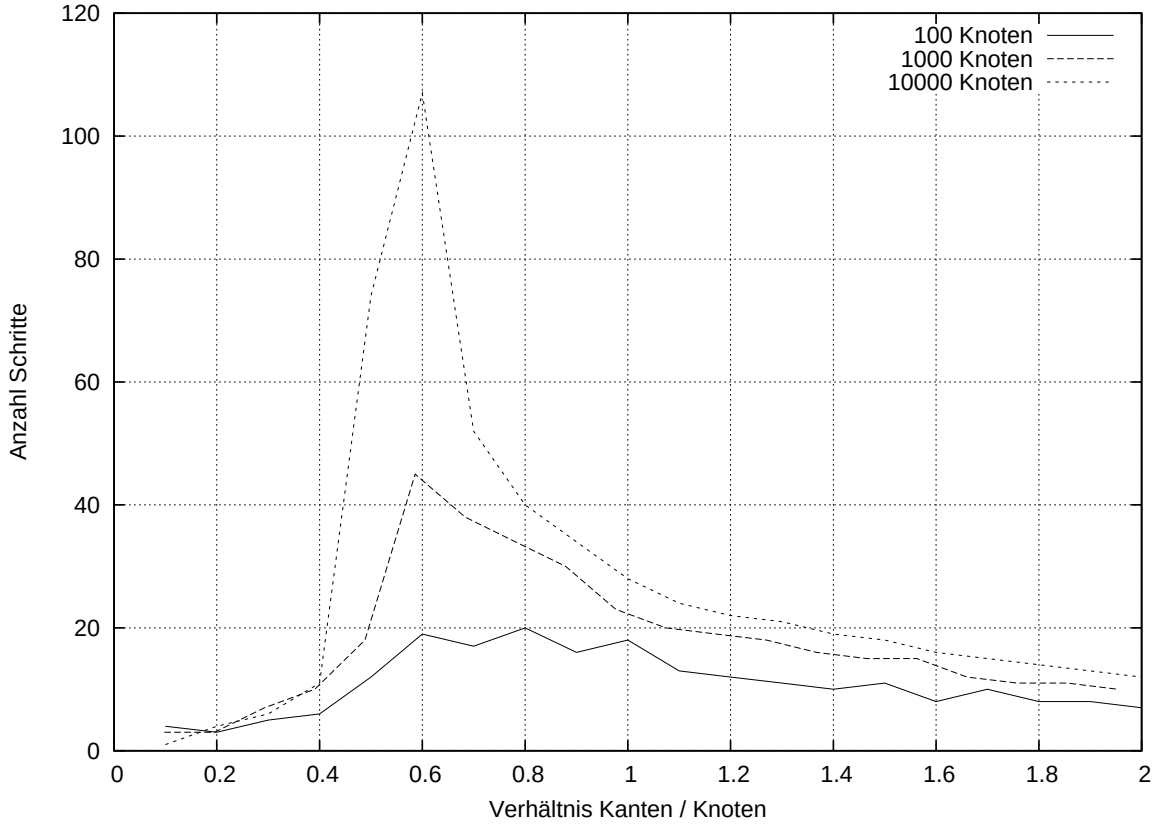


Abbildung 3.12: Benötigte Schritte ungerichteter Graphen mit 100,1000,10000 Knoten und Kanten/Knotenverhältnis von $0.1 - 2.0$

Abbildung 3.12 zeigt die tatsächlich benötigte Anzahl von Schritten für unterschiedliche Graphgrößen und einem Kanten/Knotenverhältnis $0.1 \leq \frac{|E|}{|V|} \leq 2.0$. Hier zeigt sich ein in Abhängigkeit der Graphgröße mehr oder weniger ausgeprägtes Maximum. Zunächst ist zu bemerken, dass bis $\frac{|E|}{|V|} < 0.4$ eine moderate Zunahme der benötigten Schritte zu sehen ist. Hierfür ist die Gleichverteilung der eingefügten Kanten verantwortlich. Zunächst werden bei idealer Gleichverteilung durch sukzessives Einfügen von $\frac{|V|}{2}$ Kanten auch $\frac{|V|}{2}$ Knotenpaare erzeugt. Bis dato ist der Durchmesser (längster Pfad zwischen den Knoten innerhalb eines Graphen) solcher Subgraphen gering. Dies ändert sich, wenn weiter Kanten eingefügt werden. Mit steigender Anzahl der Kanten werden wiederum die bestehenden Subgraphen (idealerweise Knotenpaare) zu größeren Subgraphen verbunden und so weiter. Daher steigt der Durchmesser solcher Subgraphen weiter an. Ein Maximum wird bei $\frac{|E|}{|V|} = 0.6$ erreicht. Das Einfügen weiterer Kanten führt primär, zu einem Anstieg des Vernetzungsgrades der Subgraphen (siehe Gleichungen 3.25,3.26). Hiermit entstehen innerhalb der Subgraphen neue

Verbindungen, die 'Abkürzungen' schaffen, und die längsten Pfade verkürzen. Stärkere Vernetzung führt demzufolge zur schnelleren Übernahme von Netzwerken.

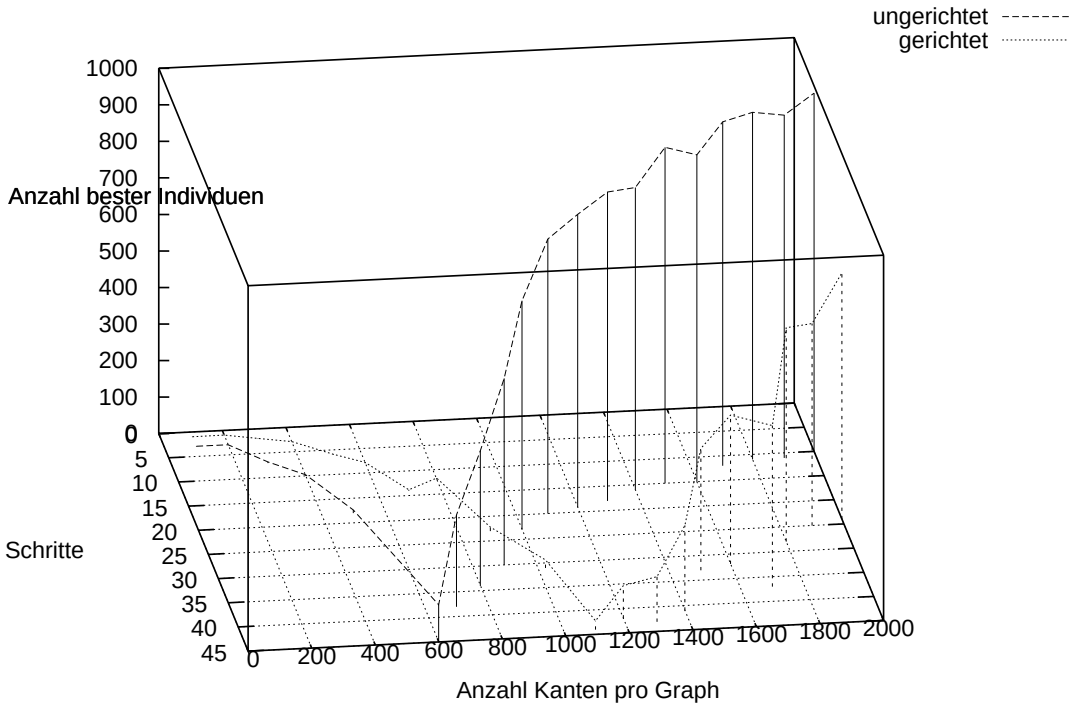


Abbildung 3.13: Benötigte Schritte und Anzahl übernommener Individuen für einen Graph mit 1000 Knoten

In Agentenumgebungen nach Definition 3 sind die Kanten zur Kommunikation gerichtet. Daher ist zumindest für einen initialen Graphen gerichtete Kanten anzunehmen. Hierbei ist jedoch zu berücksichtigen, dass Agentenbasierte Kommunikation üblicherweise das Konzept des Absenders in Nachrichten verwendet (vgl. dazu das Jade Framework [20]). Beim Aufbau eines Kommunikationsnetzwerkes ist daher der beginnende Nachrichtenfluss gerichtet und kann je nach lokalem Bedarf in mehrstufige Kommunikationsprotokolle münden. Weiterhin ist beim Aufbau neuer Kommunikationsverbindungen die Verbindung ebenfalls zunächst gerichtet, da in Agentensystemen mit reaktiven Agenten zumeist mit Hilfe von Diensteverzeichnissen einseitig aufgebaut wird (vgl. Abschnitt 2.2.4). Der Informationsfluss ist daher primär ein gerichteter Fluss, welcher in Kommunikationssituationen ungerichtet wird. Es obliegt daher letztendlich dem Agenten selbst, Informationen weiterzuleiten. Diese Betrachtungsweise letztendlich zur Analyse der Transferfunktion und Strategie von Agent und Multiagentensystem hinsichtlich ihrer 'Durchlässigkeit' für Informationen. Der in dieser Arbeit verwendete Begriff des ökonomischen Agenten (siehe Abschnitt 3.1.6.4) handelt aufgrund eigener Nutzenmaximierung. Daher wird je nach Situation, Historie und Kontext Information nur bei zu erwartendem ökonomischem Nutzen weitergeleitet. Eine generelle Aussage hinsichtlich der Informationsweiterleitung lässt sich durch diese Betrachtungsweise nicht ableiten, daher scheint

an dieser Stelle eine vergleichende Untersuchung ungerichteter Graphen sinnvoll.

Ein ungerichteter Graph mit n Kanten lässt sich durch einen gerichteten Graphen mit $\frac{n}{2}$ Kanten ersetzen. Daher ist zu erwarten, dass in einem ungerichteten Graphen bei gleich bleibenden Eigenschaften etwa halb so viele Individuen pro Zeitintervall übernommen werden können. Das bedeutet für die erweiterte Takeover time, das auch nur halb so viele Individuen abschließend übernommen werden, bzw. mit halber Wahrscheinlichkeit im Vergleich zu ungerichteten Graphen übernommen werden.

$$ETT_u(p) \approx \left(tb \left(\frac{|E|}{2|V|} \right) \right)^2 \tag{3.41}$$

$$ETT_u(G_c) \approx |V| \cdot \left(tb \left(\frac{|E|}{2|V|} \right) \right)^2 \tag{3.42}$$

$$ETT_u(T) = 2 \cdot ETT(T) \tag{3.43}$$

Zur Evaluation der vorgestellten Charakteristika wurde eine Reihe von Simulationen durchgeführt. Diese Simulation wurden mit einer Graphgröße von 1024 Knoten berechnet. Es wurden jeweils 50 Läufe gemittelt. Für jede Simulation wurden 1024 Schritte simuliert, um eine vollständige Ausbreitung für alle Graphenstrukturen sicherzustellen. In proaktiven Agentenumgebungen sind Interaktionen initial gerichtete Interaktionen. Daher wurden alle Simulationsläufe in zwei Versionen durchgeführt: für gerichtete und ungerichtete Graphen. In einer ersten Untersuchung werden pro Graph zufällig Kanten (gerichtet Abb. 3.14a,c, ungerichtet Abb. 3.14b,d) eingefügt und danach zufällig ein Agent als fittestes Individuum deklariert. Das Update der Knoten erfolgt in allen Simulationen synchron.

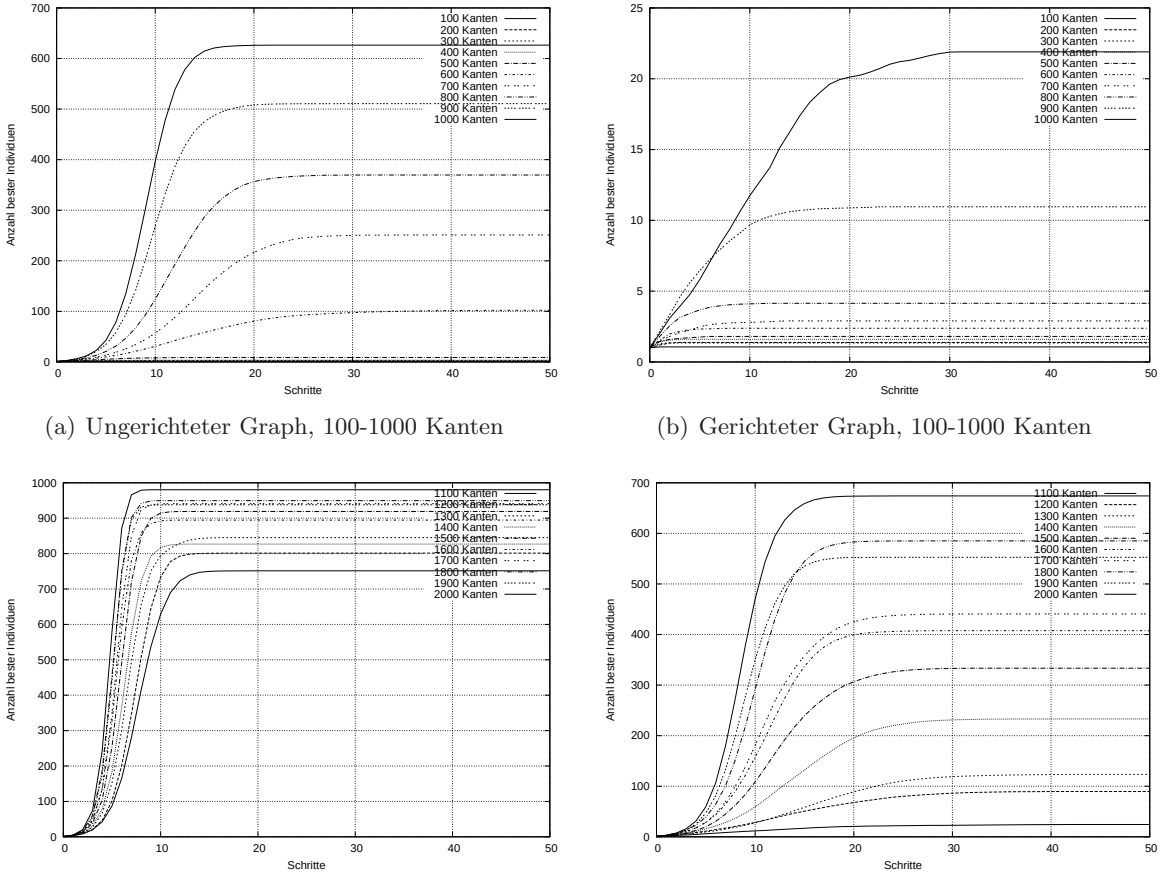
	$ETT(T)$	$ETT(G_c)$	$ETT(p)$
gerichtet Berechnung	-	1,6	0,002
gerichtet Experiment	31	1,8	0,002
ungerichtet Berechnung	-	632,9	0,633
ungerichtet Experiment	23	626,4	0,626

Tabelle 3.3: Eigenschaften der erweiterten Takeover (anhand Tabelle 3.1) time für einen Graphen mit 1000 Knoten und 1000 Kanten und experimentelle Resultate

In einer zweiten Untersuchung wurde von der Graphenebene auf die Agentenebene gewechselt (Abbildung 3.16. Hierbei wurden ausgehend von jedem Agenten (in virtueller Umgebung v_a) eine Anzahl κ von Kanten zu zufällig ausgewählten Agenten erzeugt. Damit ist der Grad der Agentenumgebung $deg(v_a) \geq \kappa$. Die Experimente zur Bestimmung der Takeover time wurden in Analogie zu den vorherigen Experimenten wieder für den gerichteten und ungerichteten Fall durchgeführt.

Aus Agentensicht zufällig vernetzte Graphen haben gegenüber den zufälligen Graphen den Vorteil, dass jeder Agent durch mindestens κ Kanten mit anderen Agenten vernetzt ist. Hierbei werden pro Agent κ zufällige Kanten zu anderen Agenten generiert. Die Wahrscheinlichkeit der Selbstvernetzung p_{self} geht hierbei gegen 0:

$$\lim_{|V| \to \infty} p_{self} = \left(\frac{2}{|V| \cdot (|V| + 1)} \right)^{\kappa} = 0 \tag{3.44}$$



(a) Ungerichteter Graph, 100-1000 Kanten

(b) Gerichteter Graph, 100-1000 Kanten

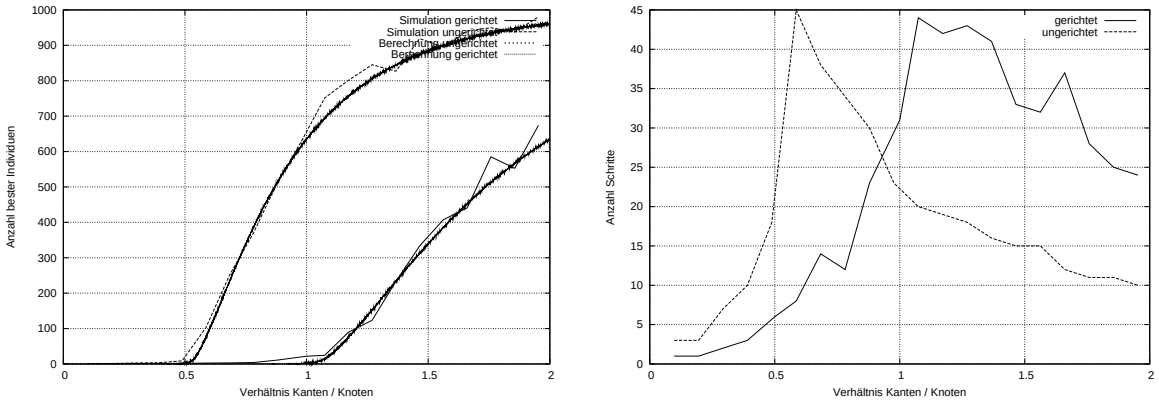
(c) Ungerichteter Graph, 1100-2000 Kanten

(d) Gerichteter Graph, 1100-2000 Kanten

Abbildung 3.14: Takeover times für ungerichtete (linke Spalte) und gerichtete (rechte Spalte) Graphen mit 1024 Knoten und unterschiedlicher Anzahl von Kanten

Aus Agentensicht vernetzte Graphen haben pro Agent mindestens eine Kante ($\kappa \geq 1$). Das bedeutet, dass jeder Agent mit κ weiteren Agenten vernetzt ist. Die Wahrscheinlichkeit zur Selbstvernetztung und der Mehrfachvernetzung mit einem anderen Agenten sind für $|V| \gg \kappa$ entsprechend gering. Die Ergebnisse von 3.16 gegenüber den global zufällig vernetzten Graphen 3.14 zeigen gleiche Resultate. Hierbei wird die maximal mögliche Anzahl von Agenten gleichzeitig erreicht und auch nahezu die gesamte Agentenpopulation wird bereits ab (statistisch) zwei Kanten pro Agent übernommen.

Wie im vorherigen Abschnitt deutlich gemacht (siehe [124] und Abbildung 3.9), besitzen schwach vernetzte zufällige Graphen keine zusammenhängende Struktur. Dies gilt für Graphen, die statistisch weniger zwei Kanten pro Knoten enthalten; im Besonderen jedoch für ein Verhältnis von Kanten zu Knoten kleiner als 0.5. Hierbei kann nicht immer der gesamte Graph übernommen werden. Die Voraussetzung der vollständigen Vernetzung von Graphen ist daher wie von Giacobini et al. [84] als nicht gegeben angesehen werden. Die entwickelten Modelle berücksichtigen daher Teilübernahmen und ermöglichen die Berechnung des übernommenen Anteils der Population. Zur Bestimmung der Zeit bis zur Übernahme der maximal möglichen Teilpopulation wurde eine Abhängigkeit von der Anzahl der Knoten und des Vernetzungs-



(a) $ETT(G_c)$ (b) $ETT(T)$

Abbildung 3.15: Vergleich ETT für gerichtete und ungerichtete Graphen mit 1000 Knoten: a) Anzahl bester Individuen jeweils Simulation und Berechnung, b) Vergleich der Schritte bis Übernahme abgeschlossen

grades festgestellt. Hierfür sind bis zur genauen Abschätzung und Berechnung noch weitere Untersuchungen notwendig.

3.3.5.3 Schwach vernetzte, dynamische und zufällige Graphen

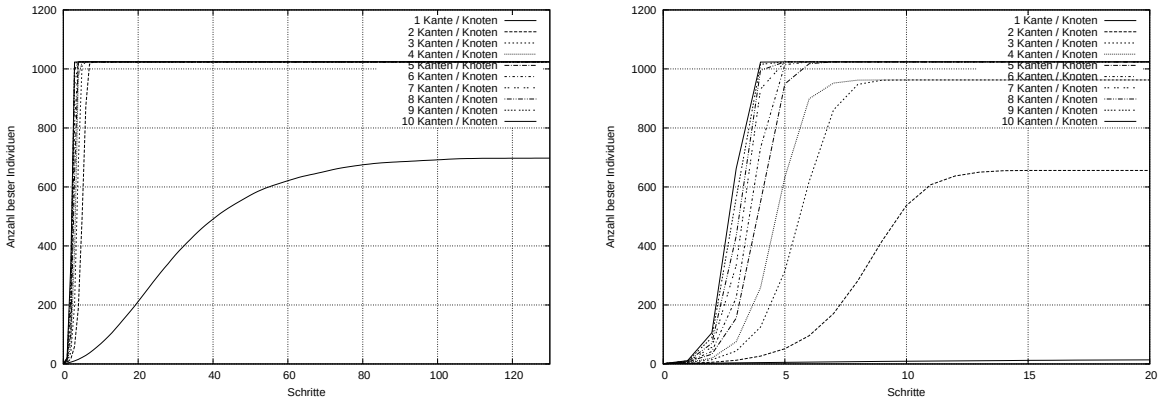
Die Untersuchung dynamischer Graphen bildet die logische Weiterführung der Takeover time für verteilte Systeme. Reale Netzwerke, wie die betrachteten Peer-to-Peer Grid Strukturen oder virtuelle Organisationen sind im seltensten Falle statisch, so dass eine Untersuchung dynamischer Aspekte gerechtfertigt erscheint.

Je nach Anteil der Dynamik im Vergleich zu den bisher betrachteten statischen Graphen lassen sich dynamische Graphen als *Erdős Rényi Zufallsgraphen* [63] (ER Graph) betrachten. Im Vergleich zur Knoten und Kanten Modellierung mittels $G(V, E)$ wird lediglich die Wahrscheinlichkeit für das Vorhandensein von Kanten betrachtet: $G(V, p)$. Hierbei wird die Gesamtmenge aller Graphen mit $|V|$ Kanten und der Wahrscheinlichkeit der Existenz einer Kante mit $p = p(n)$ modelliert. Die Anzahl der Kanten $|E|$ eines solchen Graphs ist dann mittels $|E| = p(n)\binom{|V|}{2}$ gegeben. Die beiden Modelle $G(V, E)$ und $G(V, p)$ sind homogen im Sinne ihrer resultierenden Eigenschaften (z.B. Grad der Knoten), jedoch sind ER Graphen formal einfacher zu erfassen.

Ziel dieses Abschnittes ist es daher, Dynamikaspekte mit den bekannten Resultaten für zufällige Graphen [63, 109] zu vergleichen und gleichzeitig die Unterschiede zu statischen Graphen hinsichtlich der takeover time herauszuarbeiten.

Im Wesentlichen sollte Dynamik kürzere Takeover times und damit einen höheren Selektionsdruck erzeugen. Denkbar ist im günstigsten Fall eine neu entstehende Kante, welche den Transfer von Kopien des besten Individuums in entfernte Regionen erlaubt oder bislang nicht verbundene Subgraphen verbindet. Somit wird im Gegensatz zu regulären Gitterstrukturen, wie beispielsweise in cEA's, ein schnellerer Informationsfluss ermöglicht. In dynamischen Graphen wird nach hinreichend langer Zeit der gesamte Graph übernommen.

$$\lim_{t \to \infty} ETT(G_c) = |V|$$



(a) Ungerichteter Graph, 1-10 Kanten pro Knoten

(b) Gerichteter Graph, 1-10 Kanten pro Knoten

Abbildung 3.16: Vergleich *ETT* für ungerichtete(a) und gerichtete(b) Graphen mit 1024 Knoten und pro Knoten 1-10 Kanten

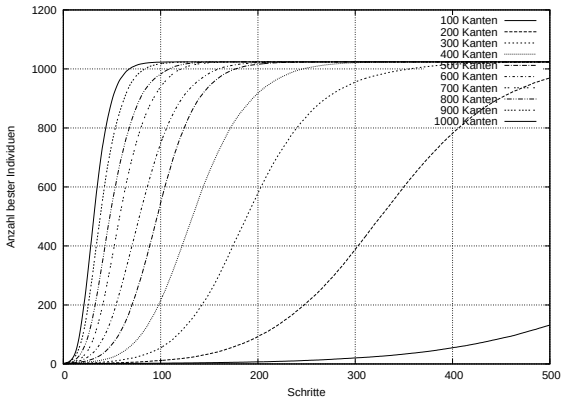
Die Versuchsergebnisse sind ähnlich wie bei statischen Graphen auch jeweils in Experimente mit gerichteten und ungerichteten Graphen unterteilt. Abbildung 3.17 zeigt in *a*) und *c*) jeweils gerichtete Graphen mit $100-1000$ Kanten pro Graph und $1-10$ Kanten pro Agent. In *b*) und *d*) sind jeweils gerichtete Graphen mit $100-1000$ Kanten pro Graph und $1-10$ Kanten pro Agent abgebildet.

Abbildung 3.17 zeigt eine deutlich kürzere Takeover time für ungerichtete Graphen gegenüber gerichteten Graphen. Die Informationen in ungerichteten Graphen können bestehende Verbindungen in beiden Richtungen durchlaufen. Dieses Verhalten ist ähnlich zu den statischen Graphen in Abschnitt 3.3.5.2. Im Gegensatz zu statischen Graphen wird immer die gesamte Population in endlicher Zeit übernommen.

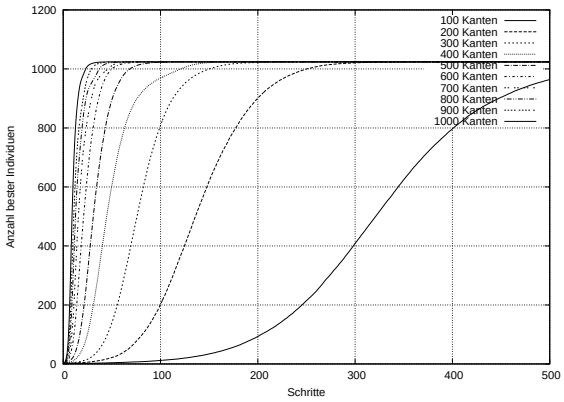
3.3.5.4 Gerichtete und ungerichtete Graphen

Die Takeover time wurde im Rahmen dieser Arbeit ausführlich analysiert. Beginnend mit der Einführung des Threads und Buttons Phänomen in zufälligen Graphen und von Erdős und Rényi Graphen [63] konnte die resultierende Größe des maximalen verbundenen Subgraphen in nachfolgende theoretische und praktischen Analysen verwendet werden. Die Analyse wurde zunächst nach zwei unterschiedlichen Gesichtspunkten durchgeführt: statische und dynamische Graphen, die wiederum je in Untersuchungen gerichteter und ungerichteter aufgeteilt wurden. Hierbei trat der verwendete Ansatz der Untersuchung von Grid und Peer-to-Peer Strukturen in den Vordergrund, so dass sowohl gerichtete Graphen, als auch konkrete dynamische Graphstrukturen untersucht wurden. Beides stellt im Kontext der Takeover time Neuland dar. Hierzu wurde die erweiterte Takeover time (*ETT*) eingeführt, um insbesondere nicht vollständig übernommene Graphen zu quantifizieren. Für die Bewertung der Ergebnisse wurden Simulationen durchgeführt, die a) die theoretisch vorausgesagten Ergebnisse (statische Graphen) bestätigten und b) einen Vergleich mit anderen Forschungsergebnissen zu erlauben.

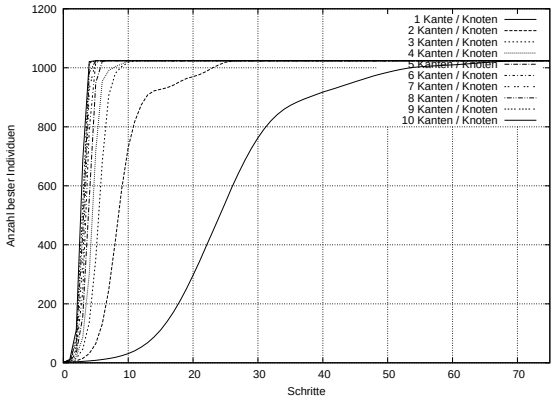
Die Evaluation erfolgt in zwei Experimentgruppen, je eine für statische und eine für dynamische Graphen mit jeweils 1024 Knoten. Je Experiment wurden 50 Wiederholungen durchgeführt. Die Ergebnisse beider Experimentgruppen wurden jeweils mit Resultaten anderer



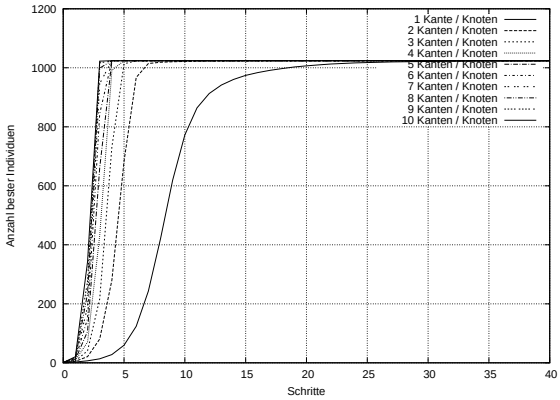
(a) gerichteter Graph 100-1000 Kanten pro Graph



(b) ungerichteter Graph 100-1000 Kanten pro Graph



(c) gerichteter Graph 1-10 Kanten pro Knoten



(d) ungerichteter Graph 1-10 Kanten pro Knoten

Abbildung 3.17: Vergleich $ETT(T)$ für gerichtete(a,c) und ungerichtete(b,d) Graphen mit 1000 Knoten und unterschiedlicher Anzahl von Kanten pro Knoten/Graph. Die Wahrscheinlichkeit zur Ersetzung einer Kante beträgt $p = 0.1$

Forschungen verglichen. Hierzu wurde die Takeover time für cEA mit L5-Nachbarschaft (vgl. Abschnitt 2.3.6 bzwl. [207]) und zufällige ER-Graphen ([63, 84]) herangezogen.

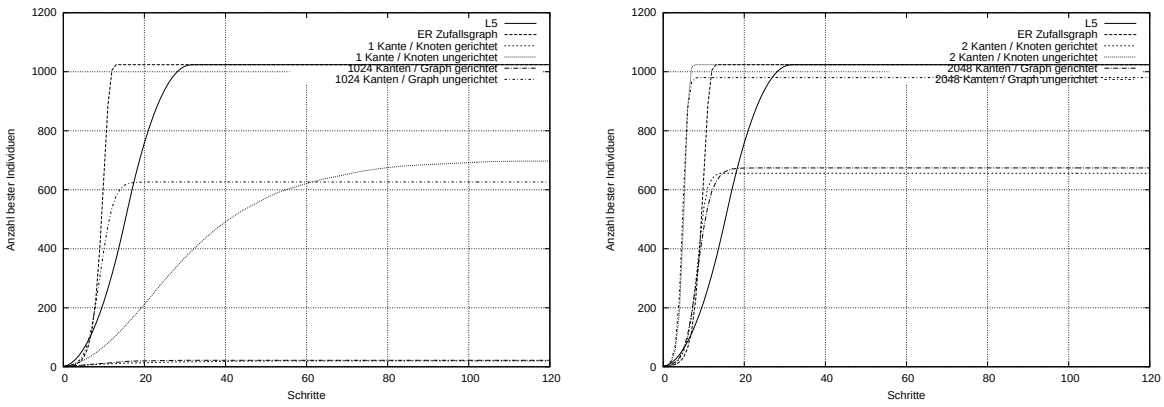
Für die Experimentgruppe mit statischen Graphen wurden für Graphen mit 1024/2048 Knoten herangezogen. Die verwendeten Graphen sind:

- Je einer gerichteten Kante pro Knoten, zu einem zufälligen Knoten führend (insgesamt 1024/2048),
- Je einer ungerichteten Kante pro Knoten, zu einem zufälligen Knoten führend (insgesamt 1024/2048)
- Je 1024/2048 gerichtete Kanten pro Graph
- Je 1024/2048 ungerichtete Kanten pro Graph

Abbildung 3.18a) zeigt die Auswertung des Vergleiches mit 1024 Kanten. Hierbei zeigt sich, dass die ungerichteten Graphen auf je einen Anteil von 628 (1 Kante/Knoten) bzw. 697

(1024 Kanten pro Graph) von 1024 möglichen Kopien erreichen konnten. Im ungerichteten Fall lag dieser Anteil bei unbedeutenden 20 bzw. 22 Kopien. Damit ist in zufälligen statischen Graphen bei einem Konten zu Kanten Verhältnis von 1 eine Komplettübernahme sehr unwahrscheinlich. Die Zeit bis zur Übernahme eignet sich daher nur bedingt als Vergleichskriterium mit den Referenzgraphen. Im Fall der 1024 zufällig verteilten Kanten ist die Zunahme der Anzahl der besten Kopien bis zum Stillstand in etwa zwischen den gewählten Referenzgraphen, während für je eine Kante pro Knoten ein deutlich flacherer Anstieg erfolgt. Damit erzeugen statische und ungerichtete Graphenstrukturen mit Kanten/Knotenverhältnis von 1 einen deutlich geringeren Selektionsdruck als beispielsweise L5 Strukturen. Für gerichtete Graphen ist der Selektionsdruck verschwindend gering, da ein kaum nennenswerter Anteil des Graphen übernommen wird.

Abbildung 3.18b) vergleicht die Referenz Takeover times mit Graphen mit je 2048 Kanten. Hierbei zeigt sich eine deutlich gestiegene Anzahl übernommener Knoten im Vergleich zu a). Bemerkenswert ist ein steilerer Anstieg der ungerichteten Graphen als bei den Referenzdaten. Im Fall von 2 Kanten pro Knoten wird wiederholbar der gesamte Graph übernommen. Für 2048 zufällig verteilte Kanten werden im Schnitt 981 von 1024 Knoten übernommen. Die Anzahl der übernommenen Knoten in gerichteten Graphen entspricht in etwa dem ungerichteter Graphen mit halb so vielen Kanten (vgl. Abschnitt 3.3.5.2, Gleichungen 3.41, 3.42, 3.43). Damit wird im Falle von 2 ungerichteten Kanten pro Knoten deutlich weniger Zeit benötigt, als in Gitterstrukturen von cEA.

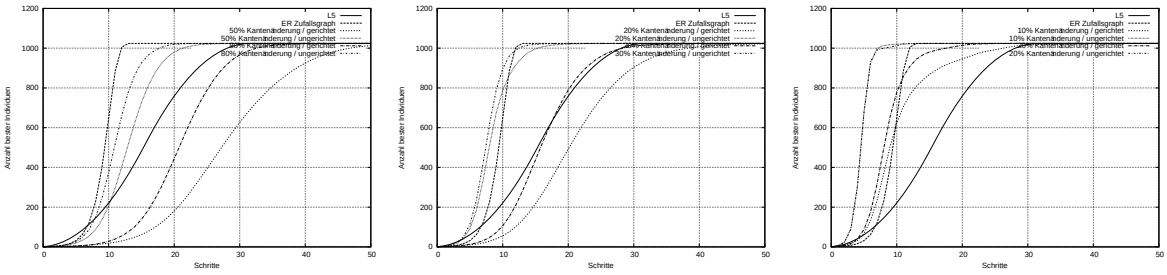


(a) 1024 Kanten (b) 2048 Kanten

Abbildung 3.18: $ETT(T)$ in statischen Graphen mit 1024 Knoten und 1024 bzw. 2048 Kanten. Vergleich der Takeover time mit Referenzgraphen cEA (L5-Nachbarschaft) und ER-Zufallsgraphen

Die zweite Experimentgruppe untersuchte den Effekt dynamischer Kanten auf den Einfluss der Takeover time. In jedem Schritt wurden mit einer Wahrscheinlichkeit von p Kanten umgesetzt. Hier zeigt sich, dass immer alle Knoten übernommen werden, solange genügend Zeit zur Verfügung steht. Im Unterschied zu statischen Graphen werden früher oder später die Kanten so verändert, dass der Informationsfluss alle Knoten erreicht.

Abbildung 3.19a zeigt gerichtete/ungerichtete Graphen mit 512 Kanten. Die hohe Dynamik in ungerichteten Graphen mit $p = 0.5$ bzw. $p = 0.8$ sorgt für eine mit L5 vergleichbare Takeover time bei deutlich reduzierter Kantenanzahl. Für $p = 0.8$ wird lediglich ein Viertel



(a) 512 Kanten (b) 1024 Kanten (c) 2048 Kanten

Abbildung 3.19: $ETT(T)$ in dynamischen Graphen mit 1024 Knoten und 512, 1024 bzw. 2048 Kanten und unterschiedlichen Kantenänderungswahrscheinlichkeiten. Vergleich mit takeover time in Referenzgraphen cEA(L5) und ER-Zufallsgraphen

der in einem gitterartigem cEA benötigten Kanten vorausgesetzt. Steigt die Anzahl der Kanten an, ist die erforderliche Dynamik deutlich geringer. 3.19b und c zeigen Simulationen mit 1024 und 2048 Kanten. Hier wird lediglich eine Dynamik von $p = 0.3$ bzw. $p = 0.2$ vorausgesetzt, um geringere Takeover times zu erhalten. Im Fall von 2048 Kanten reicht sogar die Verwendung gerichteter Kanten aus. Obwohl der Anstieg der Kurven im ungerichteten Fall steiler ist, so verläuft die asymptotische Annährung an das Maximum sehr flach, besonders im gerichteten Fall.

Abbildung 3.19a verdeutlicht die Zahlen und Unterschiede der dargestellten Simulationsläufe noch einmal im Vergleich mit L5.

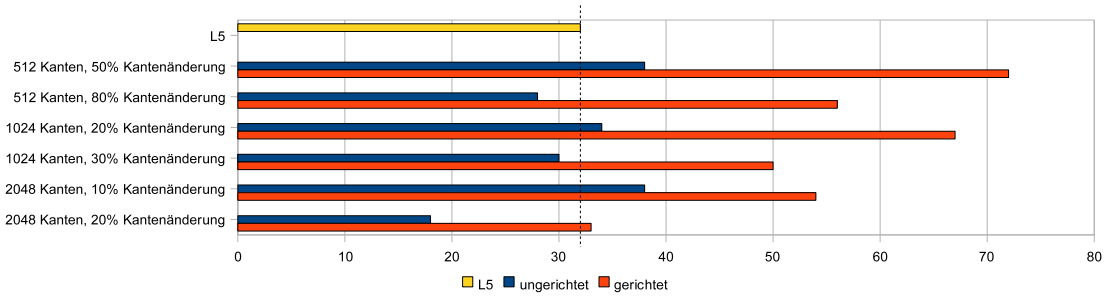


Abbildung 3.20: $ETT(T)$ in dynamischen Graphen mit 1024 Knoten und 512, 1024 bzw. 2048 Kanten und unterschiedlichen Kantenänderungswahrscheinlichkeiten. Vergleich mit takeover time in Referenzgraphen cEA(L5)

3.3.5.5 Zusammenfassung

Abschließend lässt sich sagen, dass der induzierte Selektionsdruck statischer Zufallsgraphen vergleichbar mit cEA und auch globaler Selektion ist. Dynamische Graphen haben einen etwas höheren Selektionsdruck. Das bedeutet auch für schwach vernetzte Graphen wie beispielsweise in P2P Systemen oder virtuellen Organisationen einen vergleichbaren Informationsfluss zwischen verbundenen Knoten. In statischen gerichteten/ungerichteten Graphen findet ab 5/2 Kanten pro Knoten praktisch eine Übernahme des gesamten Graphen statt. In dynamischen

Graphen findet in endlicher Zeit immer eine Gesamtübernahme statt, die mit höherer Dynamik abnimmt. Dynamische Graphen bieten auch bei geringem Vernetzungsgrad $\frac{|E|}{|V|} < 1$ gute Informationsweiterleitung. Damit besteht für dynamische Strukturen generell kein Nachteil hinsichtlich des Selektionsdrucks bzw. der Takeover time gegenüber 2D Gitterstrukturen. Der Einsatz von evolutionären Verfahren in zufällig vernetzten Strukturen mit Topologieänderungen besitzt ähnlichen Selektionsdruck wie bereits gut untersuchte Gitterstrukturen.

Die Verteilung von Informationen in Graphen ist ein wichtiger Aspekt zur Bearbeitung verteilter Problemstellungen. Die Verteilung der Gesamtlösung auf die einzelnen Individuen (Knoten) und damit der Grad der Dezentralität ist ebenfalls ein wichtiger Aspekt in komplexen Informationssystemen.

3.3.6 Dezentralität

Evolutionäre Algorithmen betrachten trotz Parallelisierung bis auf Individualebene immer die Informationen einer Gesamtlösung pro Individuum. Das bedeutet, jedes Individuum enthält eine vollständige Lösung. Wie in Kapitel 1.2.1 dargelegt, ist eine vollständige Informationsgewinnung in verteilten Systemen nicht immer möglich. Der Ansatz der evolutionären Agenten trägt diesem Umstand Rechnung durch Verteilung der Lösung auf mehrere Agenten innerhalb der Population. Dieser Abschnitt entwickelt ein Dezentralitätsmaß zur Einordnung evolutionärer Verfahren hinsichtlich ihres Verteilungsgrades.

Ein einzelner Algorithmus, der auf Populationsebene (panmiktischer EA), Subpopulationsebene (Master-Slave, dEA), Individualpopulationen (cEA) oder auf Individualebene (evolutionärer Agent) arbeitet, wird im folgenden als *evolutionärer Prozessor* bezeichnet. Dieses Konzept abstrahiert von der physikalischen Instanz eines Prozessors. Hierbei ist das Ziel, eine *Dezentralitäts*-Metrik zu entwickeln, die Aussagen darüber zulässt, über wie viel Informationen des populationsbasierten Verfahrens ein evolutionärer Prozessor verfügt. Dies macht die unterschiedlichen Lösungsansätze parallelter evolutionärer Verfahren hinsichtlich ihrer Verteilung vergleichbar. Im Falle klassischer EAs verwaltet der evolutionäre Prozessor die gesamte Population der Individuen (=Lösungen) und jedes Individuum enthält eine vollständige Lösung. Es wird davon ausgegangen, dass Lösungen im Allgemeinen gültig sind. Das andere Extrem stellen evolutionäre Agenten dar. Der evolutionäre Prozessor arbeitet hier lediglich auf Individualebene und das Individuum enthält einen Teil der Lösung. Dazwischen liegen die unterschiedlichen Arten paralleler EA (siehe Kapitel 2.3.4).

In die Dezentralitätsbetrachtungen fließen sowohl der Anteil der von einem evolutionären Prozessor verwalteten bzw. gesteuerten Individuen als auch deren jeweiliger Lösungsanteil ein:

$$\mathcal{Z}_i = 1 - \frac{|A_{sub}|}{|A|} \cdot \upsilon \tag{3.45}$$

Die Gesamtmenge der Individuen sei mit A und die Menge der auf evolutionärem Prozessor i verwalteten Individuen mit A_{sub} bezeichnet. Der Anteil an der Gesamtlösung sei mit $\upsilon \in [0,1]$ bezeichnet. Damit ist die Dezentralität $\mathcal{Z} \in [0,1[$ bei größtmöglicher Verteilung der Informationen nahe 1 und bei Konzentration auf einem einzigen evolutionärem Prozessor 0. Tabelle 3.3.6 listet die jeweiligen Werte bzw. Bereiche für unterschiedliche Algorithmen auf und ermöglicht den Vergleich mit evolutionären Agenten.

Im panmiktischen Modell kontrolliert der evolutionäre Prozessor alle Individuen, wobei jedes die gesamte Lösung enthält. Im Master-Slave Modell schwankt dieser Wert zwischen

EA-Typ	Individuen in (Sub)Population	υ
Panmictic	$\|A\|$	1
Master-Slave	$\|A_{sub}\|$,$\|A\|$	1
dEA	$\|A_{sub}\|$	1
cEA	1	1
Evolutionäre Agenten	1	$\frac{1}{\|A\|} \ldots \|A\|$

Tabelle 3.4: Ausgewählte evolutionäre Verfahren und ihre Dezentralitätsfaktoren 'Individuen in der (Sub)Population' und υ

$|A_{sub}|$ und $|A|$, da der Master die Individuen für Berechnungen auf die einzelnen Slaves verteilt. Die Kontrolle liegt damit vorübergehend bei den Slaves. Erst nach erfolgtem Eingang der Ergebnisse beim Master ist die gesamte Populationsinformation wieder zentral. Im dEA Modell haben die Subpopulationen $|A_{sub}|$ zumeist gleiche Größe. In cEA befindet sich in einer Zelle zumeist ein Individuum. Für alle EA-Typen bis auf evolutionäre Agenten beträgt der Anteil eines Individuums an der Lösung 1. Für evolutionäre Agenten hingegen wird in der Regel die Lösung verteilt erbracht und jedes Individuum läuft als eigener evolutionärer Prozessor.

Beispiel 6 (Dezentralität)
Ein dEA arbeitet mit einer Populationsgröße von $|A| = 10$ und zwei Subpopulationen $|A_{sub_1}| = |A_{sub_2}| = 5$. Jedes Individuum enthält eine vollständige Lösung ($\upsilon = 1$). Der Dezentralitätsfaktor beträgt hierfür

$$\mathcal{Z}_1 = \mathcal{Z}_2 = 1 - \frac{5}{10} \cdot 1 = 0.5$$

Ein cEA mit gleicher Populationsgröße und $|A| = 10$ Subpopulationen $|A_{sub}| = 1$ hat einen Dezentralitätsfaktor für jeden evolutionären Prozessor von

$$\mathcal{Z}_i = 1 - \frac{1}{10} \cdot 1 = 0.9$$

Der evolutionäre Agentenansatz kommt bei gleichen Voraussetzungen mit einer gleichmäßig verteilten Lösung über alle 10 Agenten ($\upsilon = 0.1$) auf folgenden Wert:

$$\mathcal{Z}_a = 1 - \frac{1}{10} \cdot 0.1 = 0.99$$

Daraus ergibt sich die in Abbildung 3.21 gezeigte Dezentralitätsrelation. Hierbei kann $\mathcal{Z}$ als ein Maß herangezogen werden, welches Aufschluss darüber gibt, inwieweit das Verfahren parallelisierbar ist.

Für konkrete Simulationen und nicht zuletzt ganz besonders in praktischen Anwendungen lässt sich υ für evolutionäre Agenten teilweise nicht genau angeben. Zum Einen ist dies dem verteilten Modell geschuldet. Hierbei sind für einen Beobachter (der die Berechnungen ausführt) nicht alle Informationen zugänglich. Auf Agentenebene ist dies per se nicht möglich. Andererseits ist anhand einer vorliegenden Lösung nicht exakt nachvollziehbar, welche Lei-

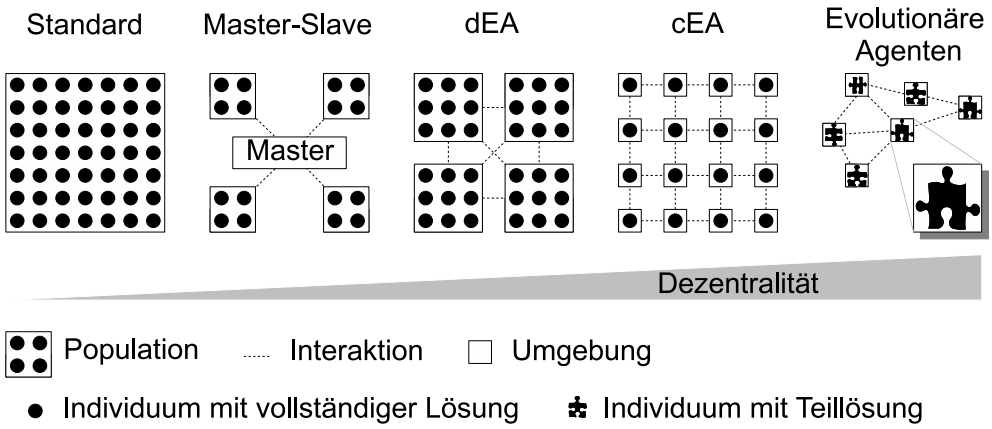


Abbildung 3.21: Dezentraliät paralleler evolutionärer Algorithmen

stung von einzelnen Akteuren erbracht wurde. Beim Betrachten eines (komplexen) Produktes ist nicht immanent ersichtlich welcher Akteur (=Agent) wie viel zum Gesamtprodukt beitrug.

Abschließend lässt sich das Dezentralitätsmaß als Indikator für populationsbasierte Verfahren verstehen. Hierbei kann $\mathcal{Z}$ als ein Maß herangezogen werden, welches Aufschluss darüber gibt, inwieweit das Verfahren parallelisierbar ist. Mit Hilfe von $\mathcal{Z}$ können unterschiedliche evolutionäre Verfahren im Hinblick auf verteilte Verarbeitung eingeordnet werden. Nicht zuletzt aus dem Problembereich der Verteilung ergeben sich daraus Vorteile für verteilte Verfahren und besonders für den Ansatz evolutionärer Agenten in stark verteilten Umgebungen.

3.4 Zusammenfassung

Das Ziel dieses Kapitels war die Erstellung einer formalen Beschreibung evolutionärer Agenten. Das formale Modell soll die Probleme der Adaption und Optimierung komplexer Informationssysteme (Verteilung, Heterogenität, Dynamik) adressieren. Ebenso sollte ein nachfragebasiertes Verhalten geschaffen werden, welches auf Bedarf aus der Umgebung reagiert. Auf Basis des formalen Modells wurde das verteilte Optimierungsproblem (VOP) definiert. Der entwickelte verteilte Optimierungsansatz vereinigt die Vorteile von evolutionären Verfahren und der Agententechnologie. Hierbei wurden folgende Aspekte ergänzt:

- Die Integration ökonomischer Modelle in den Optimierungsansatz. Hierdurch wird **Geldfluß** durch das System aus Agenten ermöglicht. Die Umgebung stellt initial Geld bereit. Per Nachfrage nach Systemdiensten gelangt Geld in das System und fließt zwischen den Agenten.

- Einführung einer **lokalen Fitness**. Der Besitz von Geld wird mit Fitness gleichgesetzt und erlaubt die lokale Bewertung von Agenten. Durch die Erweiterung auf externe lokale Fitness wird eine Bewertung zwischen Agenten durch lokale Selektion ermöglicht.

- Einführung einer agentenspezifischer **Reproduktionsschwelle** (θ), die in Abhängigkeit des vorhandenen Geldes evolutionäre Reproduktion ermöglicht oder verhindert. Zur Bestimmung von θ wurden unterschiedliche Verfahren vorgeschlagen.

Durch die Verbindung ökonomischer und evolutionärer Modelle mit Agententechnologie konnten wesentliche Punkte zur Lösung der Problemstellung beigetragen werden. Es wurde ein Verfahren entwickelt, welches Optimierung und Adaptivität aus einer Bottom-Up Perspektive ermöglicht. Trotz völliger Dezentralität von Problem und Informationen ist die Durchführung von Systemdiensten möglich. Das entstehende Gesamtsystem ist in wesentlichen Eigenschaften adaptiv, die primär als Reaktionen auf Umgebungszustände zu sehen sind.

Durch Einführung der Reproduktionsschwelle θ erfolgt die Reproduktion lokal als Reaktion auf ausreichend vorhandenes Geld. Global gesehen, ist die Populationsgröße damit eine beobachtbare emergente Eigenschaft als Reaktion auf Geld aus der Umgebung. Die Populationsgröße ist dann indirekt von der Nachfrage nach Systemdiensten abhängig. Ein System - genügend Hardware vorausgesetzt - kann immer mit der Nachfrage skalieren. Auf der anderen Seite werden nicht benötigte Dienste (Systembestandteile) automatisch beendet und Ressourcen freigeben.

Ein weiterer wichtiger Aspekt zur Adaptivität und Optimierung der Funktionalität stellt die Verbreitung erfolgreicher Strategien dar. Hierfür ist ebenfalls die Reproduktionsschwelle als Basis anzuführen. Diese ermöglicht nach Hollands Schema-Theorem die Verbreitung fitter und damit angepasster Funktionalität im System. Damit erhöht sich die Anzahl nachgefragter Dienste automatisch zu Lasten weniger nachgefragter Dienste. Die Bezeichnung Dienst gilt hier sowohl für einzelne von Agenten angebotene Funktionen als auch für komplexe, z.B. durch Organisationen angebotene Dienste. Ein weiteres emergentes Merkmal in Form von adaptivem Selektionsdruck ergibt sich aus der veränderlichen Populationsgröße. Umgebungen, die unterhalb ihrer Tragfähigkeit genutzt werden, bieten bei entsprechender Nachfrage Explorationspotential. Agenten nutzen dies in Form erhöhter Profite aus. Eine gesteigerte Anzahl von Nachkommen zur Exploration des Suchraumes ist die Folge. Andererseits führt der limitierende Faktor der Umgebung zur Erhöhung des Selektionsdruckes durch den Kampf um begrenzte Ressourcen. Damit vereint lokaler Selektionsmechanismus eine effiziente Umsetzung mit adaptivem Selektionsdruck. Zusätzlich ermöglicht dieses Verhalten ein implizites und damit selbst-adaptives Umschalten von Exploration auf Exploitation. Dies erfolgt automatisch bei Erreichung der Tragfähigkeit und bedarf keiner zusätzlichen Kontrolllogik bzw. Überwachung. Ein effizienter Mechanismus zur Steuerung der Suchstrategie entsteht.

Theoretisch wurde eine Möglichkeit aufgezeigt, um Systeme evolutionärer Agenten anpassungsfähig auch für zukünftige Funktionalität zu machen. Prinzipiell lässt sich damit ein lang laufendes System realisieren, welches adaptiv Dienste anbietet und sich auf der neue Entwicklungen der Nachfrage und der bereitgestellten Infrastruktur einstellen kann. Die Realisierung eines solchen offenen Ansatzes geht jedoch weit über die Möglichkeiten dieser Arbeit hinaus und findet daher in der praktischen Umsetzung keine weitere Betrachtung.

Ein wichtiger Aspekt ist die Erstellung verteilter Lösungen. Hierbei besteht die Gesamtlösung des Problems der Systemoptimierung in der Kombination aller Individuallösungen und ihrer Interaktionen. Das System ist damit eine Lösung für extern vorgegebene Anforderungen dar. Dieser dezentrale Ansatz ist nach aktuellem Wissensstand neu im Zusammenhang mit evolutionären Verfahren. Zur Einordnung im Vergleich mit panmiktischen und parallelen Verfahren wurde ein Dezentralitätsmaß entwickelt. Dieses erlaubt die Einordnung hinsichtlich der Verteilung der Population auf sogenannte evolutionäre Prozessoren und der Aufteilung der Gesamtlösung auf mehrere Individuen. Hiermit wurde gezeigt, dass evolutionäre Agenten prinzipiell einen höheren Grad der Verteilung als bisherige parallele Verfahren (Master-Slave, dEA, cEA) aufweisen. Dies macht ihren Einsatz in stark verteilten und komplexen Problemstellungen und Umgebungen vorteilhaft.

Kapitel 4

Prototypische Realisierung und Evaluation

Das Konzept evolutionärer Agenten wurde im vorangegangenen Abschnitt formalisiert und analysiert. Aufbauend erfolgt in diesem Abschnitt die Spezifikation und Realisierung experimenteller Prototypen zur Umsetzung und Evaluation dieses Konzepts. Die Entwicklung evolutionärer Agenten wurde an der Universität Hohenheim seit 2004 als Teilprojekt im Rahmen von *HoPIX* (Hohenheimer ProzessIntegrator eXtended) [131, 182, 183, 184] vorangetrieben. Dabei wurden Softwareagenten primär als Werkzeug eingesetzt, um die Konsequenzen theoretischer Konstrukte, Methoden und Konzepte in Wertschöpfungsnetzwerken mittels Simulation zu untersuchen. Hierbei spielt die Prozessorientierung in der Logistik eine entscheidende Rolle. Ab 2007 erfolgte die Evaluation im Rahmen einer Zusammenarbeit mit der Universität Erlangen-Nürnberg. In den dazu durchgeführten Forschungsarbeiten stand die Parallelisierung klassischer logistischer Fragestellungen und die in diesem Zusammenhang erfolgte Evaluation evolutionärer Agenten im Vordergrund [185].

In allen durchgeführten Untersuchungen diente das in Abschnitt 3 entwickelte Modell gleichermaßen als Grundlage. Hierbei spannt der Ansatz evolutionärer Agenten eine Brücke zwischen Agententechnologie und Anwendungsszenarien in verteilten Systemen. Abbildung 4.1 zeigt diesen Sachverhalt schematisch als Erweiterung des Agentenansatzes um biologische und ökonomische Aspekte.

Die prototypische Implementierung evolutionärer Agenten erfolgt in der Programmiersprache *JAVA*[1]. Dabei kamen unterschiedliche JAVA-basierte Agentensysteme bzw. Frameworks zum Einsatz: JADE [20], DIET [151] sowie eine Eigenentwicklung für [183]. Die im folgenden beschriebenen Prototypen arbeiten weitestgehend auf der gleichen Codebasis evolutionärer Agenten, die auf das jeweilige Agentensystem angepasst wurde. Zur Erhebung, Speicherung und Visualisierung von Daten in Multithreading Umgebungen wurde eigens eine Monitoring-Komponente entwickelt: *JStreaMon*[2]. Die Codebasis des Prototypen umfasst mehr als 35 Pakete mit über 100 Klassen und hat einen Umfang von ca. 10000 Anweisungen. Detaillierte Beschreibungen, die über Architektur, globale Zusammenhänge sowie spezielle Aspekte hinausgehen, finden sich in der Dokumentation des Prototypen im Javadoc Format.

Der Aufbau des Kapitels gestaltet sich wie folgt. Zunächst gibt Abschnitt 4.1 einen Überblick zur Erstellung der verwendeten Referenzarchitektur evolutionärer Agenten. Danach er-

[1] Sun JAVA SDK 1.6

[2] Siehe Projektseite unter `http://sourceforge.net/projects/jstreamon/`

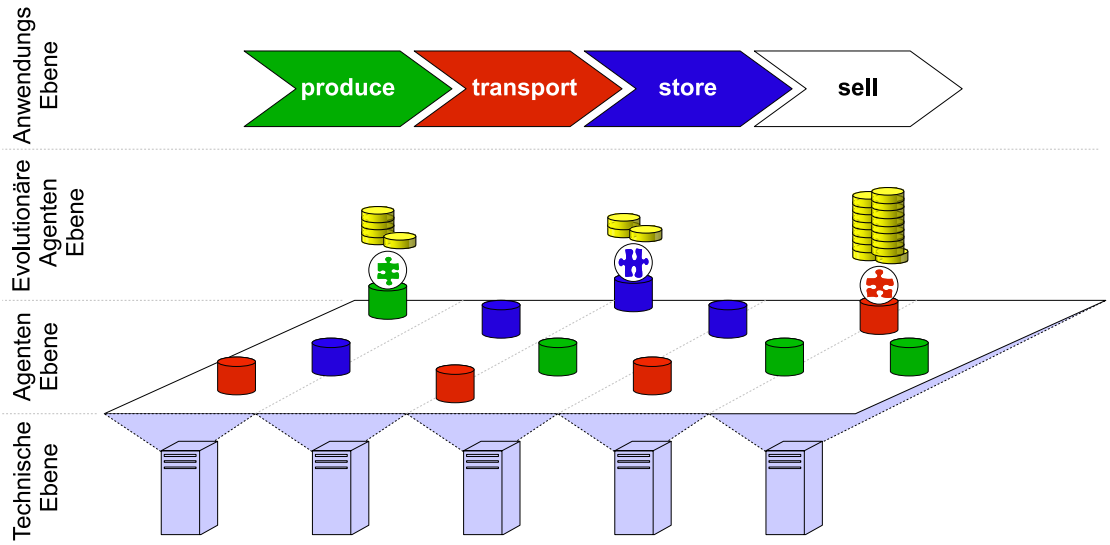


Abbildung 4.1: Evolutionäre Agenten als Brücke zwischen Anwendungsszenarien und Multi-Agenten-System

folgt die Evaluation anhand zweier Szenarien, in denen unterschiedliche Aspekte näher beleuchtet werden. Abschnitt 4.2 zeigt die Auswertung im Kontext adaptiver logistischer Netzwerke und Abschnitt 4.3 betrachtet die Fragestellung der Facility Location.

4.1 Referenzarchitektur

Zur Realisierung wurde ein weitgehend domänenunabhängigen Ansatz erstellt, der aus dem Modell (siehe Abschnitt 3.1.2) abgeleitet wurde. Hierzu zählt primär das Konzept des evolutionären Agenten mit ökonomischen und evolutionären Fähigkeiten und dessen Einbettung in das jeweils verwendete Agentensystem. Relevante domänenspezifische Implementierungsdetails sind Gegenstand der jeweiligen Experimentbeschreibungen. Für Zwecke der Datenerhebung wurde eigens eine Echtzeit-Monitoring Komponente *JStreaMon* entwickelt, um relevante Daten in dezentraler Form insbesondere in multithreaded Umgebungen einzusammeln.

Die folgenden Abschnitte bieten einen Überblick über ausgewählte Aspekte der Architektur und jeweilige Designentscheidungen. Besonderes Augenmerk gilt hierbei der Realisierung des evolutionären Agenten. Weiterhin wird kurz auf die Datenerhebung im verteilten Umfeld der Agenten sowie die verwendeten Agentensysteme eingegangen. Die grafische Darstellung erfolgt weitgehend mit Hilfe der Beschreibungssprache *UML* [14, 194].

4.1.1 Architektur

Abbildung 4.2 zeigt die wesentlichen aus dem formalen Modell abgeleiteten Anwendungsfälle evolutionärer Agenten und eines möglichen Nutzers. Hierzu zählt aus Agentensicht:

Nachrichten verarbeiten Um das Autonomieprinzip nicht zu verletzen, sind formal auf einem Agenten keine Methodenaufrufe zugelassen. Stattdessen werden alle Absichten per Nachrichtenübermittlung realisiert. Agenten reagieren auf Nachrichten mit der Ausführung von Aktionen bzw. senden selbst Nachrichten an andere Agenten.

Agenten suchen Zur Durchführung eigener Aktionen benötigen Agenten bestimmte Dienstleistungen anderer Agenten. Hierzu muss die Möglichkeit der initialen Kontaktaufnahme bereitgestellt werden.

Reproduktion Ist ein Agent erfolgreich und hat ausreichend Geld angesammelt, wird Reproduktion durchgeführt. Hierzu erfolgt zunächst die Suche (`Agenten suchen`) eines geeigneten Partners (sofern vorhanden) und anschließend die Erzeugung des Kindagenten (`Agent erzeugen`).

Agent erzeugen Sowohl initial und während der Laufzeit können Agenten manuell erzeugt werden (`Steuerung`). Auch während der Reproduktion (`Reproduktion`) erzeugen Agenten weitere Agenten

Geld verwalten Zur Durchführung aller evolutionären und geschäftlichen Prozesse ist die Ressource Geld notwendig. Es ist daher essentiell, Geldobjekte entsprechend verwalten zu können.

Steuerung Die Benutzerschnittstelle erlaubt eine bestimmte Kontrolle über das System. Während die direkte Steuerung von Agenten nicht immer gewährleistet ist, ist zumindest die manuelle Erzeugung von Agenten (`Agent erzeugen`) möglich. Weiterhin kann über die Steuerung z.B. die Höhe der Kosten für einzelne Aktionen festgelegt werden und auch die Erfassung, Verarbeitung und Speicherung von Log-Informationen ist Teil dieses Anwendungsfalles.

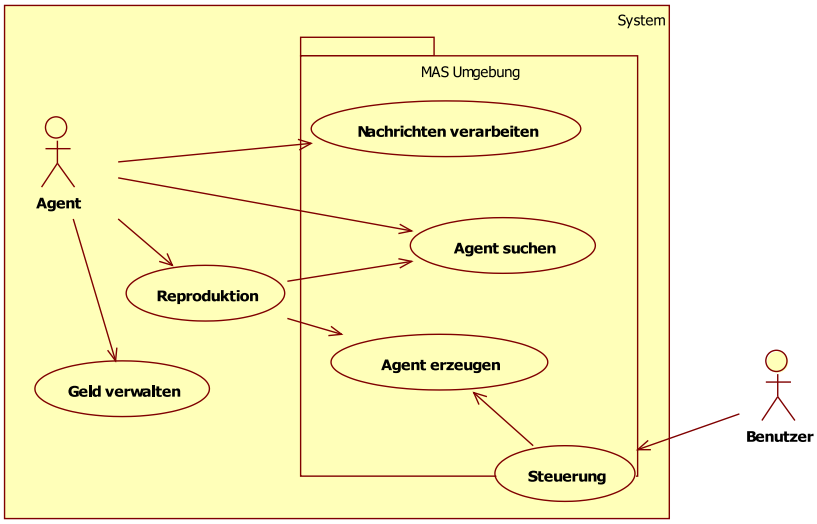


Abbildung 4.2: Wesentliche Use-Cases evolutionärer Agenten

Alle 'höheren' Funktionen, wie z.B. Suchalgorithmen oder Verhandlungen lassen sich auf oben genannte Anwendungsfälle reduzieren und sind daher nicht Teil der Basisfunktionalität. Aus den Anwendungsfällen lassen sich die wesentlichen Komponenten ableiten: Agenten, Umgebung und Steuerungskomponente für den Benutzer. Abbildung 4.3 zeigt diese Komponenten.

Der Agent zeichnet verantwortlich für die Umsetzung des Agentenmodells der Abschnitte 3.1.2, 3.1.6.4, 3.2.2. Hierbei muss zwingend die Funktionalitäten der Reproduktion und der Geldverwaltung erfolgen. Zusätzlich sind grundlegende agentenbasierte Verhaltensweisen abzudecken, sofern das genutzte MAS-Framework diese nicht bereitstellt. Hierzu zählen z.B. der Zugriff auf Nachrichtenfunktionalitäten, Yellow-Page Zugriff sofern benötigt oder auch die Erzeugung weiterer Agenten.

Die Agentenumgebung (`MAS Umgebung`) bildet das Konzept der Agentenplattform ab. Das Standardmodell einer Agentenplatform wurde von der *FIPA* (Foundation for Intelligent Physical Agents[3]) bereits 2002 spezifiziert und für Implementierungen als Basis zur Interoperabilität vorgeschlagen [25]. Durch Nutzung der Agentenplattform *JDADE* können viele der spezifizierten Eigenschaften direkt verwendet werden. Die in der abstrakten FIPA Architektur geforderten Bestandteile decken z.B. Use-Cases Nachrichten verarbeiten und Agenten suchen ab.

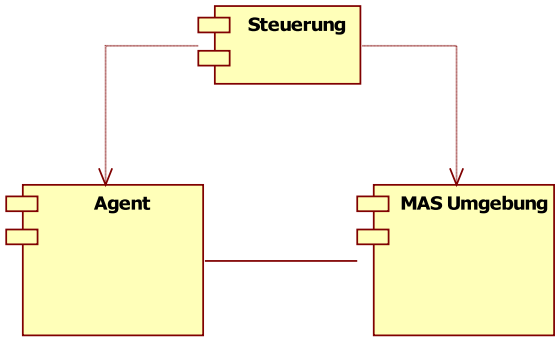


Abbildung 4.3: Komponenten des MAS

Die Steuerung erlaubt dem Nutzer Eingriffe in das System sowohl zur Laufzeit als auch vor dem Start. Hier sind z.B. Kosten für Aktionen hinterlegt und können verändert werden. Weiterhin bietet sich dem Benutzer die Möglichkeit, erfasste Daten in Echtzeit einzusehen und im begrenzten Rahmen zu bearbeiten.

Die drei in Abbildung 4.3 gezeigten Komponenten sind als JAVA Package strukturiert. Packages enthalten thematisch bzw. funktionell zusammengehörende Klassen und können rekursiv wiederum Packages enthalten. Wesentliche Aspekte der drei Komponenten werden in den nachfolgenden drei Abschnitten beschrieben.

4.1.2 Agent

Der Agent ist aus Implementierungsperspektive ein autonomer Berechnungsprozess mit Kommunikationsfähigkeit. Die Kommunikationsfähigkeit sollte im Sinne der FIPA *ACL* (Agent Communication Language [26]) standardisiert sein, um Interoperabilität unter verschiedenen Plattformen sicherzustellen. Diese Basisaspekte sind bereits durch Nutzung von Agentenframeworks großteils abgedeckt und beschrieben. Dieser Abschnitt konzentriert sich daher auf die Spezialisierung vom Agenten des Agentenframeworks hin zum evolutionären Agenten.

[3] siehe `http://www.fipa.org`

Abbildung 4.4 zeigt agentenspezifische Klassen und deren Abhängigkeiten innerhalb des Systems. Hierbei stellt die Klasse `Agent` die Basisklasse dar, aus welcher weitere Spezialisierungen abgeleitet sind. Im Wesentlichen stellt die Klasse `Agent` das in Definition 11 formalisierte Konzept dar:

- *Akt* wird durch Nachrichtenversand bzw. Methodenaufrufe abgebildet,
- *Sen* wird durch Nachrichtenempfang repräsentiert,
- Z ist der aktuelle Zustand gebildet aus allen lokalen Variablen zu einem bestimmten Zeitpunkt,
- φ wird durch die Summe alle Anweisungen innerhalb des Berechnungsprozesses des Agenten gebildet und
- s ist die leere Menge $\emptyset$

Weiterhin ist das Agentenkonzept so ausgelegt, dass Agenten - implementationstechnisch bedingt - untereinander nicht über Methodenaufrufe kommunizieren können. Zur eindeutigen Adressierung erhält jede Instanz von `Agent` beim Starten eine eindeutige `id` zugewiesen, die zur Kommunikation mittels Nachrichten verwendet wird. Ebenso kann jeder Agent Daten in die für Monitoringzwecke vorgesehene Komponente `JStreaMon` geben.

Durch die von `Agent` abgeleitete Klasse `EconomicAgent` erhält der Agent Fähigkeiten zur Geldverwaltung. Die Variable `funds` repräsentiert das dem Agenten a zur Verfügung stehende Kapital $funds(a)$. Die beiden Methoden `deposit` und `withdraw` bilden Zahlungseingänge und -abbuchungen ab. `getFunds` liefert den aktuellen Kontostand $funds(a)$. Ein Einsatz von `EconomicAgent` in ökonomischen Anwendungen ist hiermit bereits möglich. Durch entsprechende Ausgestaltung von φ, z.B. in Form von Verhandlungsprotokollen, ist eine Teilnahme in elektronischen Marktplätzen denkbar.

Gemäß der Formalisierung erfolgt eine weitere Spezialisierung von `EconomicAgent` in Form des `EvolutionaryAgent`. Hiermit werden die Konzepte der Strategie s_a und die Reproduktionsfähigkeit (Algorithmus 1) von Agenten realisiert. Dazu besitzt jeder Agent Attribute `GeneticCode` zur Speicherung der eigenen Strategie s_a, der evtl. vorhandenen Partnerstrategie sowie der externen lokalen Fitness zur Partnerstrategie. Die Ausführung der Funktionalität (Algorithmus 1) eines Agenten erfolgt in einer Endlosschleife, die in einem eigenen Thread läuft. Nähere Einzelheiten zu den unterschiedlichen Agentensystemen beschreibt Abschnitt 4.1.3.

Aufgrund des gewählten Ansatzes der lokalen evolutionären Optimierung konnte keines der gängigen evolutionären Frameworks, wie z.B. [154, 226] verwendet werden. Diese sind auf populationsorientierte Evolution ausgerichtet und die Operatoren arbeiten daher auf Populationsebene. Für den vorliegenden Ansatz stellt dies kein adäquates Implementationsziel dar und es musste eine Neuentwicklung begonnen werden.

Das evolutionäre Framework betrachtet als Mittelpunkt die genetische Information. Diese wird in Form des `GeneticCode` Objektes repräsentiert. Abbildung 4.5 zeigt schematisch ein Agentensystem, bestehend aus drei Lageragenten. Jeder Lageragent besitzt eine Strategie mit den Werten für Lagergröße, Mindestbestand und Mutationsrate. Für einen Agent ist zusätzlich ein einfaches Objekt Diagramm abgebildet. Dies stellt den aktuellen Zustand dar. Er besitzt eine `strategy`, eine `partnerStrategy`, das Geldvermögen `funds` beträgt 9, die `partnerFitness` (externe lokale Fitness für die Partnerstrategie) beträgt 1 und der `Bestand`

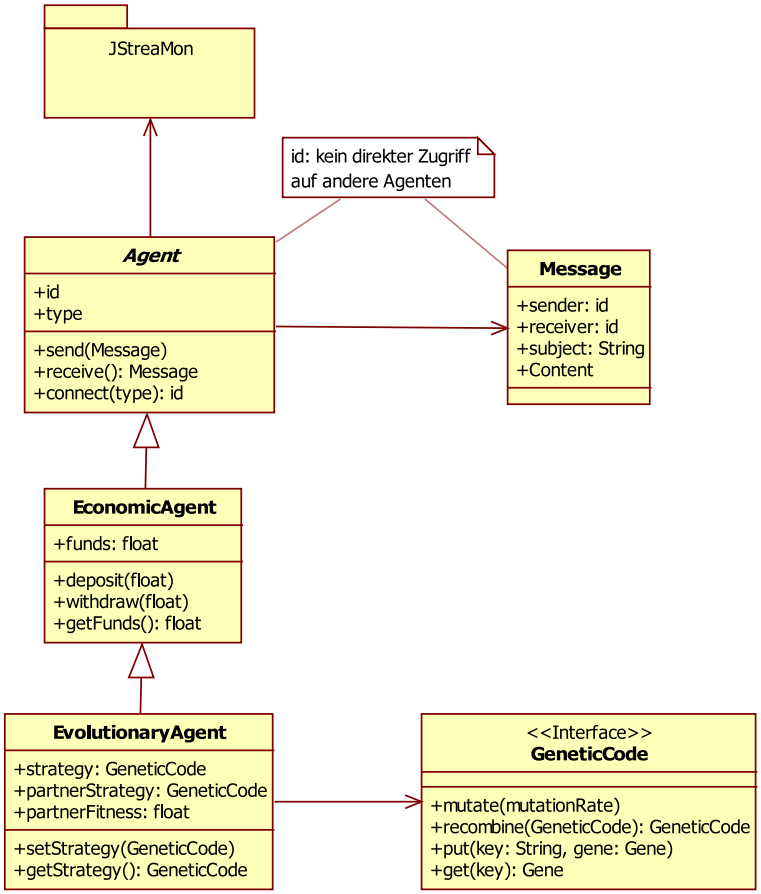


Abbildung 4.4: Agentenspezifische Klassen und Abhängigkeiten

beträgt aktuell 34. Realisiert ist `GeneticCode` als Container für *Schlüssel-Wert* Paare, wie in Abbildung 4.5 deutlich gemacht. Schlüssel ist ein String und den Wert repräsentiert ein `Gene` Objekt. Demzufolge besteht die Beispielstrategie der Agenten aus den Genen `Lagergrösse`, `Mindestbestand` und `Mutationsrate`.

Ein `GeneticCode` Objekt besitzt die Methoden `mutate(mutationsRate)` und `recombine(GeneticCode)`, die lokal Mutation und Rekombination realisieren. Bei der Rekombination wird zunächst die Methode `recombine(GeneticCode)` auf dem eigenen Strategieobjekt mit der Partnerstrategie als Parameter aufgerufen. Als Resultat erhält der Aufrufer ein neues Strategieobjekt, welches eine Kombination aller Parameter der Elternobjekte enthält. Danach führt ein Aufruf von `mutate(mutationsRate)` zur Mutation des Strategieobjektes. Dieses wird daraufhin dem Kindagent als Parameter mitgegeben. Für die unterschiedlichen Datentypen der Gene existieren unterschiedliche Gen-Implementierungen (z.B. `FloatGene`, `IntegerGene`, etc.). Demgegenüber stehen die `GeneOperator`-Klassen zur Realisierung von Mutation und Selektion auf `Gene` Ebene. Hierdurch lassen sich sehr einfach neue `Gene`-Klassen mit neuen Datentypen erstellen und deren Operatoren ebenso. Aktuell ist z.B. `FloatGene` als *Breeder Genetic Algorithm* (*bga*) [168, 169] implementiert. Wird auf ein `GeneticCode`-Objekt die Methode `mutate` aufgerufen, so wird dieser Aufruf für jedes `Gene` Objekt mit dem

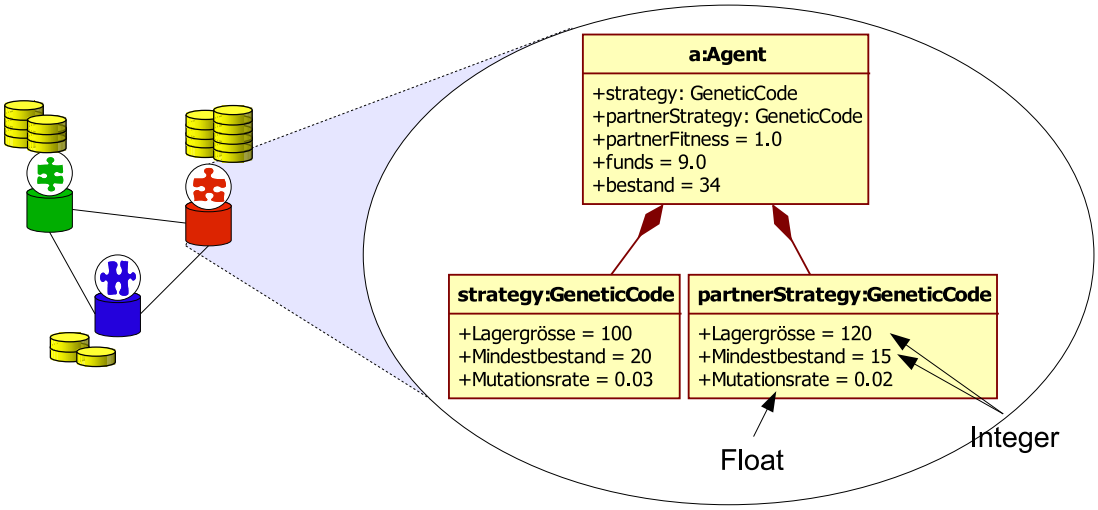


Abbildung 4.5: Agentensystem, bestehend aus drei Lageragenten, und Objektdiagramm eines Agenten mit Strategie und Geld

entsprechenden Operator wiederholt. Ein Klassendiagramm der evolutionären Klassen zeigt Abbildung D.2 im Anhang auf Seite 170.

Beim Ablauf der lokalen Selektion (siehe Abbildung 4.6) wird zunächst proaktiv eine sogenannte *Strategienachricht* verschickt (Schritt 2). Zur Identifizierung der Nachricht enthält sie den Betreff `MESSAGE_REQUEST_STRATEGY`. Empfängt ein anderer Agent eine Strategienachricht, so antwortet er mit einer *Strategieantwortnachricht* (Betreff `MESSAGE_ANSWER_STRATEGY`). Dieser Nachricht sind die Strategie und die externe lokale Fitness angehängt. Der Sender der Strategienachricht kann nun die bislang gespeicherte externe lokale Fitness mit der neuen vergleichen und gegebenenfalls die empfangene Strategie für spätere Reproduktion speichern.

Unabhängig von der lokalen Selektion erfolgt die Reproduktion, sobald $funds > \theta$. Hierbei ruft der Agent einfach ein `childStrategy = strategy.recombine(partnerStrategy)` auf und erhält damit eine Kindstrategie. Der Aufruf `childStrategy.mutate( strategy.get(Mutationrate))` mutiert diese Strategie mit der aktuellen Mutationsrate. Dieses nun vollständig erzeugte GeneticCode Objekt wird einem neuen Agenten als Strategie mitgegeben `childAgent = new Agent(childStrategy)`. Zum Schluss der Reproduktion wird noch die Hälfte des Vermögens an `childAgent` übermittelt.

`EvolutionaryAgent` stellt die Basisklasse für anwendungsspezifische Agenten dar. Diese werden in den nachfolgenden Abschnitten der Experimente näher beschrieben.

4.1.3 Agentensysteme

Die *FIPA* Spezifikationen sind in verschiedene Bereiche eingeteilt. Diese beschäftigen sich neben der Spezifikation von Agenten mit einer Reihe von Themen, wie in Abbildung 4.7 gezeigt. Ziel dieser Spezifikationen ist Interoperabilität über verschiedene Systeme hinweg und Wiederverwendbarkeit. Dieser Abschnitt geht kurz auf die einzelnen Bestandteile der Spezifikationen ein und stellt anschließend die genutzten Agentensysteme kurz vor.

Der primäre Fokus der *FIPA Abstract Architecture* [25] ist die Schaffung von Interoperabilität zwischen Agenten mit unterschiedlichen Nachrichtentransportmechanismen, Agentensprachen oder Inhaltsbeschreibungssprachen. Eine Agentenkommunikationssprache (ACL

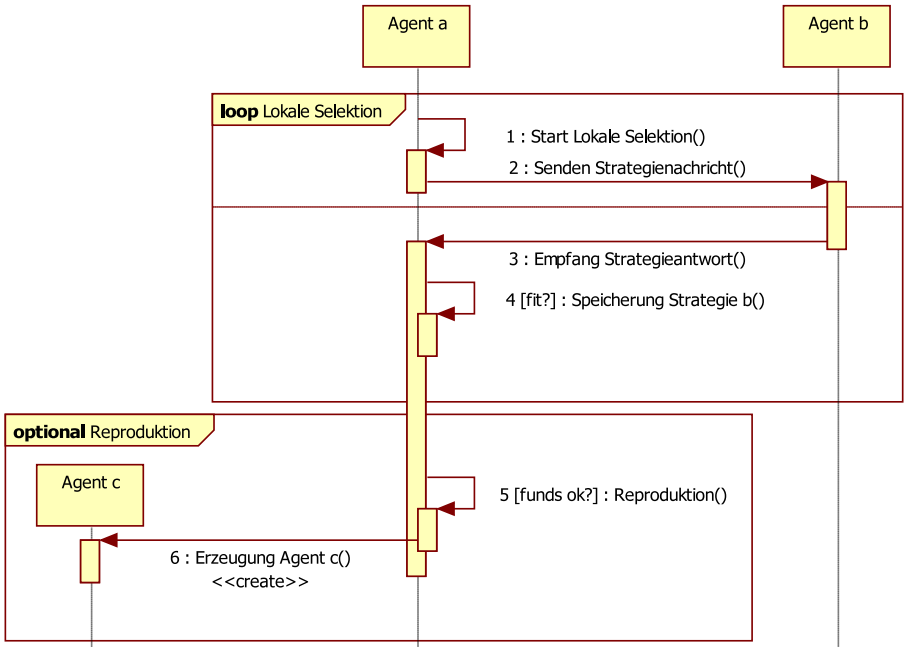


Abbildung 4.6: Sequenzdiagramm zum Ablauf der lokalen Selektion und Reproduktion

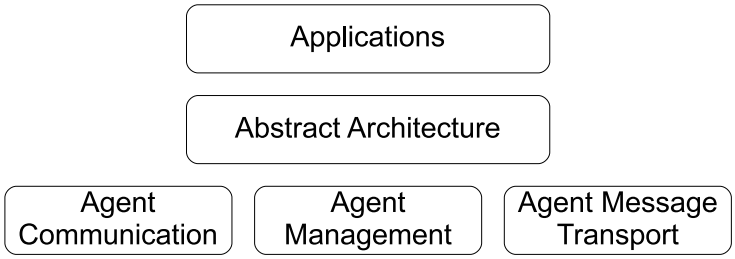


Abbildung 4.7: FIPA Spezifikationensbereiche im Agentenumfeld (Quelle: www.fipa.org)

[24]) drückt Kommunikationsakte aus und erlaubt hierdurch kompatiblen Nachrichtenaustausch. Elemente hierfür sind zum Beispiel Anfragen (engl. Request), deren Intention darin besteht, den Empfänger eine bestimmte Aktion durchführen zu lassen. Die *Agenten Management Spezifikation* [28] stellt ein normatives Framework, in welchem FIPA-konforme Agenten arbeiten. Bestandteile hiervon sind Nachrichtentransport, Anbindung der Agenten und Serviceverzeichnisse über multiple Plattformen. Schließlich existiert die *Agent Message Transport Specification* [27]. Das FIPA Referenzmodell zeichnet verantwortlich für das Nachrichtentransportprotokoll, den Nachrichtentransportservice und die Agentenkommunikationssprache.

Bezogen auf das formale Modell in Abschnitt 3.1 beschreibt eine Agentenplattform die Umgebung U und das Serviceverzeichnis einen Teil der Sichtbarkeit vis. Hierbei wird im Agentenkontext von sogenannten *Containern* gesprochen. Ein Container entspricht in dieser Arbeit einer virtuellen Agentenumgebung v. Die Struktur aller Agentenplattformen bildet damit die Struktur G_V. Nachrichten sind eine Umsetzung des Konzepts der Wahrnehmungen Sen und Aktionen Akt auf Agentenebene. Ein Agentenmanagementsystem stellt sicher, dass

sich jeder Agent in maximal einem Container (=Umgebung) befindet, wobei Container = v, Zusammenschluss der Container = V. Formale Objekte werden repräsentiert durch Klasseninstanzen bzw. beliebige Informationen innerhalb der Umgebung und die Sichtbarkeit ergibt sich implizit über die Struktur G_V und die Verteilung bzw. Vernetzung der Agenten innerhalb G_V.

Zur Umsetzung evolutionärer Agenten wurden insgesamt drei unterschiedliche Agentenplattformen eingesetzt. Die Grund hierfür lag auf den jeweils verschiedenen Designperspektiven der einzelnen Systeme. Nachfolgend finden sich die unterschiedlichen Designaspekte und ihre Konsequenzen im Kontext der Anwendung und FIPA Konformität.

JADE

Das *JADE* Framework ist ein FIPA konformes Agentensystem. Die Plattform kann auf mehrere Maschinen verteilt werden. Dank Java läuft JADE auf einer Vielzahl von Betriebssystemen einschließlich mobiler Endgeräte. Die Kommunikation erfolgt mittels ACL-Nachrichten, die über den Nachrichtendienst an jeden Agent zugestellt werden und in einer privaten Eingangswarteschlange bis zur Verarbeitung liegen. Das volle FIPA Kommunikationsmodell ist verfügbar und so liegen bereits eine ganze Reihe agentenspezifischer Protokolle vor, die 'out of the box' nutzbar sind. Der Agent Management Service und das Servicerepository sind selbst als Agent realisiert. Jeder Agent läuft in einem eigenen Thread. Dies bedeutet, dass für jeden Agenten im System ein Thread zur Verfügung stehen muss. Innerhalb eines Agenten können unterschiedliche Verhaltensweisen, sogenannte `Behaviours` installiert werden. Diese reagieren auf Ereignisse, wie z.B. Nachrichtenempfang bestimmter Nachrichten. Mit Hilfe dieser `Behaviours` ist die Realisierung asynchroner Verhaltensweisen, wie lokale Selektion, Reproduktion, Kauf und Verkauf von Gütern effizient und leicht zu implementieren. Mobilität von Agenten wird unterstützt.

Insgesamt legt JADE den Schwerpunkt auf Interoperabilität, nicht zuletzt dank kommerzieller Anwender in der Industrie[4]. Die Konformität und der daraus resultierende Einsatz standardisierter Sprachen, Ontologien und Protokolle produzieren für Anwendungen, die nicht darauf angewiesen sind, teilweise Overhead. Gleiches gilt für das Threading Verhalten. Daher ist die Anzahl der aktiven Agenten in realistisch nutzbaren Systemen beschränkt auf einige Dutzend bis wenige Hundert. Gleichwohl erlaubt JADE nach kurzer Einarbeitungszeit schnelles und effizientes Entwickeln.

DIET

Um die Beschränkungen des ressourcenhungrigen und schwergewichtigen JADE zu umgehen wurde als Alternative die DIET Multiagentenplattform [151] verwendet. Hier liegt der Fokus auf einer leichtgewichtigen Umsetzung des Agentenparadigmas. Ein Agent läuft nicht in seinem eigenen Thread, sondern es existiert ein Threadpool. Agenten werden den Threads zugewiesen und laufen bis keine Aktivität mehr vorliegt bzw. die Kontrolle explizit abgegeben wird. Die gesamte Kommunikationsarchitektur basiert auf dem Peer-to-Peer Ansatz, so dass Nachrichten effizient und schnell zwischen den beteiligten Agenten ausgetauscht werden. Wichtig ist in diesem Zusammenhang das *fail-fast Prinzip*: stehen nicht genügend Ressourcen des Betriebssystems zur Verfügung, so werden z.B. Nachrichten verworfen. Im Falle

[4] z.B. Whitestein Technologies `http://www.whitestein.com/`

eines Kommunikationsrotokolls, welches für jede empfangene Nachricht zwei neue Nachrichten generiert, ist klar, dass hierdurch schnell jedes System in die Knie gezwungen werden kann und damit einfriert oder ganz abstürzt. Fail-fast realisiert eine natürliche Ressourcenbeschränkung. Weiterhin ist der Verzeichnisservice so ausgelegt, dass lediglich ein zufälliger Agent des gewünschten Services zurückgeliefert wird. Mit DIET können ebenso wie bei JADE verschiedene Container auf unterschiedlichen Rechnern gestartet werden. Durch den leichtgewichtigen Ansatz sind bis zu einige hunderttausend Agenten möglich. Mobilität wird unterstützt.

Insgesamt ist die Intention von DIET sehr auf den organischen Ansatz ausgelegt. Die Skalierung wird durch leichtgewichtige Umsetzung der Agentenprinzipien erreicht. Allerdings wird keine FIPA Konformität umgesetzt. Anstelle schwergewichtiger, intelligenter Agenten liegt hier der Schwerpunkt eher auf kleinen Agenten, die in großer Zahl durch Interaktion mit Hilfe einfacher Regeln Probleme lösen. Das DIET Projekt endete offiziell im Jahre 2003 und die letzte Version datiert auf 2005, so das die Community mittlerweile nicht mehr aktiv ist. Es erfolgt derzeit keine Weiterentwicklung der Software.

EAT

JADE und DIET realisieren die Ausführung des Agentenprozesses jeweils mittels Threads. Hierdurch entsteht eine gegenseitige Abhängigkeit von Agentensystem und Betriebssystem. Sowohl der betriebssystemabhängige Schedulingalgorithmus, als auch andere System-, Userprozesse und auch Agentenprozesse selbst beeinflussen damit die Ausführung der Agentenprozesse. Besonders für zeitabhängige Zahlungen, wie in den Kapiteln Kosten 3.1.4 und Steuern 3.1.6.3 dargestellt, kann dies zu unerwünschten Effekten führen. Als Beispiel sei ein Lageragent a angeführt, dessen Kosten pro Sekunde abgerechnet werden. Hier führt hohe Systemlast (z.B. verursacht durch andere Agenten oder sonstige Prozesse) zu erhöhten Kosten für a, da die zeitliche Abfolge von Kauf- und Verkaufstransaktionen stark verzögert ist. Damit verbleiben Güter länger im Lager und führen zu Verzerrungen auf der Kostenseite, die auf das Gesamtsystem Einfluss haben.

Daher wurde der Ansatz auf ein einfaches, getaktetes Agentensystem übertragen: *Evolutionäre Agenten geTaktet* (*EAT*). Diese Eigenentwicklung stellt geforderten Mechanismen, wie Agentenmanagement und Nachrichtentransport bereit. Die Agentenarchitektur wurde wie in Kapitel 4.1.2 beibehalten. Parallelität wurde durch die zufällige Ausführung der einzelnen Agentenalgorithmen in jedem Takt gewährleistet, so dass langfristig zeitliche Abhängigkeiten ausgeschlossen werden können. Die Taktung ermöglicht Unabhängigkeit von zeitlichen und systemischen Begrenzungen durch taktgenaue Abrechnung.

Erste Implementierungen der Prototypen erfolgten in JADE. um größere Populationen von Agenten verwalten zu können, wurde DIET genutzt. DIET und JADE arbeiten mit Threads zur Agentenausführung und daher werden die Agententhreads vom Betriebssystem beeinflusst. Aus diesem Grund wurde die getaktete Eigenentwicklung EAT erstellt, um unabhängig von Laufzeiten genaue ökonomische Abrechnungsverfahren umzusetzen.

4.1.4 Monitoring - JStreaMon

Datenerfassung und Aufbereitung in Multiagentensystemen erfordert spezielle Anforderungen an eine Monitoring Komponente. Zunächst ist eine asynchrone und nicht blockierende Datenerfassung über unterschiedliche Prozesse hinweg notwendig. Ebenso spielte beim Design die

Erfassung und Verarbeitung in Echtzeit eine wichtige Rolle, um während der Simulationsläufe bereits Entwicklungen zu verfolgen. Um Daten von einigen dutzend, hunderten oder tausenden Agenten sinnvoll zu erfassen und aufzubereiten, muss jedes Datum folgende Information beinhalten:

Datenquelle: Jede Datenquelle (Agent) benötigt dazu eine eindeutige `id` (Identität). Dies ist durch FIPA Konformität gegeben.

Datenstrom: Jede Datenquelle kann beliebig viele Datenströme generieren. Am Beispiel der Lageragenten in Abbildung 4.5 sind sinnvolle zeitlich veränderliche Datenströme beispielsweise `funds` und `bestand`. Konstante Daten sind z.B. Parameter der Strategie, wie `Lagergrösse`, `Mindestbestand` und `Mutationsrate`.

Gruppierung: Heterogene Agentensysteme bestehen aus mehreren Typen von Agenten, z.B. Lageragenten, Produktionsagenten, etc.. Hierbei ist die Zuordnung eines Datums zu einer Gruppe sinnvoll für spätere Auswertungen.

Die Speicherung der Daten erfolgt in einem zentralen Datenmodell und wird über Views dargestellt. Abbildung 4.8 verdeutlicht das angewandte *Modell View Controller* (*MVC*) Muster. Ein ausführliches Klassendiagramm D.1 befindet sich im Anhang. Die Daten werden von den Controllern erzeugt und über verschiedene Datenströme im Modell gespeichert. Views holen entsprechend ihrer Konfiguration die Daten aus dem Modell und bereiten diese in Echtzeit für den Benutzer auf.

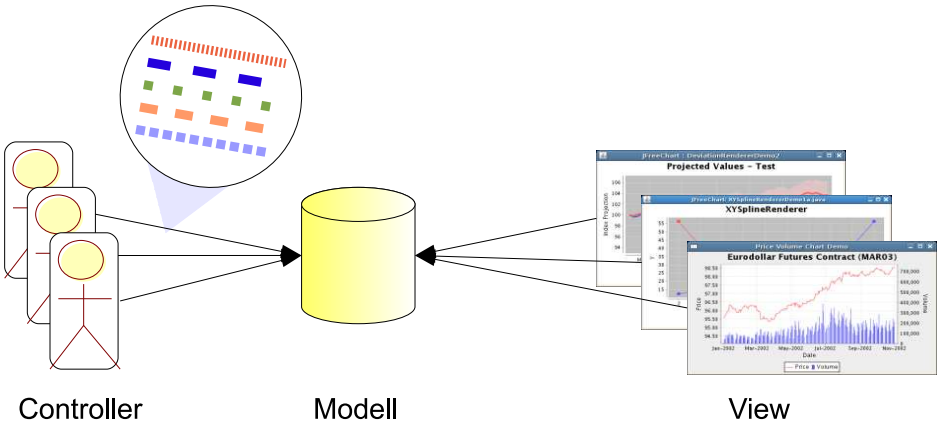


Abbildung 4.8: JStreaMon Modell-View-Controller Muster. Die unterschiedlichen Datenströme eines Agenten sind schematisch vergrößert dargestellt.

Zur Speicherung der Daten im Modell existieren die zwei Methoden `setValue(source, group, stream, value)` und `addValue(source, group, stream, value)`. Der Unterschied besteht darin, dass `setValue` den Wert einfach speichert und einen evtl. vorhandenen älteren Wert überschreibt, während `addValue` den neuen Wert zum vorhandenen Wert hinzufügt. `addValue` ist dann sinnvoll, wenn Agenten dem Modell die Aufsummierung von Daten über einen längeren Zeitraum überlassen.

Zur Datenaufbereitung stellt das Modell eine Vielzahl von Operationen bereit. Dazu zählt der Abruf einzelner Daten oder aggregierte Informationen zu Datenströmen. So liefert

`getSum(lagergroesse)` unter Angabe des Datenstroms `lagergroesse` die Summe der Lagergröße aller Agenten. Neben der Einschränkung über Datenströme lässt sich darüber hinaus nach Gruppierung filtern. Die Datenaufbereitung umfasst neben der Summe die Operatoren Maximum, Minimum und Durchschnitt.

Zur grafischen Aufbereitung stehen folgende Views zur Verfügung: Balkendiagramm, Histogramm, Liniendiagramm und Fileview. Jeder View kann über ein grafisches Interface einfach zur Anzeige von Datenströmen, -gruppen, Operatoren und Filtern konfiguriert werden. Durch JStreaMon lassen sich synchrone und asynchrone Datenströme in Echtzeit geeignet visualisieren. Damit ist für Agentensysteme eine Echtzeitvisualisierung möglich.

4.2 Adaptive Logistiknetzwerke

Dieser Abschnitt beschreibt eine exemplarische Anwendung des evolutionären Agentenansatzes zur Anpassung und Optimierung einer *Wertschöpfungskette* (engl. *Supply-Chain*) wie in Abbildung 4.9 dargestellt. Hierbei wird zunächst eine Parallele zwischen komplexen adaptiven Systemen (*CAS* siehe Abschnitt 2.1.6) und Wertschöpfungsnetzwerken in Abschnitt 4.2.1 hergestellt. Anschließend erfolgt die Problembeschreibung in Abschnitt 4.2.1 und der experimentelle Aufbau (Abschnitt 4.2.2). Zum Schluss erfolgt die Präsentation der Ergebnisse in Abschnitt 4.2.3.

4.2.1 Szenario

Als Beispiel eines CAS wird ein Wertschöpfungsnetzwerk verwendet. Hierzu sind zunächst CAS-spezifische Charakteristika wie in Abbildung 4.9 zu evaluieren. Logistische Systeme bestehen meist aus einer großen Anzahl unterschiedlicher Akteure. Im Kontext der vorliegenden Arbeit werden alle Akteure als Agenten bezeichnet. Alle Agenten sind Teil unterschiedlicher Organisationen. Hierzu sei eine Menge von Standorten eines Produzenten angeführt. Jede Produktionsstätte agiert autonom in Sinne lokaler Entscheidungen und ist gleichzeitig auch Teil einer größeren Organisation. Ebenso ist die Flotte von Spediteuren zu betrachten. Ein einzelner Transporter agiert selbständig im Sinne lokaler Entscheidungen z.B. im Straßenverkehr und ist gleichzeitig jedoch in Entscheidungsprozesse auf Organisationsebene eingebunden (z.B. Zielfestlegungen). Damit lassen sich unterschiedliche Entscheidungsebenen innerhalb einer Supply-Chain wie z.B. einzelne Akteure, deren Organisationen und auf globaler Ebene die Supply-Chain darstellen. Auf jeder Ebene finden Anpassungsprozesse statt. Ein Beispiel hierfür sind veränderliche Preise als Ergebnisse von Verhandlungen zwischen den Akteuren und Organisationen. Die Untersuchung eines dezentralen Koordinationsmechanismus am Beispiel der Produktion von Tischen findet sich in Eymann [64]. Auf Ebene der Agenten finden sich Transportrouten im Falle von Transportagenten oder Verträge zwischen Organisationen, die von den Entscheidungen einzelner Agenten abhängen und damit zeitlich veränderlich sind. Wesentlich ist im logistischen Kontext ebenso die Anpassung der Struktur und Dimension von Transportnetzwerken, die sogenannte *Netzstrukturplanung* [95]. Als Resultat entsteht Feedback aus all den lokalen Interaktionen zwischen Agenten und ihren Organisationen. Profit ist ein interessantes Beispiel von Feedback in ökonomischen Dimensionen oder auch strategischer Einkauf und Verkauf von Gütern in Folge von zuvor geschlossenen Verträgen. Schließlich ergeben sich auf globaler Ebene aus den genannten Eigenschaften emergente Muster. Hierzu zählen z.B. veränderliche Märkte im Zuge von Innovationen oder Nachfrageveränderungen aufgrund schwankender Nutzerpräferenzen. Die Auswirkungen verbreiten sich innerhalb des Netzwerkes

und als Folge finden Anpassungsprozesse auf allen Ebenen des Wertschöpfungssystems statt. Beispiel von Nachfrageveränderungen ist der Absatzeinbruch bei Automobilfirmen und deren Zulieferern im Jahre 2008/2009[5]. Ein anderes Beispiel ist das Aufschaukeln von Nachfrageschwankungen entlang einer Supply-Chain, auch als Bullwhip-Effekt [140] bekannt.

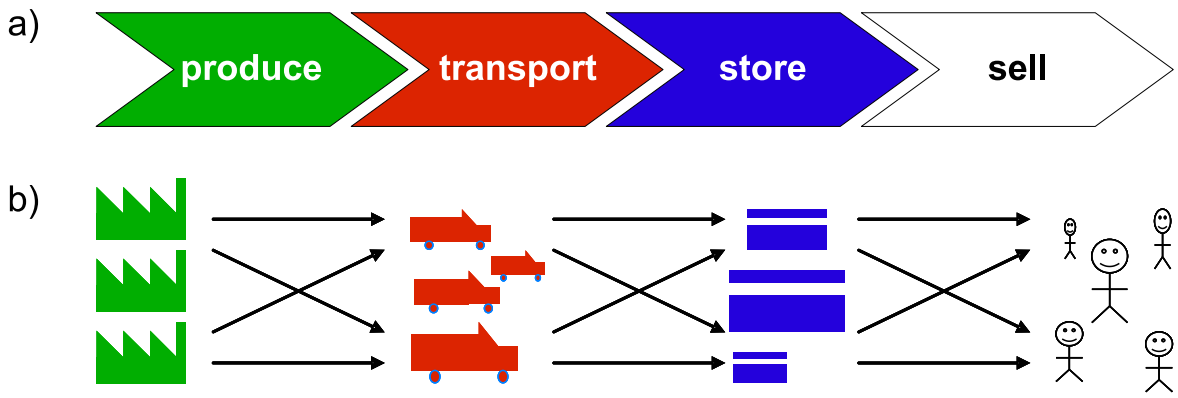


Abbildung 4.9: a) Distributionsprozess einer Wertschöpfungskette vom Produzenten zum Konsumenten, b) Schematische Struktur des Distributionsnetzwerkes

Logistische Prozesse nach Pfohl [191] sind Netzwerke in denen Objekte (Material, Informationen, Geld, Güter, etc.) fließen. Der Fluss in logistischen Netzwerken wird als Bewegung der Objekte entlang der Kanten und (vorübergehende) Speicherung bzw. Transformation in den Knoten aufgefasst.

Entsprechend dem Prozess in Abbildung 4.9a) fließen Güter über vier Schritte durch das Netzwerk. Im ersten Schritt, der Produktion, erfolgt eine Transformation von Basismaterialen (nicht betrachtet) in die gewünschten homogenen Produkte. Im zweiten Schritt erfolgt der Transport in die Zwischenlager, gefolgt von der Lagerung. Im vierten Schritt findet der Verkauf an den Kunden statt. Jeder Agent repräsentiert einen Akteur (Produzent, Transporteur, Lager, Kunde) mit unterschiedlichen Parametern, die das Verhalten des Agenten und das Verhalten des gesamten Netzwerkes bilden. Dabei ist kein einzelner Agent in der Lage, das gesamte Netzwerk zu steuern bzw. zu kontrollieren.

Dieses Kapitel beschreibt die Anwendung evolutionärer Agenten zur Bestimmung von Transportnetzwerken. Hierbei steht generelle Anwendbarkeit des verteilten Ansatzes und die Evaluation im Vordergrund. Zunächst wird das Experiment in Abschnitt 4.2.2 beschrieben. Abschließend erfolgt die Auswertung der Ergebnisse in 4.2.3.

4.2.2 Experiment

Um die im vorangegangenen Abschnitt beschriebene Wertschöpfungskette zu realisieren, wurde das DIET Agentenframework [151] eingesetzt. Hiermit kann eine große Anzahl von Agenten simuliert werden. In DIET hat jeder Agent ein sogenanntes `familyTag`, welches die Zugehörigkeit zu einer bestimmten Gruppe von Agenten anzeigt. Entsprechend dem Prozess in 4.9a) gibt es vier Gruppen von Agenten: `ProducerAgent` (Produzenten), `TransportAgent` (Transporteure), `StorageAgent` (Lager) und `ConsumerAgent` (Kunden).

[5] siehe hierzu Financial Times Deutschland: Das Letzte: Hummer R.I.P., Ausgabe 10.06.2008; Insolvenz der Auto-Zulieferer, http://www.capital.de/unternehmen/100019108.html; GM prüft Abschied von Hummer [96]

Entsprechend dem bottom-up Design der Plattform ist es nicht möglich, nach konkreten Agenten zu suchen. Stattdessen stellt die Umgebung eine `connectMe(familyTag)` Methode bereit, die mit einem zufälligen Agenten mit dem jeweiligen `familyTag` verbindet. Dieser Mechanismus bildet einen Peer-to-Peer Service Discovery Mechanismus ab. Nach erfolgreicher Verbindung via `connectMe(familyTag)`, ist die Identität des Agenten bekannt und kann für zukünftige Kontakte verwendet werden. Jeder Agent besitzt zusätzlich zu seinem eigenen `familyTag` einen sogenannten `partnerTag`. Dieser verweist auf den benötigten Service anhand des gegebenen Prozesses. Damit können die Agenten in unterschiedlichen Rollen und beliebigen anderen seriellen Prozessen wiederverwendet werden. Entsprechend des abgebildeten Prozesses sind die `familyTag` Zuordnungen wie folgt:

- `ProducerAgent`: erzeugen Güter selbst (`familyTag`: produce → `partnerTag`: `null`)
- `TransportAgent`: transportieren Güter zu den `StoreAgenten` (`familyTag`: transport → `partnerTag`: `produce`)
- `StorageAgent`: lagern und verkaufen Güter (`familyTag`: store → `partnerTag`: `transport`)
- `ConsumerAgent`: kaufen Güter (`familyTag`: consume → `partnerTag`: `store`)

Zum Beispiel bekommt der `ConsumerAgent` den `store partnerTag`, um `StorageAgenten` über die Plattform kontaktieren zu können. Hat ein Agent den benötigten Service kontaktiert, so bleibt die Verbindung solange aktiv, bis einer der Handelspartner stirbt oder nicht mehr liefern kann. Zusätzlich gibt es ein sogenanntes `serviceSeekInterval`. Dieses gibt an, wann Agenten nach Services suchen. Wird ein Agent mit besserem Preis gefunden, so wird die aktive Handelsverbindung geschlossen und stattdessen die neue Verbindung genutzt. Das Intervall `strategySeekInterval` gibt die Zeit an, nach der Strategienachrichten versendet werden.

Die einzelnen Agenten haben je nach ihrer Funktion unterschiedliche Aufgaben. Der Produktionsagent produziert kontinuierlich nach `produceTime` ein Produkt, bis seine Lagerkapazität `produceStoreCapacity` erreicht ist: $t_{produceTime} + 1 = t_{produceTime} + produceTime$.

Der `StorageAgent` versucht, in seinem Lager immer anhand seiner Strategie Produkte vorrätig zu halten. Dazu gibt es zwei Strategieparameter: $s(storesize)$ und $s(minimumstock)$. Bei Unterschreiten von $s(minimumstock)$ ordert der `StorageAgent` neue Produkte bei einem `TransportAgent`. Kunden werden sofort bedient, sofern genügend Produkte im Lager vorrätig sind.

Der `TransportAgent` hat eine bestimmte Transportkapazität, die per Strategie festgelegt ist ($s(transportCapacity)$). Bestellungen, welche die Transportkapazität übersteigen, werden abgelehnt. Es werden also keine Teilbestellungen ausgeführt. Die Transportzeit wird für alle Transporte als konstant angesehen.

`ConsumerAgenten` bestellen bei den `StorageAgenten` in zufälligen Intervallen, um die Nachfrage nicht zu homogen zu gestalten. Das Bestellintervall wird aus einer Gleichverteilung $U[0, orderinterval]$ zufällig gewählt, die der `ConsumerAgent` wartet bis zur nächsten Bestellung: $t_{order} + 1 = t_{order} + U[0, orderinterval]$. Die Konsumeragenten werden als konstante Nachfrage angesehen und sind daher keine evolutionären Agenten.

Treten bei den Bestellungen zwischen den Agenten Fehlbestellungen auf, dann wird dies als sogenannte *missed order* protokolliert. Fehlbestellungen können auftreten, wenn z.B. ein Agent nicht genügend Produkte besitzt, oder nicht genügend Kapazität hat, um die Bestellung auszuführen.

Die Steuern, zu zahlen an die Umgebung, sind wie folgt zu berechnen:

$$\begin{aligned}\mathcal{T}_a(t) = &0.06 \cdot s(produceStoreCapacity) \\ &+ 0.06 \cdot s(transportCapacity) \\ &+ 0.06 \cdot s(storesize)\end{aligned} \tag{4.1}$$

Die Faktoren wurden empirisch festgelegt. Den Agenten jeden Typs jeweils nur die entsprechende Kapazität abgerechnet. Nachfolgend die verwendeten Konstanten und initialen Strategieparameter für die Simulationen aufgeführt:

- `serviceSeekInterval`: 30s. Nach Ablauf von `serviceSeekInterval`, ein Agent sucht erneut nach dem benötigten Service und wechselt ggf. den Anbieter
- `strategySeekInterval`: 15s. Nach Ablauf von `strategySeekInterval`, ein Agent versendet seine Strategienachricht und erhält andere Strategien. Diese werden ggf. als Partnerstrategie verwendet (siehe Abschnitt 4.1.2).
- `mutationProbability`: 0.03. Benötigt für `GeneticCode.mutate(mutationProbability)`.
- Anzahl Simulationen zur Durchschnittsbildung: 50
- Profit auf jeder Stufe: 30%
- Produktionsagent (`ProducerAgent`):
  - `produceTime`: 500ms
  - `produceStoreCapacity`: 50
  - $\theta_{ProducerAgent} = s(produceStoreCapacity) \cdot 2$
  - Anzahl initial: 50
- Transportagent (`TransportAgent`):
  - `transportCapacity`: 20
  - $\theta_{TransportAgent} = s(transportcapacity) \cdot 2$
  - Anzahl: 1
- Storageagent (`StorageAgent`):
  - `storesize`: 20
  - `minimumstock`: 20. Es wird sofort nachbestellt, wenn Produkte verkauft wurden.
  - Anzahl: 50
- Kundenagent (`ConsumerAgent`):
  - `orderinterval`: 10s
  - `ordersize`: 10
  - $\theta_{StorageAgent} = s(storesize) \cdot 2$

– Anzahl: 50

θ wurde rational auf das Doppelte der für eine Lagerfüllung notwendigen Kosten für alle Agententypen festgelegt (siehe hierzu jeweils die Einstellungen pro Agent). Die Evaluation und Untersuchung der Effekte erfolgt am Beispiel der Transportagenten. Von diesen befindet sich zu Beginn der Simulation jeweils ein Agent im System.

4.2.3 Ergebnisse

Es gelten die in Abschnitt 4.2.2 vorgestellten Einstellungen, solange nichts anderes angegeben wurde. Zunächst wird die Entwicklung der Transportkapazitäten aller Transportagenten betrachtet. Hierzu wird für eine Experimentreihe die Nachfrage konstant gehalten und eine weitere Experimentreihe erfolgt mit veränderlicher Nachfrage. Für die Experimentreihe mit veränderlicher Nachfrage werden zusätzlich die missed orders angezeigt. Anschließend erfolgt eine Untersuchung von θ. Hierzu werden die missed orders für unterschiedliche Werte von θ abgebildet, um den Einfluss der Reproduktionsschwelle auf das Systemverhalten zu dokumentieren. Darüber hinaus wird θ selbst als Variable in die Strategie aufgenommen und angepasst. Hierfür werden das Maximum, Minimum und der Durchschnitt innerhalb der Population aufgezeichnet. Es folgt die Betrachtung der Verteilung der Transportkapazitäten aller Transportagenten. Hierzu werden fünf Histogramme zu unterschiedlichen Zeitpunkten abgebildet.

Im Anhang finden sich weitere Ergebnisse, die nicht primär der Demonstration abgeleiteter Eigenschaften dienen. Hierzu zählt der Vergleich unterschiedlicher Selektionsmethoden und eine Initialisierung mit 50 anstelle nur einem Transportagent. Die durchgeführten Experimente sind im Einzelnen:

Kategorie	Experiment	Diagramm
Adaptivität Populationsgröße	Verlauf der summierten Transportkapazität aller Transportagenten	4.10
	Verlauf der summierten Transportkapazität mit Veränderung der Nachfrage	4.11
Adaptivität Strategie	Verlauf von *missed orders*	4.12
	Vergleich von *missed orders* für unterschiedliche Werte von θ	4.13
	Verlauf von variablem θ (Maximum, Durchschnitt, Minimum)	4.14
	Histogramm der Transportkapazitäten zu unterschiedlichen Zeitpunkten	4.15
Parametrisierung	Vergleich *missed orders* für unterschiedliche lokale Selektionsmethoden	B.1 (Anhang)
	Verlauf Transportkapazität für zwei Selektionsmethoden bei Initialisierung mit 50 Transportagenten	B.2 (Anhang)

Tabelle 4.1: Experimente zu adaptive Logistiknetzwerke

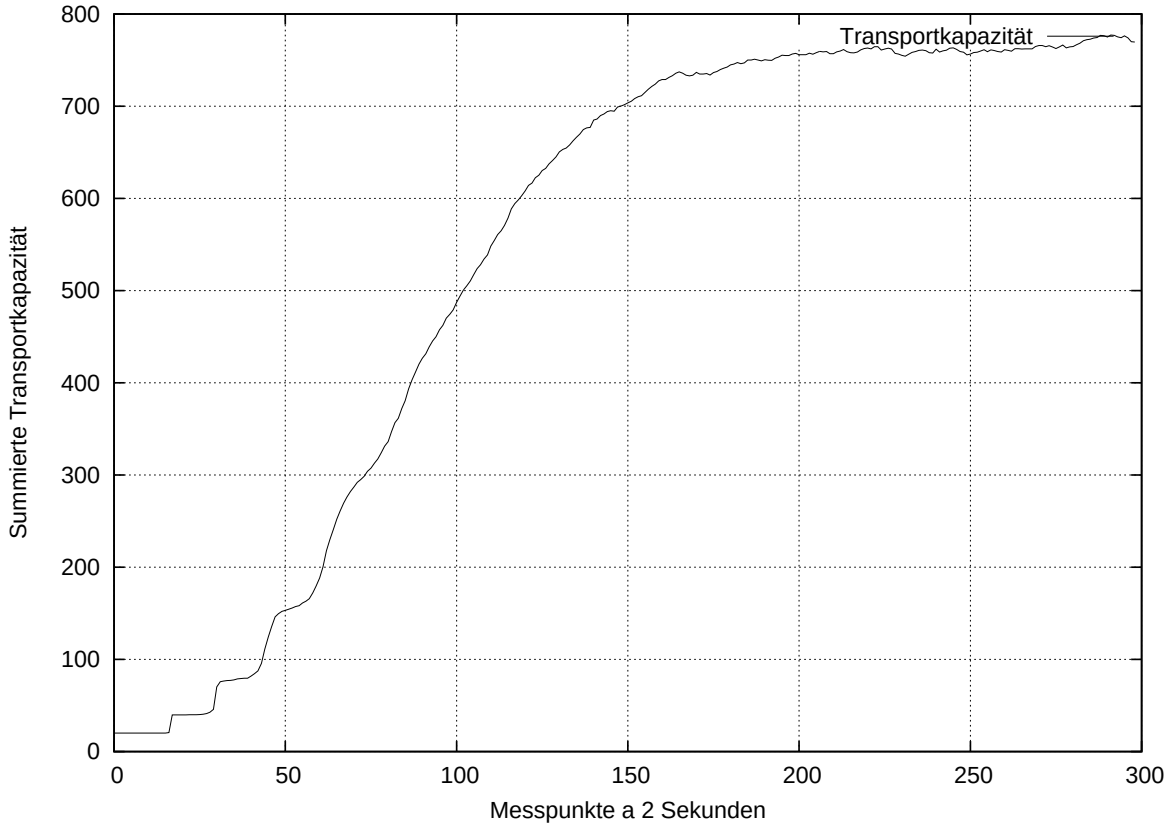


Abbildung 4.10: Summierte Transportkapazität über die gesamte Simulationszeit

Abbildung 4.10 zeigt die aufsummierte Transportkapazität aller Transportagenten über die gesamte Dauer des Simulationslaufes. Die Gesamtdauer beträgt 600 Sekunden, wobei alle 2 Sekunden die Summe aufgezeichnet wurde (insgesamt 300 Messpunkte). Deutlich ist zum Beginn der Messung das treppenförmige Ansteigen der Kapazität zu sehen. Hierbei erfolgte zu Beginn die Reproduktion zunächst nur durch Mutation, da lediglich ein Transportagent vorhanden war. Das stufenartige Ansteigen der Transportkapazität geht schnell über in ein gleichförmiges Ansteigen. Hierfür sind zum Einen Effekte der Mutation und später der Rekombination verantwortlich und zum Anderen die Durchschnittsbildung der Simulationsläufe. Der steile Anstieg bis zum Erreichen der Tragfähigkeit der Umgebung verläuft zügig von 20 bis auf ca. 750 und pendelt um diesen Wert. Insgesamt zeigt sich hier die Adaption auf Populationslevel als schnelles Ansteigen der Anzahl von Transportagenten und damit verbundenem Anstieg der Transportkapazität. Hier zeigt sich das 'Umschalten' der explorativen in eine exploitative Suche. Sobald die maximale Umgebungskapazität (in Form von Nachfrage) erreicht ist, erfolgt kein weiterer Anstieg der Transportkapazität.

Abbildung 4.11 zeigt einen weiteren Simulationslauf mit summierter Transportkapazität. Die Dauer des Simulationslaufes betrug diesmal 1000 Sekunden, wobei erneut alle 2 Sekunden eine Aufzeichnung der Daten erfolgte. Im Unterschied zur ersten Simulation wurde die Bestellmenge der Consumeragenten zwischen 400 und 800 Sekunden von 10 auf 15 Produkte pro Bestellung erhöht. Deutlich zu sehen ist der Anstieg nach 400 Sekunden und der Abfall nach 800 Sekunden. Kurz nach Erhöhung der Ordermenge zeigt die Kurve einen leichten Einbruch.

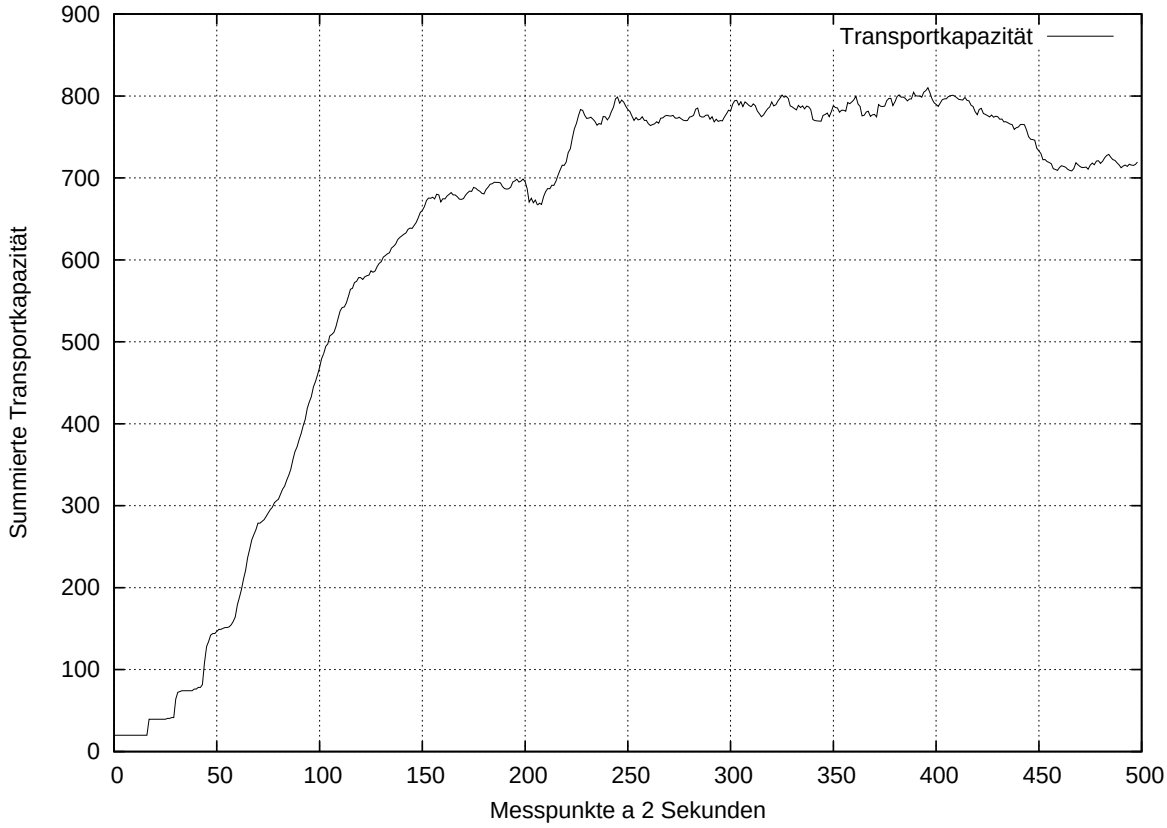


Abbildung 4.11: Summierte Transportkapazität über den gesamten Lauf. Zwischen 400s und 800s wurde die Bestellmenge von 10 auf 15 erhöht.

Die Ursache hierfür kann an bereits angepassten Transportagenten mit einer Transportkapazität von $s(transportcapacity) < 15$ liegen. Diese sind auf eine kleinere Ordermenge angepasst und bekommen keine Aufträge mehr, da ihre Größe nicht ausreicht. Die Erhöhung der Bestellmenge bewirkt aus Sicht der Agenten eine Umgebungsveränderung. Bereits angepasste Agenten sterben aus, während Agenten mit höherer Transportkapazität eine vergrößerte Umgebung vorfinden. Damit stirbt ein Teil der Population aus, während der Rest explorativ die Suche fortsetzt und die Population zeitnah in etwa 100 Sekunden (Messpunkte 200 bis 250) an die neue Umgebung anpasst.

Analog zu Abbildung 4.10 zeigt Abbildung 4.12 die während der Simulation aufgetretenen sogenannten *missed orders*. Diese Bestellungen konnten nicht ausgeführt werden, etwa weil ein Agent nicht über genügend Kapazität verfügt, gerade beschäftigt ist, bankrott gegangen ist oder keine Produkte im Lager hat. Der anfängliche Peak ist einerseits der geringen Anzahl von Transportagenten zuzurechnen. Andererseits müssen sich erst auch alle Agenten jeweils einen freien Dienstleister suchen. Da dies in DIET nicht über ein zentrales Diensteverzeichnis, sondern dezentral abgewickelt wird, entstehen zu Beginn Verwerfungen. Erstaunlicherweise pendelt sich diese zunächst zufällige Servicesuche überraschend schnell ein. Gleichzeitig mit dem Anstieg der Transportkapazität sinkt die Zahl der *missed orders* schnell. Aufgrund der organischen Servicediscovery und der kontinuierlichen Populationsveränderungen verbleibt ein Rest an *missed orders* zu jedem Zeitpunkt.

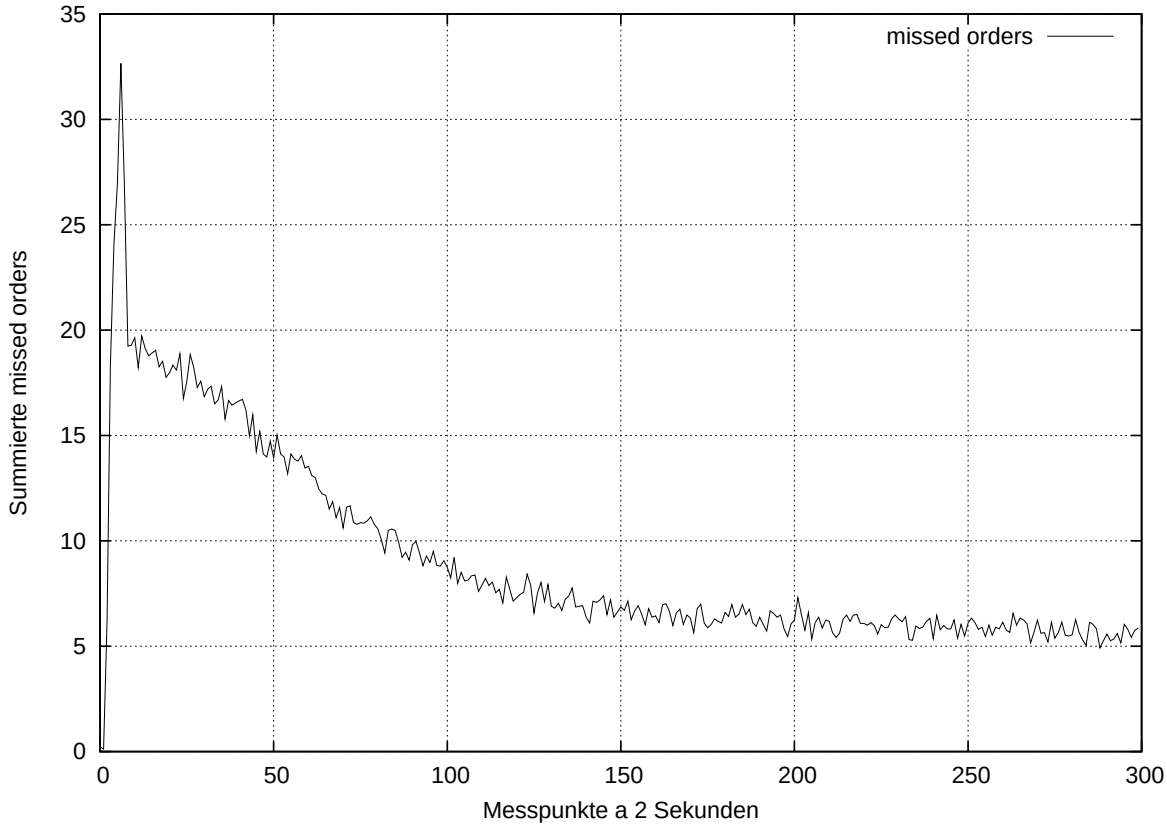


Abbildung 4.12: Sogenannte missed orders über den gesamten Simulationszeitraum

Eine weitere Reihe von Experimenten beschäftigte sich mit der Evaluation unterschiedlicher Werte für θ. Hierbei wurden Werte von 30, 40, 50, 60 und selbst-adaptive θ mit Startwert 40 verwendet. Abbildung 4.13 zeigt die resultierenden *missed orders* während der Simulation bis zum Erreichen der Tragfähigkeit. Hierbei ist zu beobachten, dass kleinere Werte von θ zur Erhöhung der Adaptionsgeschwindigkeit führen. Je höher θ, desto langsamer erfolgte die Annäherung der *missed order*s dem Minimum. Die Werte aller unterschiedlichen Simulationen konvergierten jedoch zum gleichen Minimum. Bei ansonsten gleichbleibenden Umgebungsbedingungen führt die Verringerung von θ zur Erhöhung der Adaptionsgeschwindigkeit, da in gleicher Zeit bei gleichbleibendem Profit der Agenten mehr Reproduktionen erfolgen. Werte von $\theta < 25$ führten zum Aussterben der gesamten Population der Transportagenten und wurden daher nicht berücksichtigt. Daher kommt dem richtigem Konfiguration von θ eine entscheidende Rolle zu. Mit dem rational vorgeschlagenem Wert von 40 wurde eine gute Wahl getroffen, denn der beste Wert liegt etwas tiefer bei 30.

Einen Sonderfall bildet die Verwendung von θ als Strategieparameter für Transportagenten $s(theta)$. Hierbei konnten aus Sicht der Ergebnisse keine signifikanten Unterschiede erkannt werden. Abbildung 4.14 zeigt den Verlauf von $s(theta)$ für die gesamte Population der Transportagenten. Hierzu wurde das Maximum, der Durchschnitt und das Minimum aufgezeichnet. Bemerkenswert ist der leichte Anstieg von initial $s(theta) = 40$ auf $s(theta) = 45$ nach 150 Sekunden. Aus Sicht des Agenten ist dieses Verhalten vorteilhaft, da hierdurch lokal vorgehaltende Geldmittel $funds(a)$ ebenfalls höher sind. Eine Erhöhung der Geldmittel führt zu

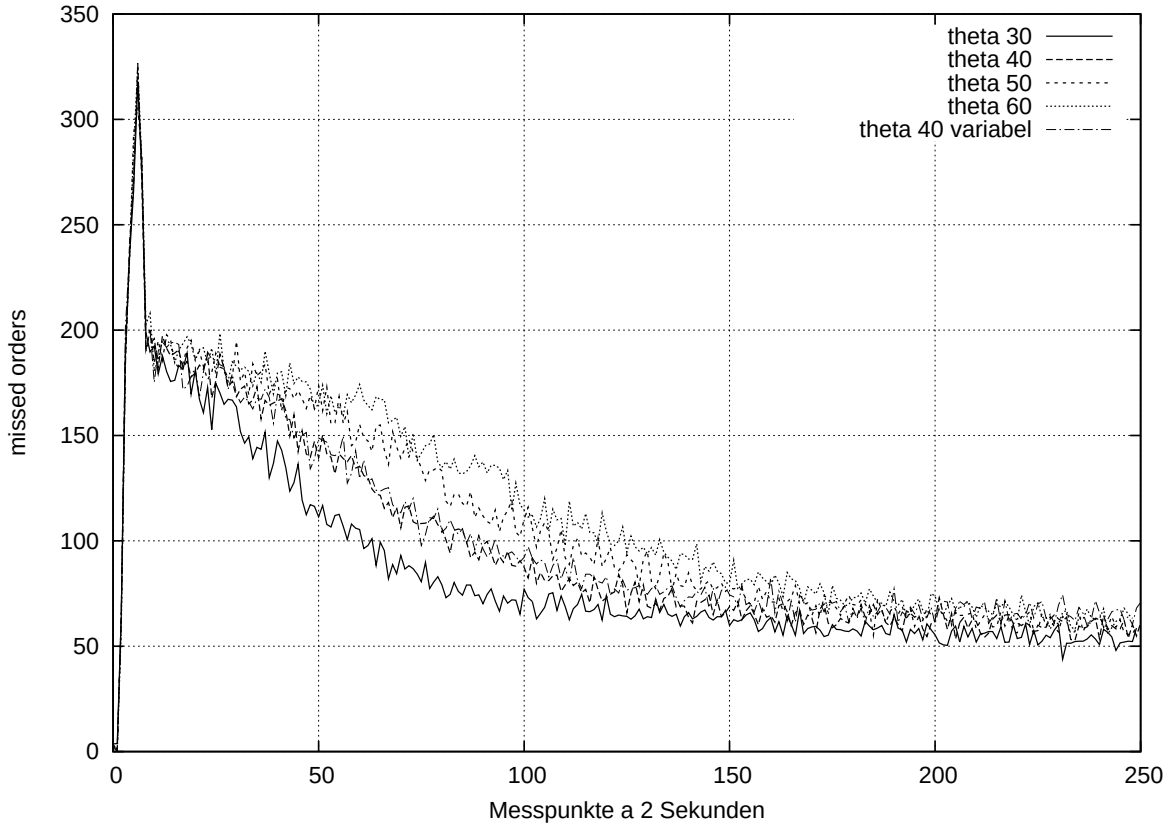


Abbildung 4.13: missed orders für unterschiedliche θ

längerer Lebensspanne, insbesondere, wenn sich die Umgebungsbedingungen verschlechtern.

Abbildung 4.15 zeigt Schnappschüsse für $t = 1, 50, 100, 250, 500$ über die Kapazitäten der Transportagenten als Histogramm. Zusätzlich ist der Mittelwert angezeigt. Aus Agentensicht ist die initiale Strategie mit einer Transportkapazität von $s(transportcapacity) = 20$ suboptimal, da die Bestellmengen der Kunden deutlich geringer ausfallen. Andererseits wird die volle Transportkapazität, egal ob genutzt oder nicht, besteuert (siehe Gleichung 4.1). Daher ist bei unveränderter Umgebung eine Transportkapazität von 10 optimal.

Abbildung 4.15a) zeigt zum Zeitpunkt $t = 1$ einen Transportagenten. Zeitpunkt $t = 50$ (Abbildung 4.15b)) zeigt bereits die Entstehung einiger effizienterer, da Transportagenten mit geringerer Kapazität von $s(transportcapacity) = 19$. Zum Zeitpunkt $t = 100$ hat sich dieser Trend fortgesetzt und eine weitere Diversifizierung der Transportkapazitäten stattgefunden mit effizienteren Strategien von $s(transportcapacity) \in \{15, 16, 18, 19\}$. Die am stärksten verbreitete Strategie ist jedoch weiterhin $s(transportcapacity) = 20$. Die Population ist bereits auf 29 Agenten angewachsen.

Nach $t = 250$ (Abbildung 4.15d)) hat sich der Trend des Populationswachstums und der Strategieanpassung fortgesetzt. Die Strategien sind nun verteilt im Intervall $[3, 30]$ und der Mittelwert ist auf 16.16 gesunken. Bereits zwei Transportagenten verwenden die optimale Strategie von $s(transportcapacity) = 10$. Ein weiterer Effekt der geringeren Steuerbelastung von optimalen Strategien ist eine Vergrößerung der Population auf 43 Transportagenten bei gleichbleibender Gesamtkapazität.

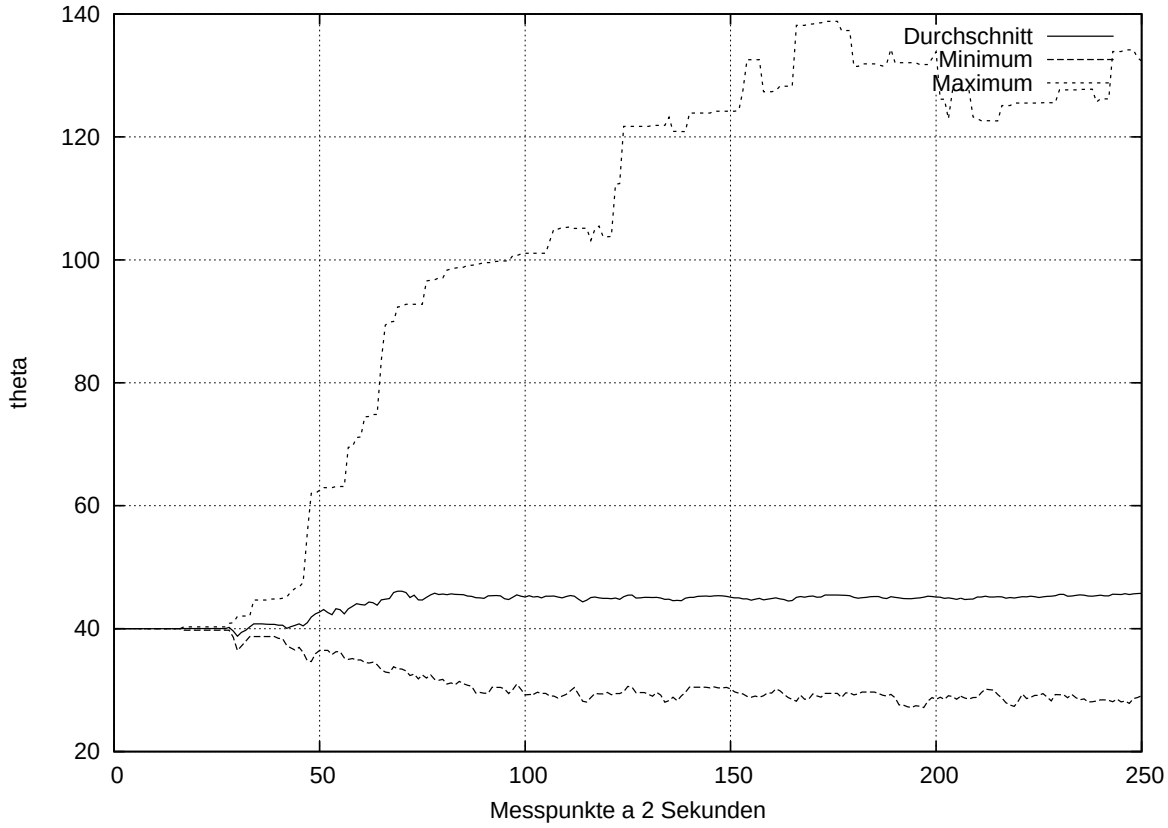
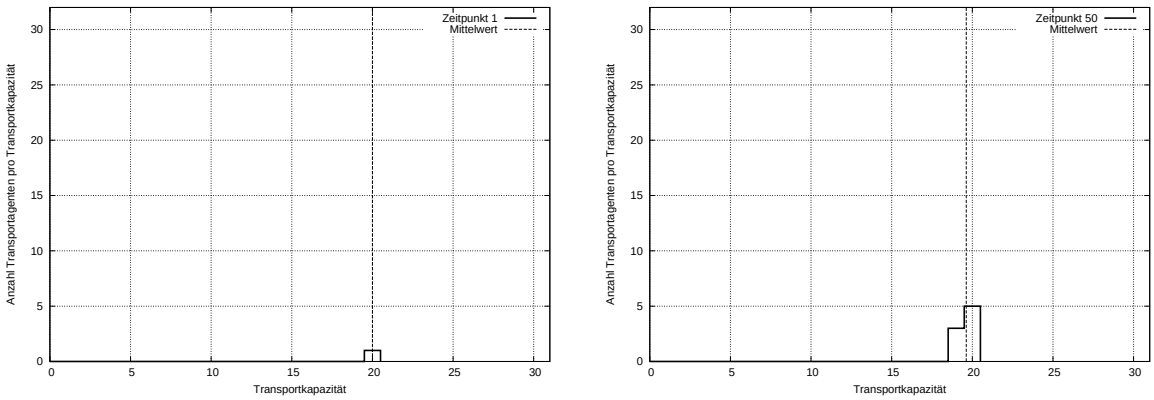


Abbildung 4.14: Entwicklung selbst-adaptiver θ mit Startwert 40

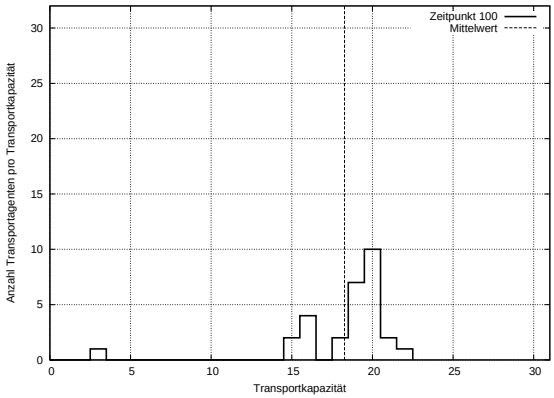


(a) Zeitpunkt 1

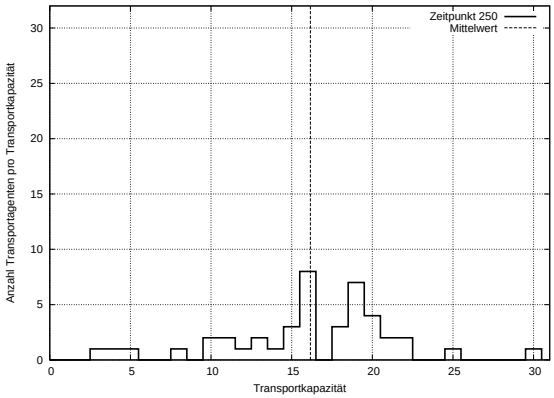
(b) Zeitpunkt 50

Abbildung 4.15: Histogramme der Transportkapazität zu den Zeitpunkten 1(a) und 50(b)

Der Schnappschuss $t = 500$ (Abbildung 4.15e)) zeigt schließlich den Endzustand. Der Großteil der Population läuft mit optimaler Strategie und eine weitere Vergrößerung der Population auf 72 Transportagenten hat stattgefunden. Sichtbar sind auch Agenten mit $s(transportcapacity) < 10$. Diese befinden sich temporär in der Population, bis die Geld-

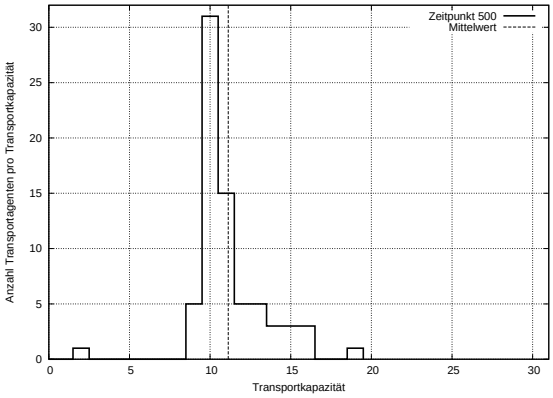


(c) Zeitpunkt 100



(d) Zeitpunkt 250

Abbildung 4.15: Histogramme der Transportkapazität zu den Zeitpunkten 100(c) und 250(d) (Forts.)



(e) Zeitpunkt 500

Abbildung 4.15: Histogramm der Transportkapazität zum Zeitpunkt 500(e) (Forts.)

mittel aufgebraucht sind, da sie keine Aufträge bekommen.

Die Versuchsergebnisse zeigen, dass evolutionäre Agenten in verteilten Logistikumgebungen zum Einsatz kommen können. Hierbei ließen sich alle im Modell vorhergesagten Effekte beobachten. Hierzu zählt der Adaptionsprozess auf Populationsebene und die Strategieanpassung auf Agentenebene. Der kontinuierliche Anpassungsprozess konnte deutlich verifiziert werden am Beispiel veränderlicher Kundennachfrage. Ebenso wurden wertvolle Erkenntnisse der ökonomischen Perspektive aufgezeigt. Bezüglich der Bestimmung von θ hat sich gezeigt, dass der rationale Ansatz nahezu optimale Werte ergeben kann. Somit kann dieser Ansatz zum Einsatz kommen, wenn nicht auf alle Daten seitens eines Problemlösers Zugriff besteht. Es lässt sich daher auch zur Optimierung von Problemen einsetzen, die über mehrere Organisationen bzw. Unternehmen verteilt sind. Hierbei muss lediglich die Erweiterung evolutionärer Agenten in ein bestehendes Agentensystem eingebracht werden und eine einheitliche Kommunikation unter den Agenten sichergestellt sein.

4.3 Facility Location in Mixed Mode Environments

Mixed Mode Environments (*MME*) bezeichnet Netzwerke, bestehend aus unterschiedlichsten Arten von Geräten in physikalischen Umgebungen. Hierzu zählen Sensoren, Aktuatoren, Roboter, unbemannte Fahrzeuge (engl. *Unmanned Vehicles*, *UV*), mobile Endgeräte (z.B. PDA, Handy), *Human Interface Devices* (dt. Eingabegeräte, *HID*), PC, Server usw. Diese Geräte sind in heterogenen physikalischen Umgebungen (indoor, outdoor, unter Wasser, im Boden, etc.) verteilt und interagieren über unterschiedliche Kommunikationskanäle. Die Heterogenität der Geräte ist bedingt durch unterschiedliche Hersteller, Hardware, Software und auch Eigentümer. Mit dem Begriff MME sind unterschiedliche zum Teil überlappende Bereiche zusammengefasst: *Wireless Sensor Networks* (*WSN*), *Wireless Sensor and Actor Networks* (*WSAN*) und *Ubiquitous Computing* (Ubiquitäres Rechnen) [7].

Diesen Bereichen gemeinsam ist die verteilte Bearbeitung einer gemeinsamen Aufgabe durch das Netzwerk von Geräten, welche die Knoten darstellen. Geräte sowie die Umgebung sind heterogen, dynamisch und räumlich verteilt. Knoten können ausfallen, neu hinzukommen oder die Verbindung zum Netzwerk verlieren. Zusätzlich können sich mobile Knoten in der Umgebung bewegen. In MME kann ein Netzwerk nicht auf konstante Verfügbarkeit der Knoten vertrauen und muss trotz fehlender zentraler Kontrollinstanz möglichst selbstadaptiv und selbstoptimierend bleiben. Vor dem Hintergrund dieser Aspekte bedeutet gemeinsames und verteiltes Bearbeiten von Aufgaben ein ideales Anwendungsgebiet evolutionärer Agenten.

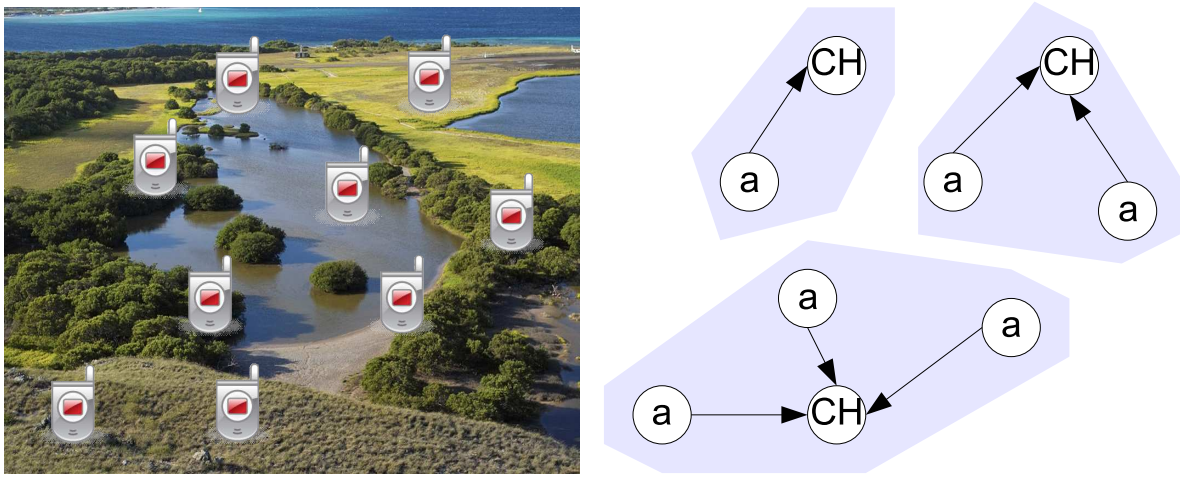
In den folgenden Abschnitten wird zunächst ein Beispielszenario vorgestellt. Danach erfolgt in Abschnitt 4.3.2 die Problembeschreibung und damit die Adaption des Ansatzes evolutionärer Agenten auf das konkrete Szenario. Im Abschnitt Experiment 4.3.3 wird näher auf die Realisierung eingegangen und verwendete Einstellungen, Annahmen und Implementierungsdetails vorgestellt. Schlussendlich erfolgt die Auswertung der Ergebnisse in Abschnitt 4.3.4.

4.3.1 Szenario

Agentenbasierte Ansätze für MMEs sind ein vielversprechendes Anwendungsgebiet von Agenten [7]. Hierbei wird jedes Gerät durch einen Agenten repräsentiert. Der Agent kennt die Fähigkeiten und den Kontext des Gerätes, auf welchem er sich befindet. Zur Untersuchung des Ansatzes evolutionärer Agenten wird von der physikalischen Ebene abstrahiert und lediglich das Agentensystem betrachtet.

Abbildung 4.16a) zeigt ein einfaches Szenario zur Überwachung einer Umgebung mittels Sensoren. Hierbei handelt es sich um das Monitoring z.B. von Schadstoffkonzentration in der Umgebung. Dazu sind die Sensoren über das gesamte Gebiet verteilt. Die Kommunikation erfolgt im Allgemeinen drahtlos. Ziel ist eine möglichst kostengünstige Überwachung der Umgebung. Kostengünstig bedeutet in diesem Kontext effizient und energiesparend.

Die Überwachung erfolgt lokal auf jedem Knoten, indem z.B. jede Stunde eine Messung vorgenommen wird. Zur Messung wacht der Knoten aus seinem Stromsparmodus oder Schlafmodus für kurze Zeit auf und schläft danach wieder ein. Die Messwerte werden zeitnah an entsprechende Leitstellen weitergegeben. Die Reichweite eines Sensors ist begrenzt, so dass der betreffende Sensor nicht direkt Kontakt mit der Leitstelle aufnehmen kann. Eine Weiterleitung der Informationen über andere Knoten ist nur möglich, wenn diese sich im Moment des Sendens nicht im Stromsparmodus befinden. Daher entsteht aus dieser Situation heraus ein Kommunikationsproblem.



(a) Umgebung mit Sensoren (b) Agentennetzwerk

Abbildung 4.16: Monitoringszenario mit Sensoren: a) Sensoren in Umgebung, b) zu a) korrespondirendes Agentensystem (a - Agent, CH - Cluster-Head)

Aus diesem Grund ist das Sensornetzwerk in sogenannte *Cluster* unterteilt. Abbildung 4.16b) zeigt ein Sensornetzwerk, bestehend aus Clustern (grau hinterlegt). Auf jedem Sensor läuft ein Agent (mit *a* gekennzeichnet). Ein Agent pro Cluster übernimmt die Aufgabe des sogenannten *Cluster-Head* [150] (mit CH gekennzeichnet). CH-Agenten übernehmen neben der Schadstoffmessung zusätzliche Kommunikationsaufgaben. Daher wechseln CH-Agenten nicht in den Schlafmodues zwischen den Messungen. Messungen werden an den Cluster-Head gesendet, der mit einer Bestätigung antwortet, um eine sichere Übertragung zu gewährleisten. Wird keine Bestätigung empfangen, so geht der Agent davon aus, dass kein Cluster-Head aktiv ist und wechselt selbständig in den Cluster-Head Modus. Der Cluster-Head sorgt für die Weiterleitung auf Cluster-Head Ebene. Daher verbleibt aus der Menge der Agenten die Teilmenge der Cluster-Head Agenten stets aktiv. Prinzipiell kann jeder Agent die Aufgaben des Cluster-Heads übernehmen, so dass nach einer gewissen Zeit bei entsprechend geringer Energieversorgung ein Cluster-Head wieder in den Normalmodus wechselt. Ein anderer Agent kann dann anstelle des bisherigen Cluster-Head einspringen und für eine neue Kommunikationsstruktur sorgen. Hierdurch wird eine dauerhafte Überwachung mit allen Sensoren gewährleistet und gleichzeitig bei 'richtiger' Positionierung der Cluster-Heads eine energieeffiziente Weiterleitung ermöglicht.

Das weiter oben angesprochene Kommunikationsproblem und die optimale Bestimmung der Cluster-Heads lässt sich auf ein Facility-Location Problem reduzieren.

4.3.2 Problembeschreibung

In diesem Abschnitt wird zunächst das Facility Location Problem [249, 250] formal beschrieben[6]. Danach erfolgt die Formulierung des evolutionären Agentenmodells für das Facility Location Problem.

[6]Der Name Facility Location wird auch im deutschen Sprachgebrauch benutzt. Facility Location beduetet Standortbestimmung

Eine Menge $L = \{1 \dots N\}$ enthält potentielle Standorte zur Erbringung von Services. Ein Service kann an jedem Standort $l \in L$ eröffnet werden. Weiterhin existiert eine Menge von Kunden $J = \{1, \dots, M\}$, welche Services in Anspruch nehmen. Für jedes Paar (l, j) sind die Transportkosten gegeben durch $g_{l,j} \geq 0$. In der Literatur gibt es die Unterscheidung zwischen limitierten [249] und unlimitierten [17] Services. Im unlimitierten Fall kann jeder Service unbegrenzt viele Dienste erbringen, Produkte herstellen etc., während im limitierten Fall eine obere Schranke angegeben ist.

Das Ziel des einfachen Facility Location Problems ist zunächst die Bestimmung einer Teilmenge von Standorten $S \subseteq L, S \notin \emptyset$ zur Erbringung der Services. Weiterhin ist eine Zuordnung von Kunden zu Services durchzuführen um die Transportkosten zu minimieren. Im p-Median Problem entfällt Schritt 1, da die Anzahl der Standorte bereits mit p festgelegt ist. Das resultierende Problem lässt sich wie folgt formulieren:

$$F(S) = \sum_{j \in J} \min_{l \in S} g_{l,j} \rightarrow \min_{S \subseteq L} \tag{4.2}$$

Im limitierten Fall ist das Facility Location Problem eine Verallgemeinerung des Mengenüberdeckungsproblems [161] und damit NP-hart [213]. Das p-Median Problem gehört für variable p ebenso in die Klasse der NP-harten Probleme. Diese Aussage gilt in beiden Fällen für generelle Graphen und Netzwerke. Für spezielle Strukturen, wie Bäume existieren polynomielle Verfahren. Für fixe p, das P-Median Problem ist für generelle Graphen lösbar in polynomialer Zeit [198].

Die Anwendung evolutionärer Agenten auf das Facility Location Problem im Kontext des oben beschriebenen Szenario erfordert zunächst die Einführung der ökonomischen Perspektive. Diese entspricht auf Sensorebene mit der Energieversorgung, denn falls die Energie eines Knotens aufgebraucht ist, erfolgt automatisch die 'Entfernung' aus dem System. Zunächst wird ein Agentensystem als Schicht über der Hardware und evtl. vorhandenen Betriebssystemschicht angenommen. Auf jedem Sensor kann damit ein Agent laufen. Agenten können sowohl als normale Messagenten als auch als Cluster-Head fungieren. Dazu stellt jeder Cluster-Head die Kommunikation über seinen Standort j als Service zur Verfügung. Dazu zählen neben der Übermittlung von Messwerten auch die Übermittlung von Strategienachrichten im jeweiligen Cluster und Aktivierungsnachrichten, falls ein Messagent zum Cluster-Head ernannt wird. Zur Vereinfachung wird das zu betrachtende Agentensystem evolutionärer Agenten MAS daher lediglich über die Menge der Cluster-Head Agenten betrachtet. Messagenten dienen zur Repräsentation der Kunden, die den Service der Kommunikation in Anspruch nehmen.

Die Kosten zur Übermittlung von Nachrichten eines Messagenten zum Cluster-Head bestehen aus den Nachrichtenübertragungskosten $g_{a,j}$ multipliziert mit einem Profitfaktor $profitfactor_a \in \mathbb{R}$ des jeweiligen Cluster-Head a:

$$cost_{aj} = g_{a,j} * profitfactor_a \tag{4.3}$$

$g_{a,j}$ entspricht den Transportkosten und $cost_{aj}$ sind die zu zahlenden Kosten für die Nachrichtenübertragung. Der Cluster-Head bestimmt durch seinen Profitfaktor die Transportkosten. Messagenten wählen immer den Cluster-Head mit den geringsten Kosten aus, falls mehrere in Reichweite vorhanden sind und auf Nachrichten antworten:

$$\sigma_{aj}(t) = \begin{cases} 1 & \text{Messeagent } j \text{ sendet an } a \text{ zum Zeitpunkt } t \\ 0 & \text{sonst} \end{cases}$$

Die fixen Kosten für Cluster-Heads werden mit $cost_{CH}(t)$ pro Zeiteinheit angenommen, um den hohen Energieverbrauch abzubilden. Für Messagenten besteht somit die Nutzenmaximierung im Wählen eines erreichbaren Cluster-Head mit möglichst geringen Nachrichtenübertragungskosten $cost_{aj}$. Für Cluster-Head Agenten besteht die lokale Optimierung zum Einen darin, einen möglichst guten Standort zu besetzen. Dies bedeutet, dass möglichst keine weiteren Cluster-Heads in der Nachbarschaft existieren. Zum Anderen muss der Profitfaktor $profitfactor_a$ niedriger sein als bei konkurrierenden Cluster-Heads aber auch gleichzeitig kostendeckend für den Betrieb des Services der Nachrichtenübermittlung. Daher setzt sich die Strategie s_a eines Cluster-Heads aus folgenden Komponenten zusammen:

Profitfaktor ($s_a(profitfactor) \in \mathbb{R}$): gibt den Faktor an, welcher vom Cluster-Head zum eigenen Nutzen über den Nachrichtentransport hinaus verwendet wird.

Standort ($s_a(location) \in L$): ist die Angabe des Standortes. Dieser kann vom Agent nicht gewählt werden, sondern wird vorgegeben. wird jedoch für die Strategieoptimierung auf MAS Ebene benötigt.

Die Strategie des MAS besteht damit aus den Standorten aller Cluster-Head Agenten und deren Profitfaktor. Damit kann das Facility Location Problem als VOP Problem (siehe Definition 16) formuliert werden:

$$\text{minimiere } costsys(S_A), \tag{4.4}$$

unter Nebenbedingungen:

$$cost = \text{Kostenfunktion} \tag{4.5}$$

$$F(S_A) = \sum_{j=1}^{J}\sum_{a=1}^{A} g_{aj} \tag{4.6}$$

$$\mathcal{R}_a(t) = \sum_{j=1}^{J}\sum_{a=1}^{A} cost_{aj}(t) \cdot \sigma_{aj}(t) \tag{4.7}$$

$$\mathcal{P}_a(t) = 0 \tag{4.8}$$

$$\mathcal{T}_a(t) = cost_{CH}(t) + \sum_{j=1}^{J}\sum_{a=1}^{A} g_{aj} \cdot \sigma_{aj}(t) \tag{4.9}$$

Wobei $costsys(S_A)$ für die Gesamtsystemkosten steht. Zur besseren Übersichtlichkeit wurde anstelle des Standortes $s_a(location)$ einfach a verwendet, z.B. g_{aj} anstelle von $g_{s_a(location)j}$. Gleichungen 4.7 - 4.9 beschreiben die Terme der Profitrechnung für einen Cluster-Head Agenten (siehe Gleichung 3.11: $\pi_a(t) = \mathcal{R}_a(t) - \mathcal{P}_a(t) - \mathcal{T}_a(t)$). $\mathcal{R}_a(t)$ beschreibt eingehende Zahlungen über Zeitraum t, bestehend aus allen Zahlungen von Messagenten, die ihre Nachrichten über a versenden. $\mathcal{P}_a(t)$ beschreibt Zahlungen an andere Cluster-Head Agenten. Es werden keine Cluster-Head internen Zahlungen angenommen, stattdessen erfolgt eine pauschale Besteuerung per $cost_{CH}(t)$ in Gleichung 4.9. Steuern werden einerseits über die Pauschale $cost_{CH}(t)$ erhoben und andererseits über die Höhe der tatsächlichen Kommunikationskosten.

Damit bestimmt sich der tatsächliche Profit eines Cluster-Head Agenten über seine Strategievariable Profitfaktor ($s_a(profitfactor)$). Hierbei gilt ein Profitfaktor von 1 als Kostendeckend ohne Gewinn, < 1 als ruinös, d.h. der CH-Agent erwirtschaftet negativen Gewinn und > 1 als profitabel im ökonomischen Sinne.

4.3.3 Experiment

Um einen reibungslosen Start zu ermöglichen, wurde der Profitfaktor in allen Experimenten initial auf $s_a(profitfactor) = 2.0$ gesetzt. Dies ermöglicht hohe Anfangsgewinne und damit explorativ orientierte Suche. Die Einbeziehung des Profitfaktors in die Strategie ermöglicht die Selbstadaption während der Simulation. Durch die Auswahl des Cluster-Heads seitens der Messagenten wird Selektionsdruck hinsichtlich des Profitfaktors erwartet. Es ist daher davon auszugehen, dass der anfänglich hohe Profitfaktor sinken wird.

Für die Simulationen wurden zwei aus der Literatur bekannte Testprobleme herangezogen. Es handelt sich hierbei um die sogenannten *Alberta* [5] und *Galvao* [81] Probleme. Fr beide Probleme existiert die optimale Lösung. Alberta beschreibt eine Probleminstanz mit 316 Standorten und Galvao enthält 100 mögliche Standorte und die jeweiligen Kosten zwischen allen Standorten. Die verwendeten Einstellungen finden sich in folgender Übersicht:

- Anzahl der Cluster-Head Agenten initial: $|A| = 20$. Die Standortinitialisierung erfolgte zufällig aus der Menge der Standorte ($s_a(location) \in L$). Wurden zwei Agenten zufällig am gleichen Standort erstellt, so wurde der Agent mit höherem Profit entfernt.
- Profitfaktor initial: $s_a(profitfactor) = 2.0$
- Anzahl gemittelter Simulationsläufe: 50
- Mutationsrate: 0.03
- Fixe Steuern der Cluster-Head Agenten: $cost_{CH}(t) = 30000$ im Alberta Problem und $cost_{CH}(t) = 10$ im Galvao Problem. Dieser Wert ergab sich aus den gegebenen Transportkosten der beiden Probleme.
- Simulationszeitraum $\Delta t = 10000$

Die fixen Steuern $cost_{CH}(t)$ wurden in Abhängigkeit der Transportkosten der gegebenen Probleme ermittelt. Hierbei durften die Werte nicht zu hoch und auch nicht zu niedrig sein. Zu hohe Steuern erlauben dem System keine performante Suche bis hin zum Aussterben der gesamten Agentenpopulation nach dem Start. Zu niedrige oder gar keine Steuern hingegen führen dazu, dass ineffiziente Cluster-Head Agenten sehr lange operieren können. Zwei Cluster-Head Agenten ch_1 und ch_2 bieten auf benachbarten Knoten 1 und 2 ihre Dienste an. ch_1 übermittelt die Nachrichten von 10 Messagenten, während ch_2 nur eigene Messungen weitergibt. Steuern stellen in diesem Falle einen Mechanismus dar, um ch_2 sobald als möglich in einen normalen Messagenten umzuschalten. Hiermit wird langfristig effizienter Betrieb auf Netzebene erreicht, da ch_2 z.B. problemlos seine Messergebnisse an ch_1 senden kann.

Das Umschalten von Messagent in Cluster-Head Agent und andersherum wird verursacht auf den Knoten keine expliziten Kosten und wird daher auch nicht besteuert. Erwartet werden neben der Anpassung des Profitfaktors eine effiziente räumliche Abdeckung in Verbindung mit einer Anpassung der Populationsgröße der Cluster-Head Agenten. Dabei besteht die Anpassung der Systemstrategie S_A in der konstanten Optimierung der lokalen Strategie s_a, bestehend aus den Standorten und Profitraten der einzelnen Agenten.

4.3.4 Ergebnisse

Die Ergebnisse sind jeweils nach den beiden Testproblemen Alberta und Galvao aufgeteilt. Die Auswertung zeigt zunächst die pro Anzahl an Cluster-Head gefundene beste Gesamtfitness für beide Testprobleme. Danach wird die Anpassung der Populationsgröße anhand der Cluster-Head Agenten dargestellt. Die Entwicklung des durchschnittlichen Profitfaktors auf Agentenebene wird anschließend gezeigt (Strategieanpassung). Abschließend erfolgt für beide Testprobleme eine Darstellung aller durchlaufenen Lösungen über einen Testlauf. Im Anhang finden sich die gemittelten Lösungen über 50 Testläufe. Zur Auswertung[7] werden folgende Punkte vorgestellt:

- Beste Gesamtfitness pro Anzahl CH Agenten: Durchschnitt von 50 Simulationsläufen
- Anzahl CH Agenten über Zeit: Durchschnitt von 50 Simulationsläufen
- Profitentwicklung: Durchschnitt von 50 Simulationsläufen
- Gesamtfitness über Zeit: Einzelner Simulationslauf
- Gesamtfitness über Zeit: Durchschnitt von 50 Simulationsläufen (Anhang)

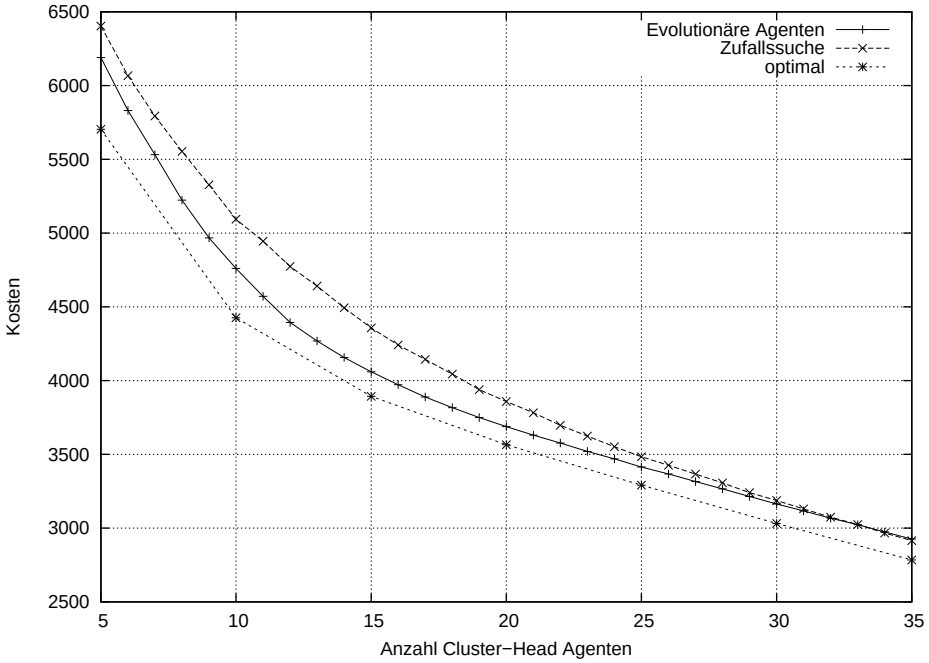


Abbildung 4.17: Vergleich Gesamtkosten Evolutionärer Agenten, Zufallssuche und Optimum pro Anzahl Cluster-Heads (Facilities) für das Galvao Problem

Während eines Testlaufes verändert sich das System der Cluster-Head Agenten ständig. Neue Agenten kommen hinzu, bestehende stellen den Betrieb ein und Messagenten wechseln ihren Cluster-Head. Aus diesem Grunde ändert sich die Gesamtanzahl bestehender CH Agenten. Hierdurch zieht jede Änderung eine andere Gesamtfitness nach sich. Die Gesamtfitness

[7] Ein Teil der Ergebnisse wurde bereits in [185] publiziert

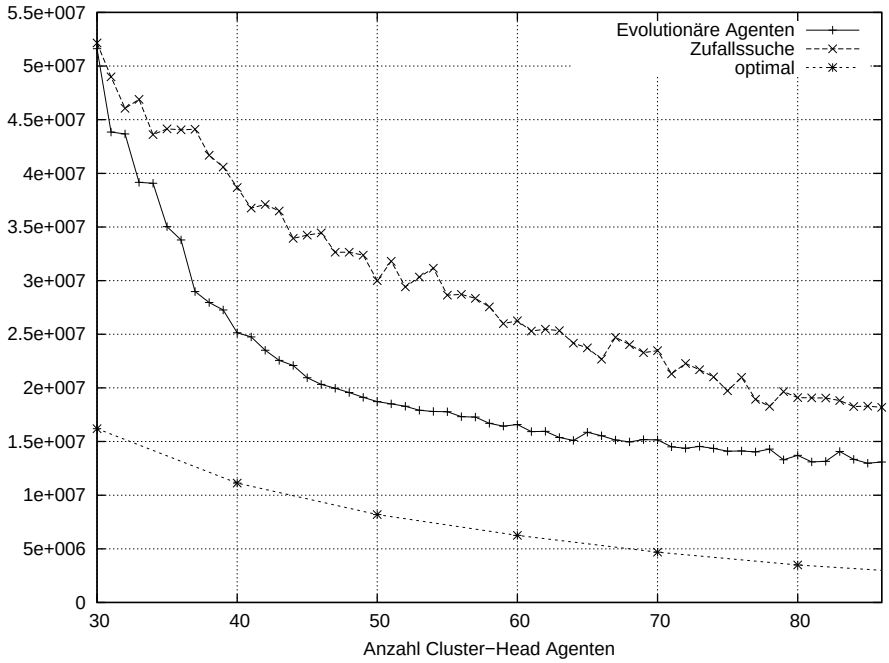


Abbildung 4.18: Vergleich Gesamtkosten Evolutionärer Agenten, Zufallssuche und Optimum pro Anzahl Cluster-Heads (Facilities) für das Alberta Problem

ergibt sich aus den summierten Nachrichtenübermittlungskosten (Gleichung 4.6). Abbildungen 4.17 und 4.18 zeigen jeweils einen Vergleich zwischen evolutionären Agenten, Zufallssuche und den Optimalwerten (jeweils mit den Testproblemen gegeben [5, 81]). Ein direkter Vergleich mit panmiktischen Algorithmen ist an dieser Stelle wenig aussagekräftig, da diese ein solches verteiltes Problem nicht bearbeiten können. Zur Darstellung wurden pro Anzahl CH Agenten jeweils die Bestwerte abgebildet.

Beide Diagramme zeigen eine deutlich bessere Performance evolutionärer Agenten im Vergleich zur Zufallssuche. Aufgrund des Einschwingverhaltens der Populationsgröße (Modell siehe Abschnitt 3.3.1 und Abbildungen 4.19 bzw. 4.20) explorieren evolutionäre Agenten stark um die Tragfähigkeit $|A_{max}|$ mit einer gewissen Abweichung (siehe Abschnitt 3.3.1). Zu Beginn wird natürlich von der initialen Populationsgröße aus bis zur Erreichung der Tragfähigkeit der Umgebung exploriert.

Abbildungen 4.19 bzw. 4.20 zeigen deutlich das initiale Einschwingverhalten bis zur Erreichung der Tragfähigkeit. Zunächst ist ein extremer Anstieg zu beobachten, der sich mit der hervorragenden Profitsituation erklären lässt. Danach schwanken die Werte der Anzahl der CH Agenten um diese Tragfähigkeit. Im Galvao Problem bedeutet dies Werte zwischen $15 - 20$ Agenten und im Alberta Problem etwa $55 - 75$. Daher lassen sich bei den Testproblemen in diesen Intervallen bei Betrachtung der Gesamtfitness in den Abbildungen 4.17 und 4.18 jeweils die besten Annährungen an die optimalen Ergebnisse beobachten.

Abbildungen 4.21 und 4.22 zeigen den gemittelten Verlauf des Strategiewertes $s_a(profitfactor)$ der Gesamten CH Agentenpopulation. Hierbei ist zunächst der schnelle Rückgang von initial 2.0 auf Werte zwischen $1.1 - 1.15$ (Galvao) bzw. $1.2 - 1.3$ (Alberta) zu beobachten. Dieser Rückgang geschieht zeitgleich mit der Anpassung der Populations-

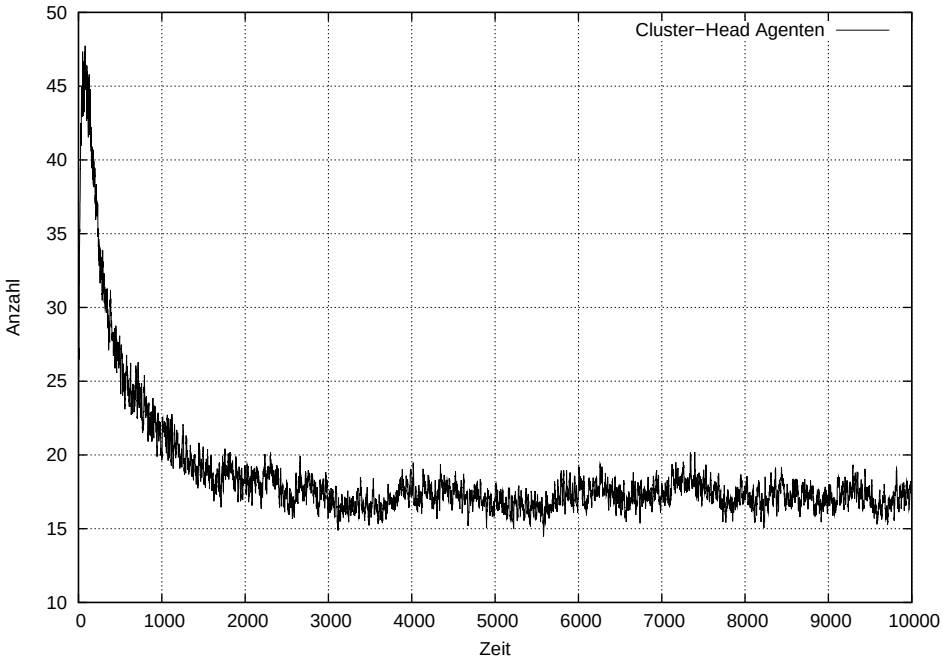


Abbildung 4.19: Anzahl von Cluster-Head Agenten pro Zeit (Galvao)

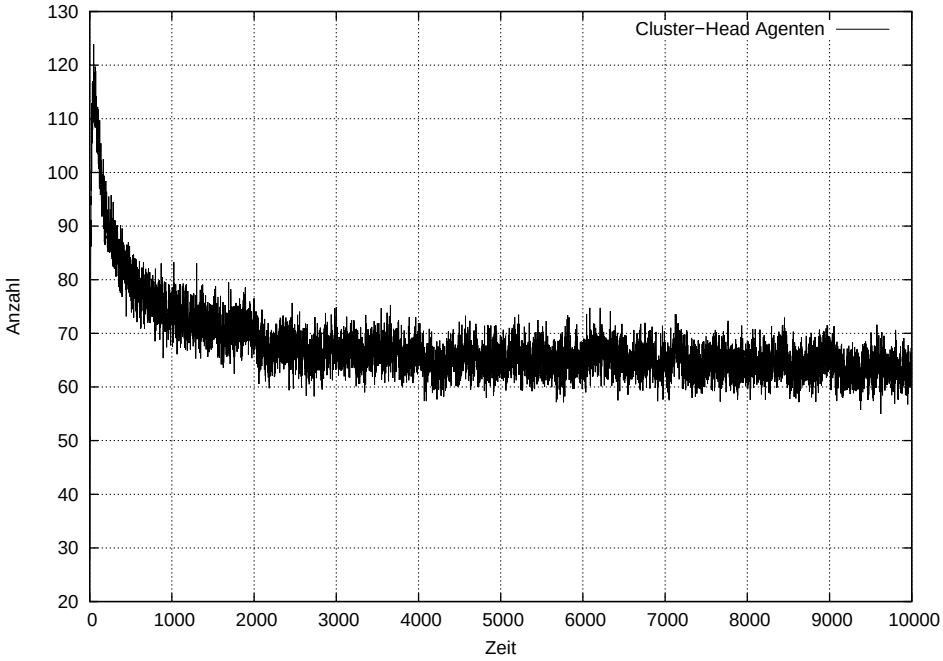


Abbildung 4.20: Anzahl der Cluster-Head Agenten pro Zeit (Alberta)

größe an die Tragfähigkeit der Umgebung. Die mögliche Interpretation beinhaltet zweierlei: steigender Selektionsdruck (Abschnitt 3.3.3) beschleunigt die Verbreitung erfolgreicher Strategien innerhalb der Population (Abschnitt 3.3.2). Zunächst beginnt die Population mit der

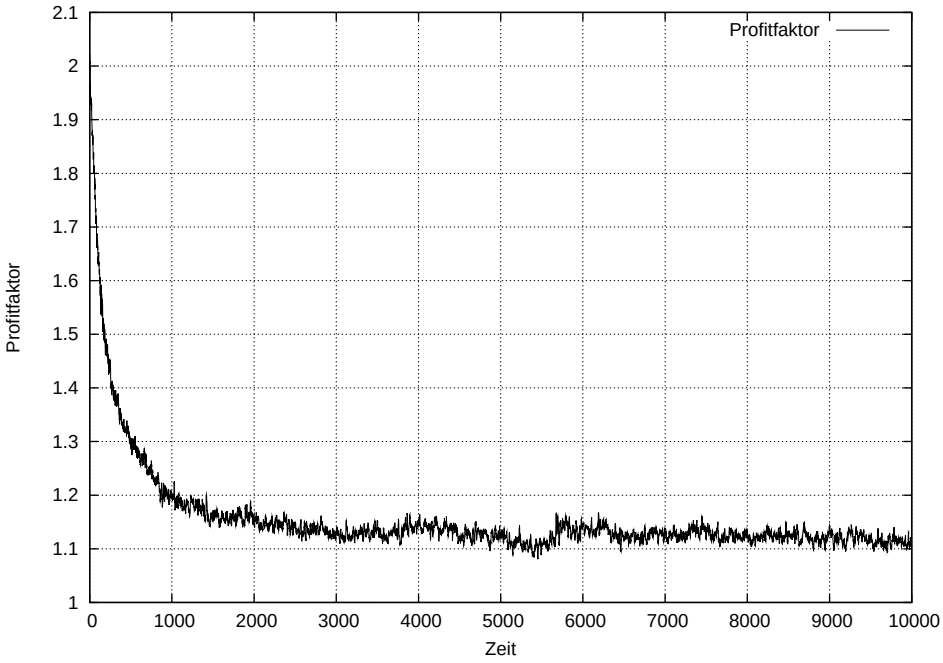


Abbildung 4.21: Profitfaktor der Cluster-Head Agenten pro Zeit (Galvao)

im Vergleich zur Tragfähigkeit geringen Größe von 20 CH Agenten. Hierbei herrscht geringer Selektionsdruck, da die Erzeugung neuer Agenten nicht zu Lasten des Profits der bestehenden Agenten fällt. Zusätzlich ist der initiale Profitfaktor sehr hoch gewählt Danach erhöht sich der Selektionsdruck, sobald die Tragfähigkeit erreicht ist und bedingt durch geringere Profite ein Umschalten von Exploration auf Exploitation in der Suchstrategie erfolgt in Abschnitt 3.3.3.

Der Profitfaktor schwingt sich so ein, dass $\pi_a(t)$ in etwa gegen null tendiert (Theorem 1). Die gezeigten Diagramme stellen Profitfaktoren über 1.0 dar. Hierbei ist zu berücksichtigen, dass die laufenden Kosten $cost_{CH}(t) = 30000$ (Alberta) bzw. $cost_{CH}(t) = 10$ (Galvao) von den Einkünften abzuziehen sind.

Abbildungen 4.23 und 4.24 zeigen alle Systemzustände, die während des Simulationslaufes erreicht wurden. Hierbei handelt es sich nicht um einen Durchschnitt, sondern um einen konkreten Simulationslauf. Mit Zustand ist in diesem Zusammenhang die Abbildung eines Paares 'Anzahl der Cluster-Head Agenten' zu Kosten gemeint. Da die Population nicht fixiert ist, sondern eine ständige Anpassung um die Tragfähigkeit herum stattfindet, verändert sich der Zustand oft.

In den Abbildungen 4.23 und 4.24 sind alle Systemzustände eines Laufes evolutionärer Agenten, einer Zufallssuche und der jeweiligen Optimalwerte abgebildet. Hierbei zeigt sich deutlich der Performanceunterschied evolutionärer Agenten zur zufälligen Suche. Während die Zufallssuche eine breite Streuung aufweist, befindet sich das System evolutionärer Agenten in beiden Problemen signifikant näher am Optimum.

Die Anwendung evolutionärer Agenten auf das Problem der Facility Location zeigt deutliche Performancegewinne im Vergleich zu einer Zufallslösung. Der Vergleich mit panmiktischen Ansätzen oder allgemein zentralistischen Ansätzen ist an dieser Stelle nicht realistisch, da die-

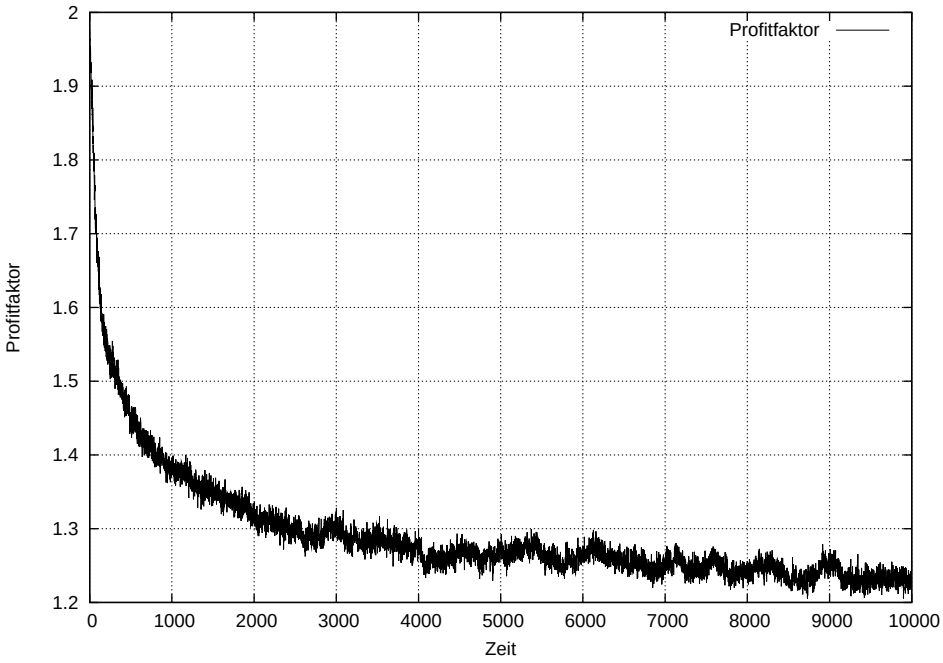


Abbildung 4.22: Profitfaktor der Cluster-Head Agenten pro Zeit (Alberta)

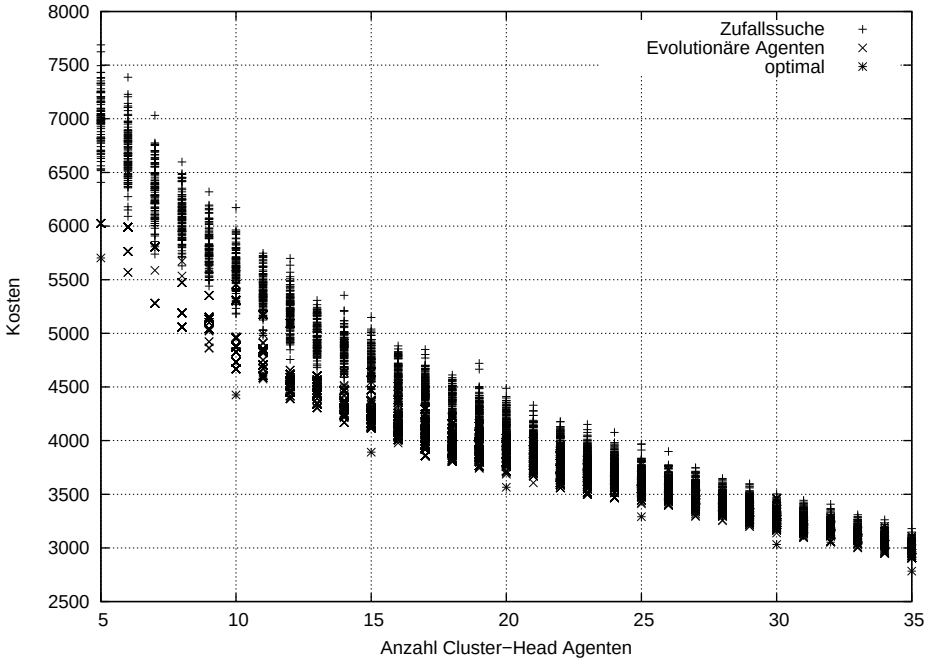


Abbildung 4.23: Gesamtkosten pro Anzahl Cluster-Heads (Galvao)

se Ansätze aufgrund des verteilten Problems in der Praxis nicht eingesetzt werden können. Daher stellt der gezeigte Ansatz eine neue Möglichkeit dar, in verteilten Systemen Optimierung zu realisieren. Alle Modellannahmen und vorhergesagten emergenten Eigenschaften

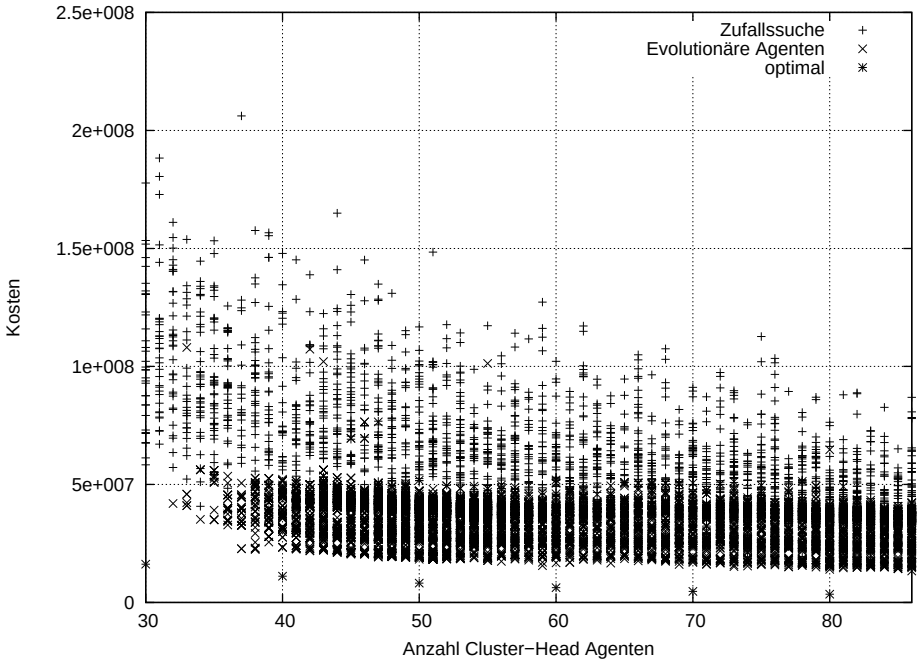


Abbildung 4.24: Gesamtkosten pro Anzahl Cluster-Heads (Alberta)

(siehe Kapitel 3.3) konnten bestätigt werden.

4.4 Zusammenfassung

Mit der Erweiterung von Agenten aus bestehenden Agentensystemen in evolutionäre Agenten werden im Wesentlichen zwei Punkte ergänzt: ökonomische und evolutionäre Funktionalität. Mit einer direkten Umsetzung der Formalisierung in ein Java basiertes und damit plattformunabhängiges System wurde wesentlich zur breiten Einsetzbarkeit beigetragen. Die durch evolutionäre Agenten implizierten Erweiterungen lassen sich, wie in den Beispielen gezeigt, sehr einfach in bestehende Systeme integrieren. Zusätzlich wurde ein eigenes taktbasiertes Agentensystem entworfen und ebenfalls als Grundlage evolutionärer Agenten verwendet. Wesentliche Beiträge wurden weiterhin in der Entwicklung lokal einsetzbarer evolutionärer Funktionalität geleistet. Ebenso innovativ ist die Neuentwicklung des als Open-Source vorliegenden Monitoringframework *JStreaMon* zu bewerten. Hiermit lassen sich insbesondere im Multithreading Umfeld anfallenden Daten in Echtzeit speichern, aufbereiten und weiterverarbeiten. Im Umgang mit Agentensystemen hat sich dieses als unverzichtbares Hilfsmittel erwiesen.

In zwei Studien mit insgesamt drei unterschiedlichen prototypischen Realisierungen hat sich der Ansatz evolutionärer Agenten als sehr robust und vorhersagegenau herausgestellt. So konnten nicht nur die im Modell herausgearbeiteten Merkmale der Adaptivität verifiziert werden, sondern es wurden zusätzlich wertvolle Erkenntnisse über das Design und die Parametrierung evolutionärer Agenten gewonnen. Hierzu zählt im Einzelnen die Verifikation der Adaption auf Populationsebene. Dazu wurden die Effekte sowohl zur Anpassung der Gesamtstrategie nachgewiesen als auch der Populationsgröße nachgewiesen. Die Anpassung und damit Optimierung der Systemstrategie und des Systemverhaltens erfolgte auf Basis gemessener Da-

ten in Abhängigkeit zur geforderten Funktionalität. Ebenso konnten Anpassungseffekte, wie z.B. Größenänderungen, vor dem Hintergrund der zur Verfügung stehenden Geldmittel beobachtet werden. Weiterhin wurden lokale Anpassungen auf Agentenebene nachgezeichnet. Hierzu zählen z.B. die Anpassung der Transportkapazität bei Transportagenten und die Anpassung des Profitfaktors bei Cluster-Head Agenten.

Beachtenswert sind die gewonnenen Erkenntnisse im Umgang mit der Bestimmung der Reproduktionsschwelle θ. Hier konnte bereits mit deren rationaler Bestimmung ein gutes Systemverhalten nahe am Optimum erreicht werden. Ebenso interessant verlief die selbstadaptive Bestimmung der Reproduktionsschwelle und des Profitfaktors in den unterschiedlichen Experimenten. Hier zeigte sich beim Profitfaktor ein Trend zum ökonomischen Gleichgewicht und damit zu schlanker Serviceerbringung. Auf der anderen Seite hat die Selbstadaption der Reproduktionsschwelle gezeigt, dass Agenten dazu tendieren, Parameter in Richtung Sicherheit zu verschieben. Damit sind für Weiter- und Neuentwicklungen im Umgang mit dezentralen und verteilten evolutionären Verfahren neue Erkenntnisse entstanden. Zusätzlich hat sich gezeigt, dass evolutionäre Verfahren nicht an a priori festgelegte Kommunikations bzw. Netzwerkstrukturen gebunden sind. Durch Einsatz in Agentensystemen sind der Verwendungszweck und die Einsatzmöglichkeiten enorm steigerbar. Durch den Einsatz und die Verbindung ökonomischer und biologischer Prinzipien ergibt sich damit eine erfolgreiche, vielversprechende Verbindung und ein nutzbringendes Forschungsfeld.

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

Die starke Vernetzung von Computersystemen hat in den letzten Jahren zu enormen Veränderungen in nahezu allen Bereichen geführt. Beispiele hierfür sind das Internet, Grid-Computing, Peer-to-Peer Netzwerke und Methoden wie z.B. Agentensysteme. Diese Entwicklung ist partiell das Resultat verteilter Probleme und verteilter Systeme. Die folgenden Problembereiche hinsichtlich Adaptivität und Optimierung von verteilten Systemen ergeben sich hieraus (siehe Abschnitt 1.2):

- **Verteilung**:
  - Zentrale Ressourcen sind begrenzt in ihrer Fähigkeit, Daten zu speichern, zu übertragen und zu verarbeiten. Informationen können nicht immer zentralisiert werden.
  - In unternehmensübergreifenden Geschäftsumgebungen existiert eine kommunikationseinschränkende Informationsasymmetrie. Informationen dürfen nicht immer zentralisiert werden.
- **Dynamik**: während zentral eine Lösung erstellt wird, hat sich das Problem bereits verändert.
- **Heterogenität**: die Bestandteile der Systeme unterscheiden sich hinsichtlich ihrer Hardware, Software, Betriebssystem, Funktionalität und Zugehörigkeit zu unterschiedlichen Organisationen.
- **Komplexität**: moderne Informationssysteme weisen komplexe Strukturen auf und lassen sich durch zentrale und Top-Down Ansätze zunehmend schwerer managen.

In diesem Umfeld dynamischer, nicht zugreifbarer und komplexer Systemstrukturen stellt die Adaption und Optimierung von Systemen und Geschäftsprozessen ein nach wie vor nur unzureichend gelöstes Problem dar. Im Allgemeinen sind Adaptions- bzw. Optimierungsverfahren so ausgelegt, dass Informationen für verteilte Probleme zentral gesammelt und bearbeitet werden, um ein möglichst gutes Ergebnis zu erzielen. Es existieren eine Reihe von speziellen Ansätzen zur verteilten Optimierung, deren Funktionsweise auf ein Problem bzw. eine eingeschränkte Anzahl von Problemen zugeschnitten ist. Häufig müssen die Akteure kooperativ zusammenarbeiten, um eine Lösung zu erreichen oder die Daten werden zentral bearbeitet. Dies

ist in verteilten Systemen, deren Kontrolle unterschiedlichen Intressengruppen unterliegen, nicht immer möglich. Zunächst wurden daher bestehende Informationssysteme untersucht und deren Ähnlichkeit mit sogenannten komplexen adaptiven Systemen (CAS) gezeigt. Natürliche CAS zeichnen sich durch eine hohe Flexibilität in dynamischen Umgebungen aus. Diese Flexibilität wird durch Selbstadaption erreicht. Bei bestehenden Adaptions- und Optimierungsverfahren in verteilten Systemen besteht Forschungsbedarf hinsichtlich Selbstadatptions- und optimierungsmechanismen. Eine Übertragung biologischer Mechanismen in Form von evolutionären und agentenorientierten Ansätzen wird daher vorgeschlagen.

Die vorgestellten evolutionären Agenten sind eine Kombination aus evolutionären, ökonomischen und agentenbasierten Ansätzen. Entgegen den üblicherweise verwendeten zentralen Ansätzen sind die hier eingesetzten evolutionären Agenten Bestandteil des Systems. Diese erlauben neben der Ausführung der Systemfunktionalität auch Veränderungen der Systemstruktur und des Systemverhaltens.

Durch die Verbindung ökonomischer und evolutionärer Modelle mit Agententechnologie konnten wesentliche Punkte zur Lösung der Problemstellung beigetragen werden. Es wurde ein Verfahren entwickelt, welches Optimierung und Adaptivität aus einer Bottom-Up Perspektive ermöglicht. Trotz völliger Dezentralität von Problem und Informationen ist die Durchführung von Systemdiensten möglich. Das entstehende Gesamtsystem ist in wesentlichen Eigenschaften adaptiv, die primär als Reaktionen auf Umgebungszustände zu sehen sind. Hierbei wurden auf Modellebene folgende Beiträge zum Stand der Forschung erbracht:

- Die Integration ökonomischer Modelle in den Optimierungsansatz. Hierdurch wird **Geldfluß** durch das System, bestehend aus evolutionären Agenten ermöglicht. Die Umgebung stellt initial Geld bereit. Per Nachfrage nach Systemdiensten gelangt durch mikroökonomischen Kreislauf Geld in das System und fließt zwischen den Agenten.

- Einführung einer **lokalen Fitness**. Der Besitz von Geld wird mit Fitness gleichgesetzt und erlaubt die lokale Bewertung von Agenten. Durch die Erweiterung auf externe lokale Fitness wird eine Bewertung zwischen Agenten durch lokale Selektion ermöglicht.

- Einführung einer agentenspezifischer **Reproduktionsschwelle** (θ), die in Abhängigkeit des vorhandenen Geldes evolutionäre Reproduktion ermöglicht oder verhindert. Zur Bestimmung von θ wurden unterschiedliche Verfahren vorgeschlagen und experimentell getestet.

- Um die Anwendung auch in dynamischen und schwach vernetzten Strukturen zu zeigen, wurden Untersuchungen zur Informationsausbreitung (Takeover-time) durchgeführt und die Erweiterte Takeover Time (ETT) eingeführt. Hiermit wurde gezeigt dass der Selektionsdruck in dynamischen und schwach vernetzten Strukturen vergleichbar mit panmiktischen und parallelen (cEA,dEA) evolutionären Verfahren ist.

- Ein neues Dezentralitätsmaß wurde entwickelt, um verteilte evolutionäre Verfahren hinsichtlich ihres Grades an Dezentralität einzuordnen.

Aus dem Modell wurden die im Folgenden beschriebenen Eigenschaften abgeleitet und in Abschnitt 4 per Simulation bestätigt.

Adaptive Systemgröße: Durch Einführung der Reproduktionsschwelle θ erfolgt die Reproduktion lokal als Reaktion auf ausreichend vorhandenes Geld. Auf globaler Ebene, ist

die Populationsgröße damit eine beobachtbare emergente Eigenschaft als Reaktion auf Geld aus der Umgebung. Die Populationsgröße ist dann indirekt von der Nachfrage nach Systemdiensten abhängig. Ein System - genügend Hardware vorausgesetzt - kann immer mit der Nachfrage skalieren. Auf der anderen Seite wird ein System aus evolutionären Agenten nicht benötigte Dienste automatisch beenden und Ressourcen freigeben.

Adaptive Systemfunktionalität: Ein weiterer wichtiger Aspekt zur Adaptivität und Optimierung der Funktionalität stellt die Verbreitung erfolgreicher Strategien dar. Hierfür ist ebenfalls die Reproduktionsschwelle als Voraussetzung notwendig. Diese ermöglicht nach Hollands Schema-Theorem die Verbreitung fitter und damit angepasster Funktionalität im System. Damit erhöht sich die Anzahl nachgefragter Dienste automatisch zu Lasten weniger nachgefragter Dienste. Die Bezeichnung Dienst gilt hier sowohl für einzelne von Agenten angebotene Funktionen als auch für komplexe, z.B. durch Organisationen angebotene Dienste.

Adaptiver Selektionsdruck: Ein weiteres emergentes Merkmal in Form von adaptivem Selektionsdruck ergibt sich aus der veränderlichen Populationsgröße. Umgebungen, die unterhalb ihrer Tragfähigkeit genutzt werden, bieten bei entsprechender Nachfrage Expansions- und damit Explorationspotential. Agenten nutzen die Nachfrage in Form erhöhter Profite durch ihre Dienstleistungen aus. Eine gesteigerte Anzahl von Nachkommen zur Exploration des Suchraumes ist die Folge. Andererseits führt der limitierende Faktor der Umgebung zur Erhöhung des Selektionsdruckes durch den Wettbewerb um begrenzte Ressourcen. Damit vereint lokaler Selektionsmechanismus eine effiziente Umsetzung mit adaptivem Selektionsdruck. Zusätzlich ermöglicht dieses Verhalten ein implizites und damit selbst-adaptives Umschalten von Exploration auf Exploitation bei Erreichung der Tragfähigkeit. Dies erfolgt automatisch bei Erreichung der Tragfähigkeit und bedarf keiner zusätzlichen Kontrolllogik bzw. Überwachung. Ein effizienter Mechanismus zur Steuerung der Suchstrategie entsteht.

Theoretisch wurde eine Möglichkeit aufgezeigt, um Systeme evolutionärer Agenten anpassungsfähig auch für zukünftige Funktionalität zu machen. Prinzipiell lässt sich damit ein langlaufendes System realisieren, welches adaptiv Dienste anbietet und sich auf neue Entwicklungen der Nachfrage und der bereitgestellten Infrastruktur autonom einstellen kann. Die Realisierung eines solchen offenen Ansatzes geht jedoch weit über die Möglichkeiten dieser Arbeit hinaus und findet daher in der praktischen Umsetzung keine weitere Betrachtung.

Ein wichtiger Aspekt ist die Erstellung verteilter Lösungen. Hierbei besteht die Gesamtlösung des Problems der Systemoptimierung in der Kombination aller Individuallösungen und ihrer Interaktionen. Das System stellt damit eine Lösung für extern vorgegebene Anforderungen dar. Dieser dezentrale Ansatz ist nach aktuellem Wissensstand neu im Zusammenhang mit evolutionären Verfahren. Zur Einordnung im Vergleich mit panmiktischen und parallelen Verfahren wurde ein Dezentralitätsmaß entwickelt.

Die Vorhersagen aus der Modellanalyse wurden in zwei prototypischen Experimenten in Abschnitt 4 validiert. Die Evaluation evolutionärer Agenten und deren Anwendung auf Problemstellungen unterstreichen die Vorteile und Tragweite der Kombination evolutionärer und agentenbasierter Verfahren. Die grundlegende evolutionäre Funktionalität ist von der Funktionalität zur Dienstbereitstellung entkoppelt. Hierdurch ist das Einführen evolutionärer Agenten in praktischen Anwendungen relativ einfach möglich. Dennoch ist die Transferfunktion φ

der Schnittpunkt zwischen beiden Bereichen. In den vorgestellten Möglichkeiten der Anpassung von Transferfunktionen kann daher der einfachste Ansatz gewählt werden. Dies umfasst lediglich die Betrachtung einzelner Konfigurations- und Tuningparameter der bereitgestellten Dienste und lässt sich mit wenig Aufwand realisieren.

5.2 Ausblick

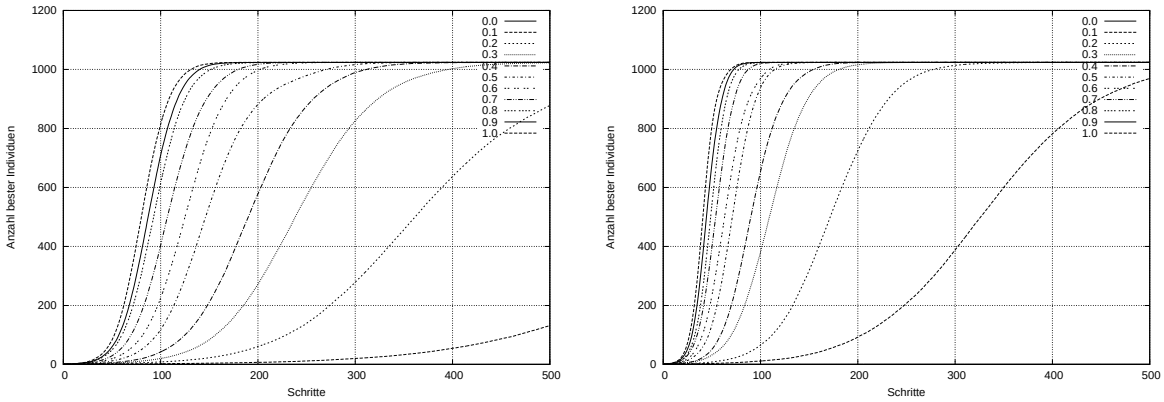
Die Entwicklung evolutionärer Agenten ist ein weiterer Schritt, um komplexe Systeme effizient zu managen im Hinblick auf ihre Adaptivität und Optimierung. Dies ist natürlich noch keine abschließende Lösung und auch kein ready-to-use Konzept für verteilte Systeme. Vielmehr wurden für einige Bereiche erste Lösungsansätze vorgeschlagen und validiert, während andere Bereiche lediglich theoretisch behandelt wurden. Im Fokus weiterer Entwicklungen auf dem Gebiet evolutionärer Agenten stehen neben der allgemeinen Erschließung neuer theoretischer und praktischer Konzepte, Methoden und Anwendungsbereiche besonders folgende Punkte.

- *Weiterentwicklung* des Ansatzes und weitere Integration und Analyse von Parametern zur Steuerung.
- *Theoriebildung und Formalisierung* im Schnittpunkt evolutionärer, ökonomischer und agentenbasierter Modelle. Hierzu gehört die Vervollständigung bestehender Konstrukte wie auch Ableitung und Entwicklung neuer Modelle. So kann zum Beispiel der in dieser Arbeit nicht betrachtete Faktor Vertrauen untersucht werden. Ebenso ist die Forschung im Bereich mehrstufiger Optimierungsverfahren (Bi- und Multilevel) ein weiterer Ansatz.
- *Methodenentwicklung* im Bereich Bottom-Up Analyse, Design und Realisierung und Abgleich mit bestehenden agentenorientierten Ansätzen.
- Ausbau des Konzeptes als Ansatz für *Lebenszyklusmanagement* elektronischer Dienste. Durch die Weiterentwicklung von Diensten sowie deren selbständiger Entstehung und Beendigung anhand von Nachfrage ist eine (Teil)Automatisierung denkbar.
- Im Bereich von Grid-Computing bzw. einer globalen Infrastruktur sind Dienste als selbständig agierende Wirtschaftssubjekte mit dem Konzept evolutionärer Agenten denkbar. Als *Geschäftsmodell* platzieren Hersteller ihre Dienste in der Infrastruktur und diese reichen Teile ihres Profites an die Hersteller weiter. Bei Erfolg findet automatisch die Ausweitung der Geschäftstätigkeit durch Anpassung der Anzahl statt.
- Die Adaptivität der Transferfunktion evolutionärer Agenten erlaubt die autonome Entwicklung neuer Dienste sowie die Entstehung komplexer kollaborativer Dienste. In diesem Zusammenhang kann konzeptuell von einer virtuellen *evolutionären Fabrik* gesprochen werden. Diese sorgt - entsprechende Infrastruktur vorausgesetzt - für nachfrageorientierte Entwicklung neuer Dienste und stellt diese bereit.
- Die Anwendung evolutionärer Agenten im Bereich *Computerspiele* zur Weiterentwicklung computergenerierter Spieler.

Anhang A

Versuchsergebnisse Takeover Time

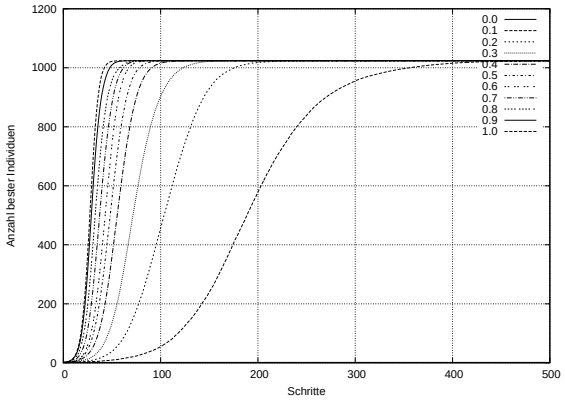
- Graphgröße: 1024
- Anzahl Kanten 100 - 1024
- Wahrscheinlichkeit für Kantenänderung 0.0 - 1.0
- Gerichtete Graphen
- Durchschnittsbildung über jeweils 50 Simulationen



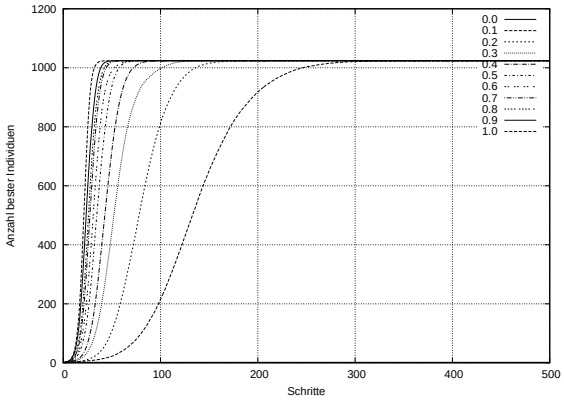
(a) 100 Kanten

(b) 200 Kanten

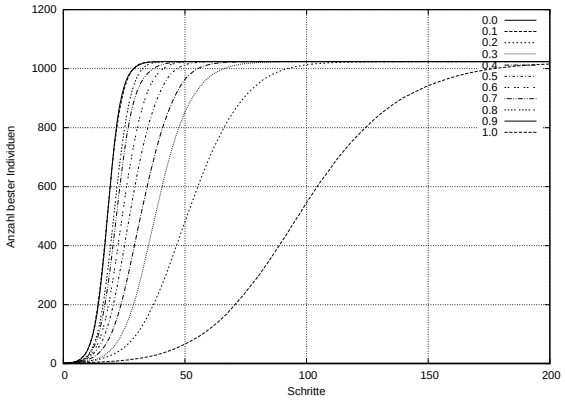
Abbildung A.1: Takeover times für 100 und 200 Kanten und Änderungswahrscheinlichkeiten für Kanten von $0.0 - 1.0$ für 1024 Knoten



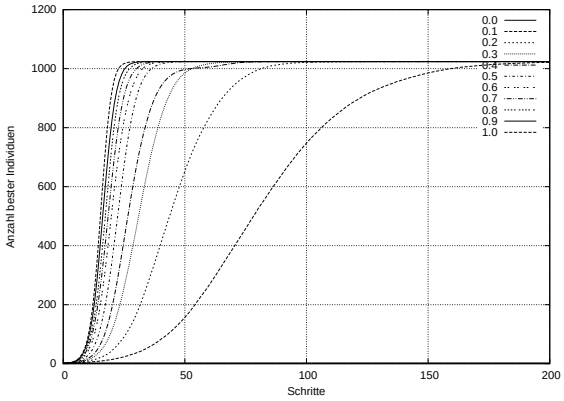
(c) 300 Kanten



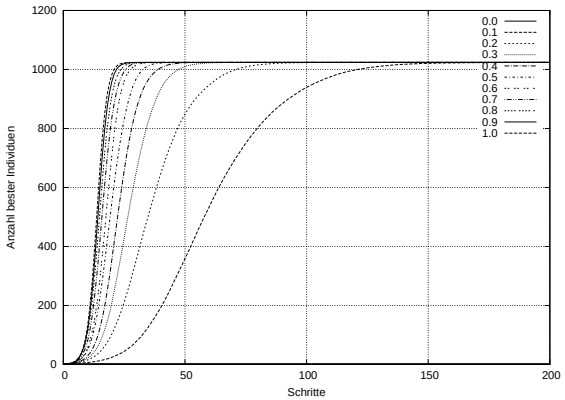
(d) 400 Kanten



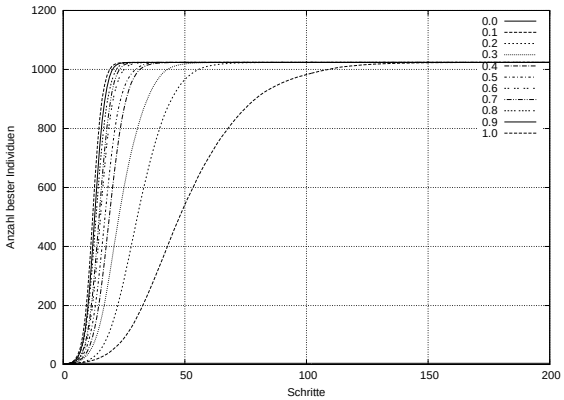
(e) 512 Kanten



(f) 600 Kanten

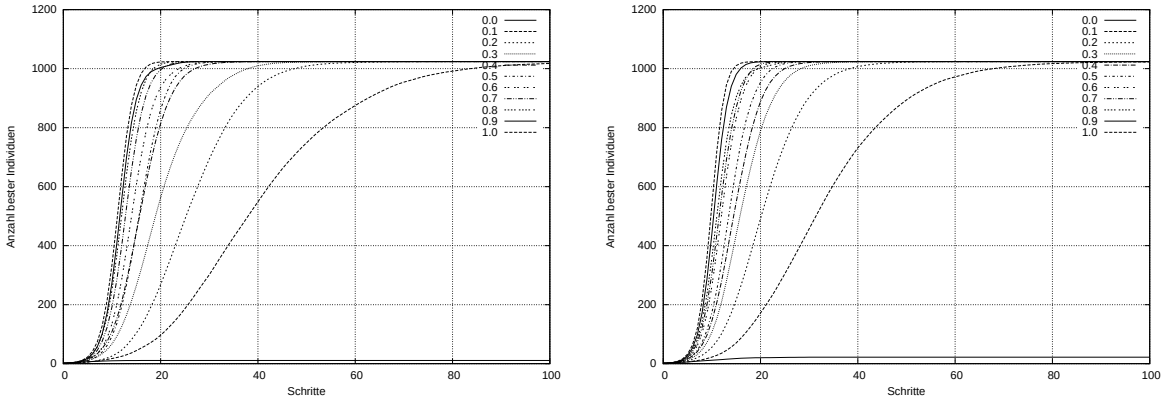


(g) 700 Kanten



(h) 800 Kanten

Abbildung A.1: Takeover times für 300 bis 800 Kanten und Änderungswahrscheinlichkeiten für Kanten von $0.0 - 1.0$ für 1024 Knoten (Forts.)



(i) 900 Kanten

(j) 1024 Kanten

Abbildung A.1: Takeover times für 900 und 1024 Kanten und Änderungswahrscheinlichkeiten für Kanten von $0.0 - 1.0$ für 1024 Knoten (Forts.)

Anhang B

Versuchsergebnisse Adaptive Logistiknetzwerke

Für die nachfolgend aufgeführten Simulationen gelten die gleichen Bedingungen wie in Abschnitt 4.2.2 aufgeführt, soweit nichts anderes beschrieben wird.

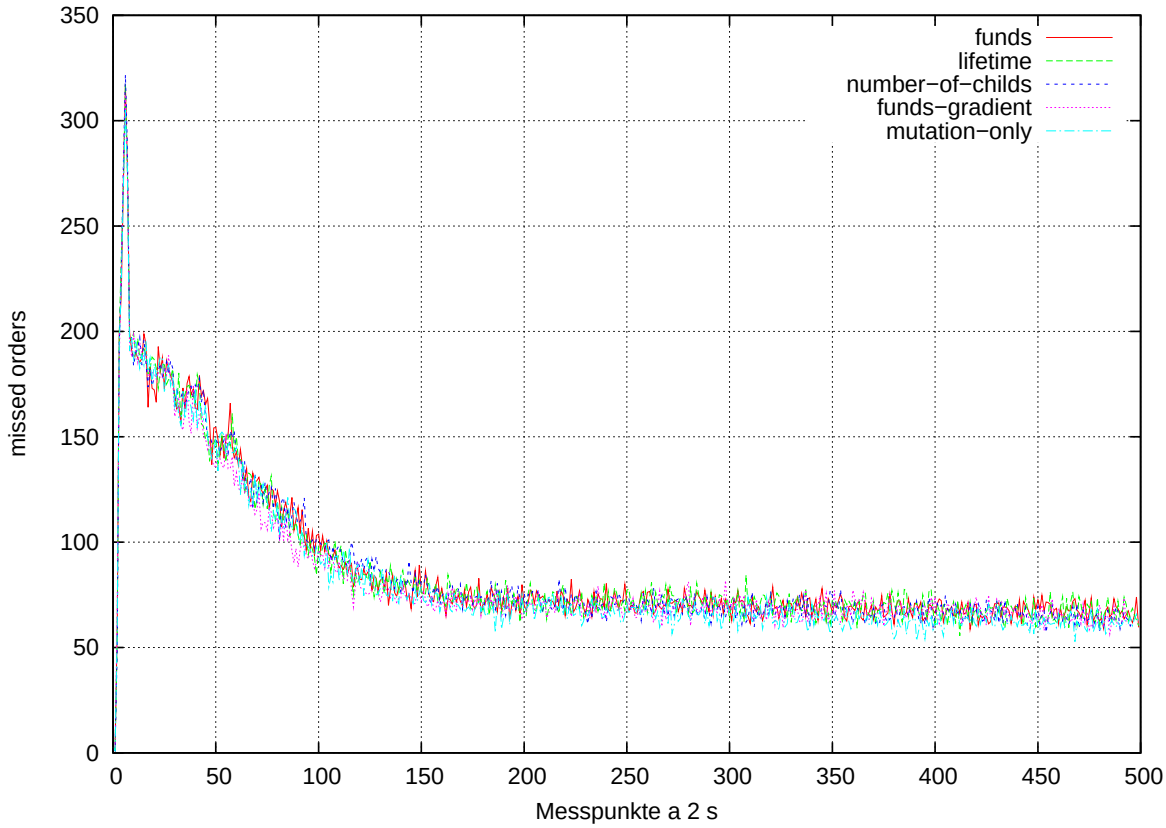


Abbildung B.1: Unterschiedliche lokale Selektionsmethoden

Abbildung B.1 zeigt missed orders für unterschiedliche externe lokale Fitnessberechnungen. Die einzelnen Methoden zur Berechnung der externen lokalen Fitness sind wie folgt:

funds: Der Wert von $funds$ wird einfach genutzt (höher ist besser).

lifetime: Anstelle des Geldes wird die bisherige Lebensdauer des Agenten verwendet (höher ist besser).

nr-of-childs: Die Anzahl der bislang erzeugten Kindagenten wird verwendet (höher ist besser).

funds-gradient: Der durchschnittliche Geldbetrag über die letzten x Zeitschritte wird verwendet (höher ist besser).

mutation-only: keine Selektion, sondern nur Mutation wird verwendet.

Abbildung B.2 verwendet initial 50 Transportagenten und vergleicht die Entwicklung der Transportkapazität mit jeweils zwei unterschiedliche externen lokalen Fitnessberechnungen.

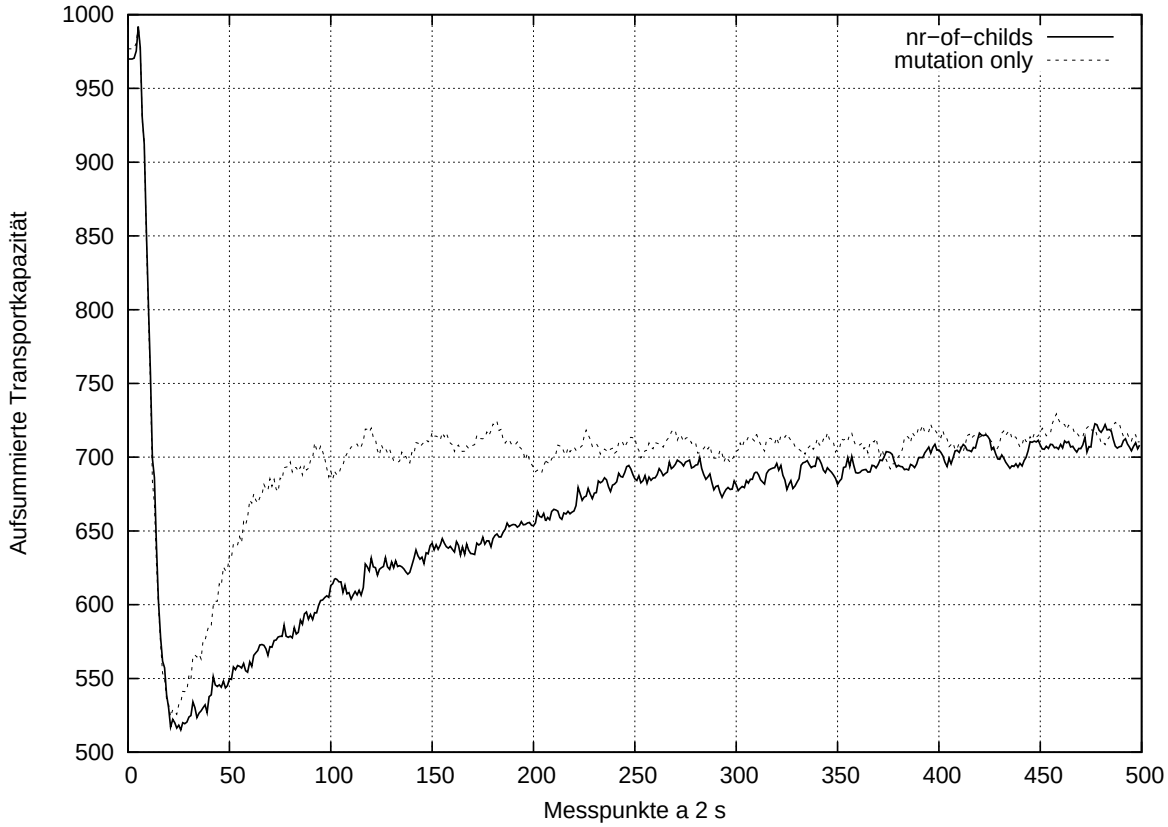


Abbildung B.2: Entwicklung der Transportkapazität bei initital 50 Transportagenten für zwei unterschiedliche externe lokale Fitnessberechnungsmethoden

Anhang C

Versuchsergebnisse Facility Location in Mixed Mode Environments

Für simulationsspezifische Details siehe Abschnitt 4.3.3.

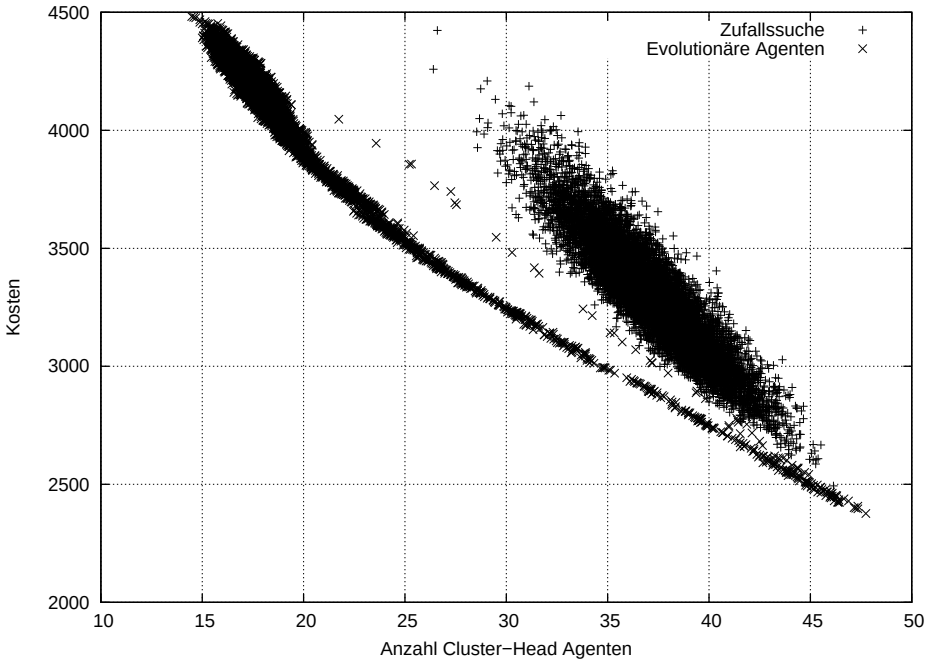


Abbildung C.1: Vergleich gemittelte Systemzustände 'Anzahl Cluster-Head Agenten' zu Gesamtkosten für evolutionäre Agenten und Zufallssuche (Galvao)

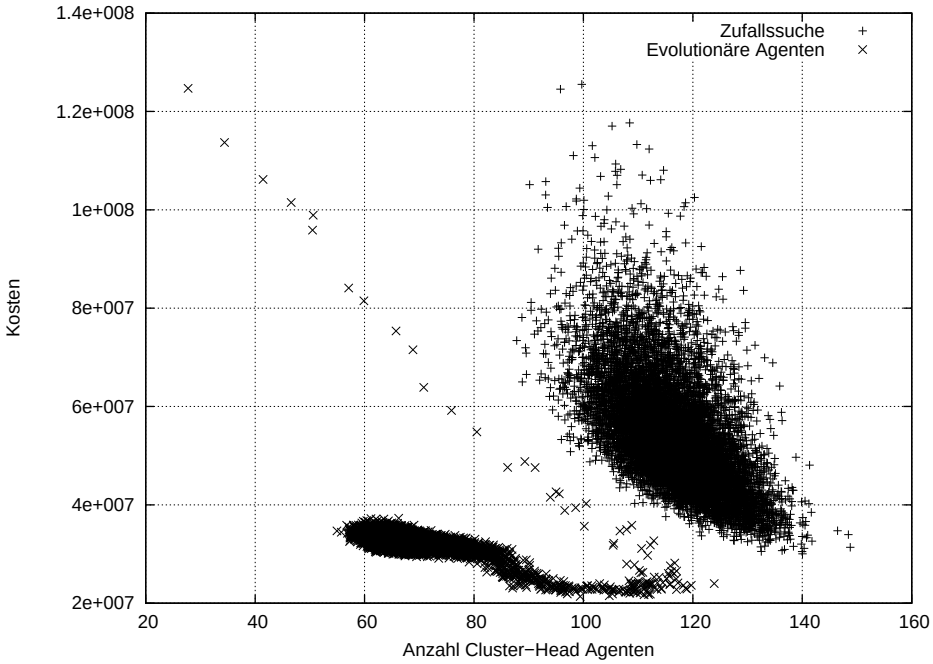


Abbildung C.2: Vergleich gemittelte Systemzustände 'Anzahl Cluster-Head Agenten' zu Gesamtkosten für evolutionäre Agenten und Zufallssuche (Alberta)

Anhang D

Implementationsdetails

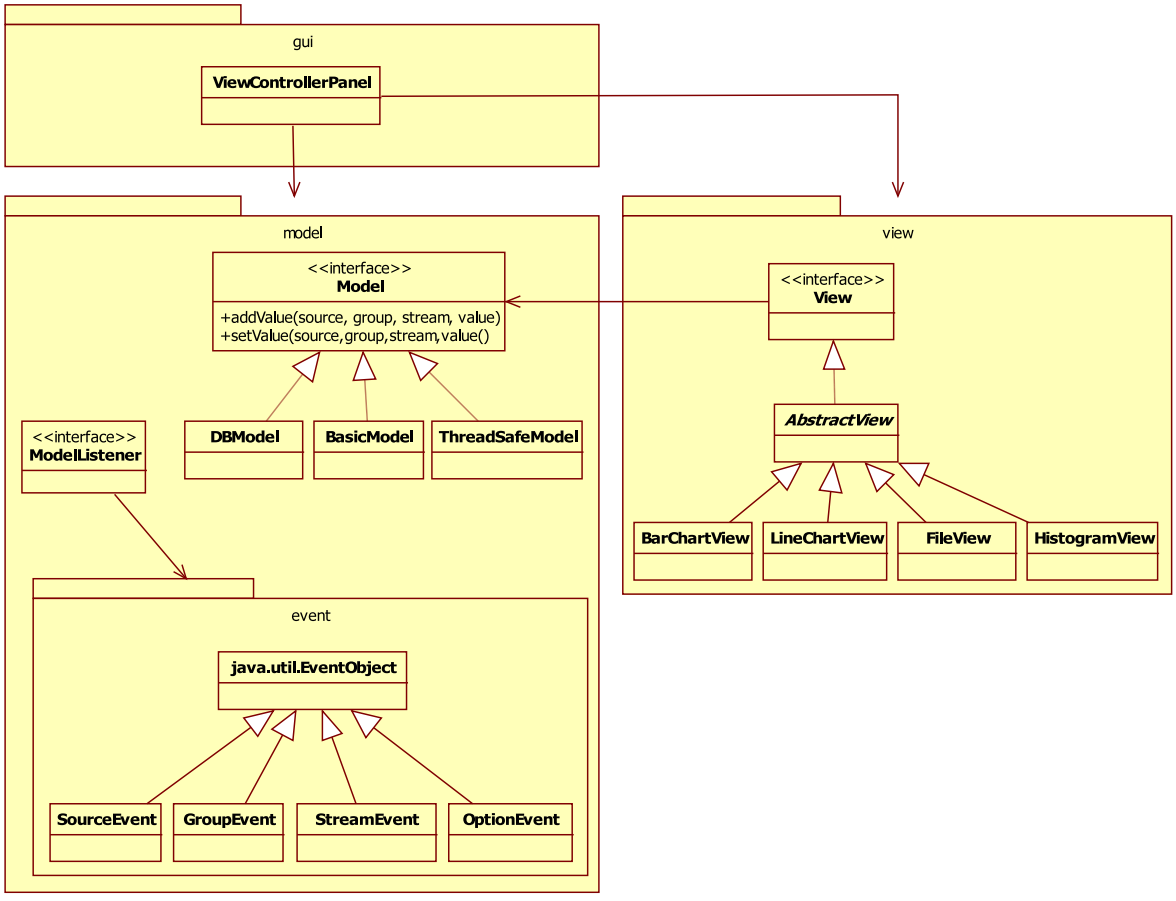


Abbildung D.1: Klassendiagramm der JStreaMon API

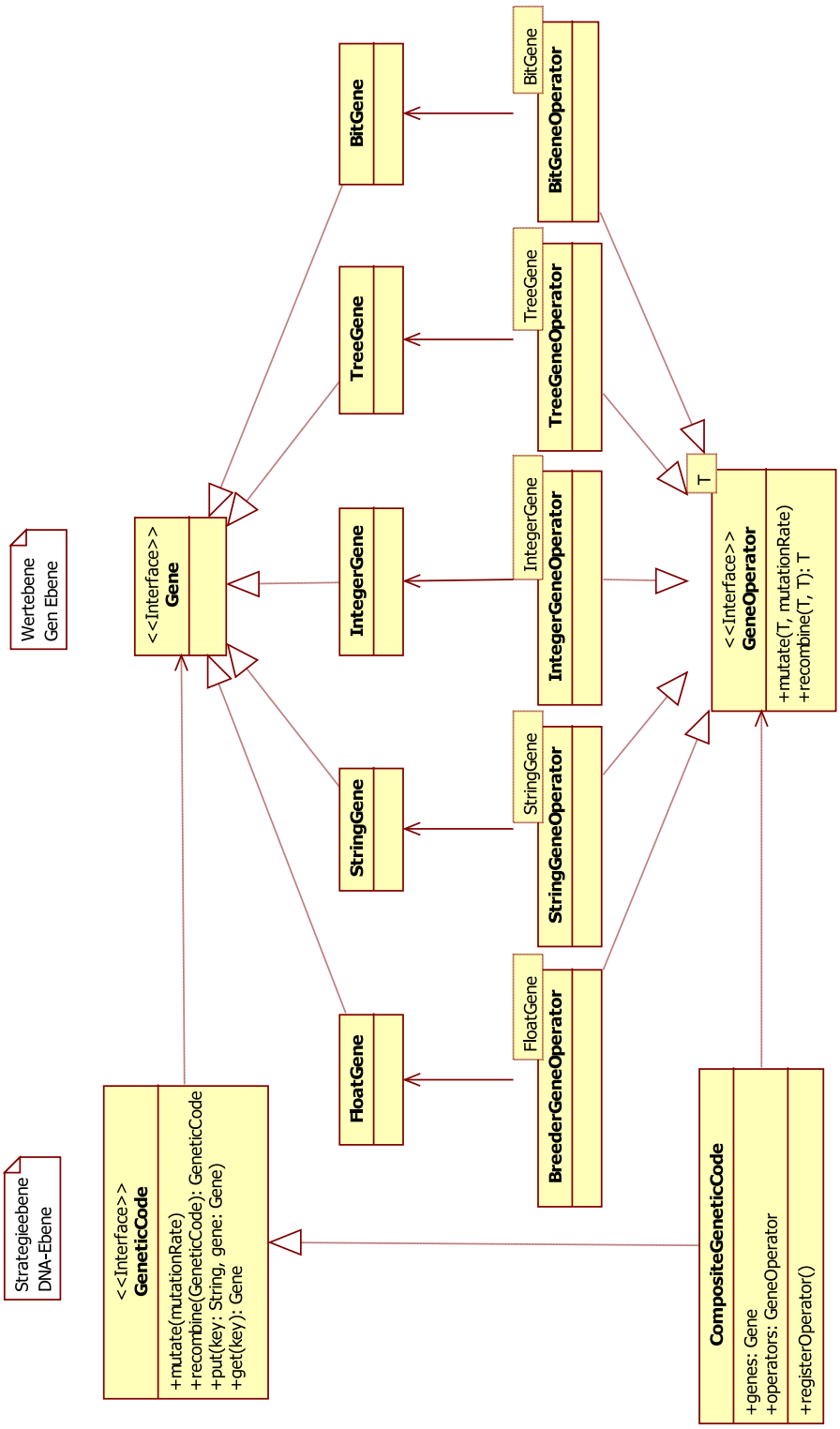


Abbildung D.2: Klassendiagramm der API für evolutionäre Funktionalität

Notationen und Abkürzungen

$\mathcal{P}_a(t)$	Zahlungen von Agent a an andere Agenten zum Zeitpunkt t
$\mathcal{R}_a(t)$	Zahlungseingänge von Agent a zum Zeitpunkt t
$\mathcal{T}_a(t)$	Steuern für Agent a zum Zeitpunkt t
$\mathcal{Z}_i$	Dezentralitätsmaß für Anteil an der Gesamtlösung durch einen Prozessor i
Φ	von MAS ausgeführte Funktion als Komposition der Transferfunktionen von Agenten φ
$\pi_a(t)$	Profit von Agent a zum Zeitpunkt t
θ	Reproduktionsschwelle
θ_a	Reproduktionsschwelle eines Agenten a
υ	Anteil einer Teillösung an Gesamtlösung
φ	Transferfunktion eines Agenten
a	Agent
A_{sub}	Subpopulation
Akt_a	Menge der Aktionen eines Agenten
$control_a(o)$	Funktion, die wahr ausgibt, wenn Agent a Objekt o alleine zugreifen kann (besitzt)
$cost$	Kostenfunktion
$costsys(MAS, t)$	Gesamtkosten, die in einem MAS zum Zeitpunkt t anfallen.
$ETT(G_c)$	erweiterte Takeover time, Erwartungswert für Größe des übernommenen Subgraphen
$ETT(p)$	erweiterte Takeover time, Wahrscheinlichkeit für Übernahme eines Individuums
$ETT(T)$	Erweiterte Takeover time, Erwartungswert der Zeit bis zum Stillstand der Übernahme

$funds(a)$	Geld, welches Agent a zur Verfügung steht
$G_U(V,E)$	Struktur der Agentenumgebung als Graph mit einer Menge Knoten V und Kanten E
$K(x)$	Kolmogorov Komplexität von x
MAS	Multiagentensystem
O	Menge der Objekte
o	Objekt in einer Agentenumgebung
s	Strategie eines Agenten
S_A	Strategie des Multiagentensystems, siehe MAS
Sen_a	für Agent a wahrnehmbarer Teil der Umgebung
U	Agentenumgebung
$vis(a)$	Sichtbarkeit der Umgebung für Agent a
VOP	Verteiltes Optimierungsproblem
z	Innerer Zustand

Die nachfolgenden mathematischen Grundbegriffe sind allgemeiner mathematischer Konsens und finden sich z.B. in [69, 213].

Sei $X := \{x_1, x_2, \ldots, x_n\}$ eine Menge X mit den Elementen $x_1, x_2, \ldots, x_n \in X$. Eine alternative Schreibweise ist $X = \{x_1, x_2, \ldots, x_n\}$. Die Reihenfolge der Elemente $x_i \in X$ ist nicht definiert. Die leere Menge ist $X = \{\emptyset\}$. Die Mächtigkeit einer Menge $|X|$ bezeichnet die Anzahl der Elemente in X. Sind X und Y Mengen, so steht $f : X \to Y$ für die Abbildung zwischen den Mengen und $x \mapsto f(x)$ für eine Abbildung zwischen den Elementen. Eine Teilmenge von X ist definiert durch $X' \subseteq X$, wenn aus $x \in X'$ immer $x \in X$ folgt. Mit $X'' \subset X$ sei eine echte Teilmenge bezeichnet, so dass gilt $\exists x \in X \mid x \notin X''$. Die Potenzmenge von X wird folgendermaßen definiert $\mathcal{P}(X) = \{U \mid U \subseteq X\}$ und bezeichnet die Menge aller Teilmengen von X. Die Mächtigkeit einer Potenzmenge $\mathcal{P}(X)$ hat genau $2^{|X|}$ Elemente.

Graphentheorie

Definition 19 (Graph)**.**
Sei $G = (V, E)$ ein Graph, bestehend aus einer Menge von Knoten V und einer Menge von Kanten $E \subseteq \{(u, v) | u, v \in V\}$. Ist G ungerichtet, dann ist die Adjazenzrelation definiert durch die Kanten symmetrisch, bzw. $E \subseteq \{\{u, v\} | u, v \in V\}$ und für jede Kante $e \in E$ gilt $\{u, v\} = \{v, u\}$.

Definition 20 (Subgraph)**.**
Ein Graph $G' = (V', E')$ ist ein Subgraph oder Teilgraph eines anderen Graphen $G = (V, E)$, wenn

- *$V' \subseteq V$ und*

- $E' \subseteq E$ *und*
- $(v_1, v_2) \in E' \rightarrow v_1, v_2 \in V'$

Definition 21 (Pfad)**.**
Sei $G = (V, E)$ ein Graph und $P = (v_1, \ldots, v_n)$ eine Folge von Knoten aus V mit der Eigenschaft, dass für alle i aus $\{1, \ldots, n-1\}$ gilt:

- *Die Menge $\{v_i, v_{i+1}\}$ ist Element von E, falls G ein ungerichteter Graph ist*
- *Das Paar (v_i, v_{i+1}) ist Element von E, falls G ein gerichteter Graph ist*
- *$\forall i, j \in \{1, \ldots, n\}$ gilt: $v_i \neq v_j \wedge i \neq j$.*

Sei P ein Pfad der Länge n, dann wird $v_1 \in P$ als *Startknoten* und $v_n \in P$ als *Endknoten* bezeichnet. Ein *Zyklus* liegt genau dann vor, wenn Start- und Endknoten von P identisch sind: $v_1 = v_n$. Ein Pfad der Länge 1 bzw. eine Kante mit identischen Start und Endknoten $e = (v, v)$ wird auch als *Schlinge* bezeichnet.

Ein *verbundener Graph* ist ein ungerichteter Graph mit mindestens einem Pfad zwischen jedem Knotenpaar. Ein Subgraph $G' \subseteq G$ ist dann maximal verbunden, wenn es keinen Knoten aus g mehr gibt, der zu G' hinzugefügt werden kann ohne die Konnektivität zu zerstören. Mehr formal lautet die Definition:

Definition 22 (Maximal verbundener Subgraph)**.**
Gegeben sei ein Graph $G = (V, E)$. Ein Teilgraph $G' = (V', E')$ ist maximal verbunden, wenn

- *G' ist verbunden und*
- *$\forall v \in V \wedge v \notin V'$ existiert kein Knoten $v' \in V' | (v, v') \in E$.*

Abbildungsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] D. Ackley and M. Littman. Interactions between learning and evolution. In C. G. Langton, C. Taylor, D. J. Farmer, and S. Rasmussen, editors, *Proceedings of the Workshop on Artificial Life (ALIFE '90)*. Reading, MA: Addison-Wesley, 1991.

[2] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, Hoboken, New Jersey, USA, 2005.

[3] E. Alba and G. Luque. Growth Curves and Takeover Time in Distributed Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference (GECCO-04), Seattle,WA, USA, June 26-30, 2004 Proceedings, Part I*, volume 3102, pages 864–876, 2004.

[4] M. Alba, E.; Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.

[5] O. Alp, Z. Drezner, and E. Erkut. An efficient genetic algorithm for the p-median problem. *Annals of Operations Research*, 122(1-4):21–42, 9 2003.

[6] R. Anane, K.-M. Chao, and Y. Li. Hybrid composition of web services and grid services. In *EEE '05: Proceedings of the 2005 IEEE International Conference on E-Technology, E-Commerce and E-Service*, pages 426–431, Washington, DC, USA, 2005. IEEE Computer Society.

[7] H. Arthur, J. Daniel, and B. Alejandro. A3ME - an agent-based middleware approach for mixed mode environments. In *The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2008)*, pages 191–196, Valencia, Spain, Oct 2008. IEEE Computer Society Press.

[8] A. Babanov, W. Ketter, and M. Gini. An evolutionary approach for studying heterogeneous strategies in electronic markets. In G. D. M. Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors, *Enigneering Self-Organising Systems*, 2977, pages 157–168. LNAI, Springer-Verlag Berlin Heidelberg, 2004.

[9] J. E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[10] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 14–21, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[11] M. Baker. Cluster computing white paper. Internet: http://www.sai.syr.edu/ chapin/papers/pdf/ieee_tfcc_wp_clustercomputing.pdf, 2000.

[12] M. Baker and R. Buyya. Cluster computing: the commodity supercomputer. *Software - Practice and Experience*, 29(6):551–576, 1999.

[13] S. Baliga and E. Maskin. Mechanism design for the environment. In K. G. Mäler and J. R. Vincent, editors, *Handbook of Environmental Economics*, volume 1 of *Handbook of Environmental Economics*, chapter 7, pages 305–324. Elsevier, 2003.

[14] H. Balzert. *Lehrbuch der Software-Technik: Software-Entwicklung*. Lehrbücher der Informatik. Spektrum Akademischer Verlag, Heidelberg, 2nd edition, 2001.

[15] W. Banzhaf. Population processing - a powerful class of parallel algorithms. *BioSystems*, 22:163–172, 1989.

[16] Y. Bar-Yam. *Dynamics of Complex Systems*. Westview Press, July 2003.

[17] A. Barros and M. Labbe. A general model for the uncapacitated facility and depot location problem. Papers 9253-a, Erasmus University of Rotterdam - Econometric Institute, 1992.

[18] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996.

[19] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol, UK, 1999.

[20] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa-compliant agent framework. Technical report, Telecom Italia Lab, April 1999.

[21] A. D. Bethke. Comparison of genetic algorithms and gradient-based optimizers on parallel processors: Efficiency of use of processing capacity. Technical report, University of Michigan, Ann Arbor, Logic of Computers Group, 1976.

[22] H.-G. Beyer and H.-P. Schwefel. Evolution strategies –a comprehensive introduction. *Natural Computing: an international journal*, 1(1):3–52, 2002.

[23] T. Blickle. *Theory of Evolutionary Algorithms and Application to System-Synthesis*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1996. ETH diss no. 11894.

[24] F. A. Board. FIPA Communicative Act Library Specification. Technical report, Foundation for Intelligent Physical Agents, 2000.

[25] F. A. Board. Fipa abstract architecture specification. Technical report, Foundation for Intelligent Physical Agents, 2002.

[26] F. A. Board. FIPA ACL Message Structure Specification. Technical report, Foundation for Intelligent Physical Agents, 2002.

[27] F. A. Board. Fipa agent message transport service specification. Technical report, Foundation for Intelligent Physical Agents, 2002.

[28] F. A. Board. Fipa agent management specification. Technical report, Foundation for Intelligent Physical Agents, 2004.

[29] E. Bonabeau. Editor's introduction: Stigmergy. *Artificial Life*, 5(2):95–96, 1999.

[30] L. B. Booker. *Intelligent behavior as an adaptation to the task environment*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1982.

[31] A. M. Brandenburger and B. J. Nalebuff. *Coopetition: kooperativ konkurrieren - Mit der Spieltheorie zum Geschäftserfolg*. Christian Rieck Verlag Eschborn, 2007.

[32] J. Branke, H. C. Andersen, and H. Schmeck. Global selection methods for massively parallel computers. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 175–188, London, UK, 1996. Springer-Verlag.

[33] J. Branke, A. Kamper, and H. Schmeck. Distribution of evolutionary algorithms in heterogeneous networks. In *Genetic and Evolutionary Computation Conference*, volume 3102 of *LNCS*, pages 923–934. Springer, 2004.

[34] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

[35] A. Brindle. *Genetic Algorithms for Function Optimization*. PhD thesis, University of Alberta, Edmonton, Canada, 1981.

[36] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[37] R. A. Brooks and P. Maes, editors. *Artificial life IV: Proceedings of the fourth international workshop on the synthesis and simulation of living systems*. MIT Press, Cambridge, 1994.

[38] J. Buford, H. Yu, and E. K. Lua. *P2P Networking and Applications*. Morgan Kaufmann, 2008.

[39] P. Buhler and J. M. Vidal. Enacting BPEL4WS specified workflows with multiagent systems. In *Proceedings of the Workshop on Web Services and Agent-Based Engineering*, 2004.

[40] P. Buhler and J. M. Vidal. Integrating agent services into BPEL4WS defined workflows. In *Proceedings of the Fourth International Workshop on Web-Oriented Software Technologies*, 2004.

[41] P. Buhler, J. M. Vidal, and H. Verhagen. Adaptive workflow = web services + agents. In *Proceedings of the International Conference on Web Services*, pages 131–137. CSREA Press, 2003.

[42] E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[43] E. Cantú-Paz and D. E. Goldberg. Modeling idealized bounding cases of parallel genetic algorithms. In J. Koza, K. Deb, M. Dorigo, D. Fogel, M. Garzon, H. Iba, and R. Riolo, editors, *Genetic Programming: Proceedings of the Second Annual Conference*, pages 353–361. Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[44] E. Cantú-Paz and D. E. Goldberg. Predicting speedups of idealized bounding cases of parallel genetic algorithms. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 113–120. Morgan Kaufmann, 1997.

[45] U. K. Chakraborty, K. Deb, and M. Chakraborty. Analysis of selection algorithms: A markov chain approach. *Evol. Comput.*, 4(2):133–167, 1996.

[46] F. S. Chong and W. B. Langdon. Java based distributed genetic programming on the internet. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1229, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann. Full text in technical report CSRP-99-7.

[47] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F. J. Varela and P. Bourgine, editors, *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 134–142. MIT Press, Cambridge, MA, 1992.

[48] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[49] C. Darwin. *On the Origin of Species*. John Murray, London, 1859.

[50] Y. Davidor, T. Yamada, and R. Nakano. The ECOlogical Framework II: Improving GA Performance At Virtually Zero Cost. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 171–176, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[51] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Distributed Artificial Intelligence*, 20(1):333–356, January 1988.

[52] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 2001.

[53] G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli. *Enigneering Self-Organising Systems Nature-Inspired Approaches to Software Engineering*. Springer-Verlag Berlin Heidelberg, 2004.

[54] M. Dorigo and C. Blum. Ant colony optimization theory: a survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005.

[55] E. H. Durfee. Distributed problem solving and planning. In J. G. Carbonell and J. Siekmann, editors, *Multi-agent systems and applications*, pages 118–149. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[56] E. H. Durfee. Distributed problem solving and planning. In G. Weiss, editor, *Multiagent Systems*, pages 121–165. The MIT Press, Cambridge, Massacusetts, London, England, 2001.

[57] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, 1989.

[58] E. H. Durfee and J. S. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. In *The Thirteenth International Distributed Artificial Intelligence Workshop*, pages 94–104, Seattle, Washington, July 1994.

[59] A. Eiben and M. C. Schut. New ways to calibrate evolutionary algorithms. In Z. Michalewicz and P. Siarry, editors, *Advanced in Metaheuristics and Optimization*, pages 153–178. Springer, 2007.

[60] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. SpringerVerlag, Berlin Heidelberg, 2003.

[61] G. Eiben and M. C. Schut. New ways to calibrate evolutionary algorithms. In P. Siarry and Z. Michalewicz, editors, *Advances in Metaheuristics for Hard Optimization*, Natural Computing Series, pages 153–177. Springer, 2008.

[62] L. M. Ellram. A structured method for applying purchasing cost management tools. *International Journal of Purchasing and Materials Management*, 32(1):11–20, 1996.

[63] P. Erdős and A. Rényi. *On Random Graphs. I.*, chapter 6, pages 290–297. Publicationes Mathematicae. Debrecen, 1959.

[64] T. Eymann. *AVALANCE - Ein agentenbasierter dezentraler Koordinationsmechanismus für elektronische Märkte*. PhD thesis, Universität Freiburg, 2000.

[65] T. Eymann, B. Padovan, and D. Schoder. Simulating value chain coordination with artificial life agents. *Multi-Agent Systems, International Conference on*, 0:423, 1998.

[66] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, February 1999.

[67] I. A. Ferguson. Touringmachines: Autonomous agents with attitudes. *IEEE Computer*, 25(5):51–55, 1992.

[68] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, S. Shapiro, C. Beck, and R. Pelavin. Specification of the KQML Agent-Communication Language. Technical report, The DARPA Knowledge Sharing Initiative, June 1993.

[69] G. Fischer. *Lineare Algebra*. Vieweg Studium, Grundkurs Mathematik. Friedr.Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 11 edition, 1997.

[70] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, 1972.

[71] M. Förster, B. Bickel, H. Bernd, and G. Kókai. Self-adaptive ant colony optimisation applied to function allocation in vehicle networks. In H. Lipson, editor, *Proceeding Genetic and Evolutionary Computation Conference (GECCO-2007)*, pages 1991 – 1998, 2007.

[72] I. Foster. What is the grid? a three point checklist. http://www-fp.mcs.anl.gov/ foster/Articles/WhatIsTheGrid.pdf, 2002.

[73] I. Foster. Globus toolkit version 4: Software for service-oriented systems. *IFIP International Conference on Network and Parallel Computing*, pages 2–13, 2006.

[74] I. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. *AAMAS Autonomous Agents and Multi-Agent Systems, New York, NY, USA*, July 2004.

[75] I. Foster and C. Kesselman. Computational grids. In *IFIP International Conference on Network and Parallel Computing*, pages 15–52. Morgan Kaufmann, 1998.

[76] I. Foster and C. Kesselman. *The grid: blueprint for a new computing infrastructure.* Morgan Kaufmann, 2004.

[77] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[78] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science, Springer Berlin / Heidelberg*, 15(3), 2001.

[79] S. C. Frey and M. M. Schlosser. Abb and ford: Creating value through cooperation. *Sloan Management Review*, 35(1):65–72, 1993.

[80] B. D. Gabriel Luque, Enrique Alba. Parallel genetic algorithms. In *Parallel Metaheuristics: A New Class of Algorithms*, pages 107–126. Wiley-Interscience, Hoboken, New Jersey, USA, 2005.

[81] R. D. Galvão and C. ReVelle. A lagrangean heuristic for the maximal covering location problem. *European Journal of Operations Research*, 88:114–123, 1996.

[82] M. Genesereth and N. Nilsson. *Logical Fundations of AI.* Morgan Kaufmann, 1987.

[83] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[84] M. Giacobini, M. Tomassini, and A. Tettamanzi. Takeover time curves in random and small-world structured populations. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1333–1340, New York, NY, USA, 2005. ACM.

[85] M. Giacobini, M. Tomassini, A. Tettamanzi, and E. Alba. Selection intensity in cellular evolutionary algorithms for regular lattices. *IEEE Trans. Evolutionary Computation*, 9(5):489–505, 2005.

[86] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[87] D. Goldberg, K. Deb, and D. Thierens. Toward a better understanding of mixing in genetic algorithms. In *Journal of the Society of Instrument and Control Engineers*, volume 32 of *1*, pages 10–16, 1993.

[88] D. E. Goldberg. *Genetic Algorithms in search, optimization, and machine learning.* Addison Wesley Longmann, Inc., 1989.

[89] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[90] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.

[91] M. Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 150–159, London, UK, 1991. Springer-Verlag.

[92] M. Gorges-Schleuter. A comparative study of global and local selection in evolution strategies. In *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*, pages 367–377, 1998.

[93] M. Gorges-Schleuter. An analysis of local selection in evolution strategies. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 847–854, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[94] I. Göpfert. *Logistik der Zukunft- Logistics for the Future.* Gabler Verlag, 2008.

[95] H.-W. Graf. *Netzstrukturplanung - Ein Ansatz zur Optimierung von Transportnetzen.* PhD thesis, Universität Dortmund, Fakultät Maschinenbau, Dortmund, 2000.

[96] M. Grassmann. GM prüft Abschied von Hummer. *Financial Times Deutschland, 4. Juni*, pages 3–3, 2008.

[97] V. Graudina and J. Grundspenkis. Technologies and multi-agent system architectures for transportation and logistics support: An overview. *International Conference on Computer Systems and Technologies - CompSysTech, Varna, Bulgaria*, pages IIIA.6–1 – IIIA.6–6, 2005.

[98] S. Graupner, N. Cook, D. Coleman, and T. Nitzsche. Management middleware for enterprise grids. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 625–632, Washington, DC, USA, 2006. IEEE Computer Society.

[99] J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Vanderbilt University, Nashville, TN, 1981.

[100] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[101] G. Görz and B. Nebel. *Künstliche Intelligenz*. Fischer Taschenbuch Verlag, Frankfurt/Main, 2003.

[102] K. Hadeli, P. Valckenaers, M. Kollingbaum, and H. V. Brussel. Multi-agent coordination and control using stigmergy. *Computers in Industry*, 53(1):75–96, January 2004.

[103] G. Harik, D. E. Goldberg, E. Cantú-paz, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *Evolutionary Computation*, pages 7–12. IEEE Press, 1997.

[104] U. Hasenkamp, S. Kirn, and M. Syring. *CSCW - Computer Supported Cooperative Work*. Addison-Wesley, 1994.

[105] C. Haubelt. *Automatic Model-Based Design Space Exploration for Embedded Systems – A System Level Approach*. PhD thesis, University of Erlangen-Nuremberg, Germany, Berlin, July 2005.

[106] T. D. Haynes and R. L. Wainwright. A simulation of adaptive agents in a hostile environment. *Proceedings of the 1995 ACM Symposium on Applied Computing*, pages 318–323, 1995.

[107] H. Heilmann, R. Alt, and H. Österle. *Virtuelle Organisationen*. Dpunkt Verlag; Auflage: 1, 2005.

[108] C. Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, 1986.

[109] R. V. D. Hofstad. Random graphs and complex networks. http://www.win.tue.nl/~rhofstad/NotesRGCN2008.pdf, 2008.

[110] J. H. Holland. Outline for a logical theory of adaptive systems. *J. ACM*, 9(3):297–314, 1962.

[111] J. H. Holland. Adaptation in natural and artificial systems. Technical report, Unversity of Michigan, The University of Michigan Press, Ann Arbor, USA, 1975.

[112] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[113] J. H. Holland. *Hidden Order: How Adaptation Builds Complexity*. Addison Wesley Publishing Company, 9 1996.

[114] P. T. Hraber, T. Jones, and S. Forrest. The ecology of echo. *Artificial Life*, 3(3):165–190, 1997.

[115] I. Hörmann. Karten auf den Tisch. *Technology Review*, 1:89, January 2009. Interview mit Andreas Hoffjan.

[116] M. N. Huhns and L. M. Stephens. Multiagent systems and societies of agents. In G. Weiss, editor, *Multiagent systems: a modern approach to distributed artificial intelligence*, pages 79–120. MIT Press, Cambridge, MA, USA, 1999.

[117] C. Jacob, J. Rehder, J. Siemandel, and A. Friedmann. Xneurogene: a system for evolving artificial neural networks. *Modeling, Analysis, and Simulation of Computer Systems, International Symposium on*, 0:329, 1995.

[118] C. Jäger and B. Boucke. Strukturen und Typen - Ausrichtung der Organisationsstruktur. In *Vom Fraktal zum Produktionsnetzwerk: Unternehmenskooperationen erfolgreich gestalten*, pages 91–121. Springer, Berlin, 1999.

[119] W. Jaśkowski, K. Krawiec, and B. Wieloch. Winning ant wars: Evolving a human-competitive game strategy using fitnessless selection. In *Genetic Programming 11th European Conference, EuroGP 2008, Proceedings*, volume 4971, pages 13–24. Springer-Verlag, 2008.

[120] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Intertational Journal of Applied Artificial Intelligence*, pages 145–189, 2000.

[121] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[122] J. Joseph and C. Fellenstein. *Grid Computing*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[123] A. Kapsalis, G. D. Smith, and V. J. Rayward-Smith. A unified paradigm for parallel genetic algorithms. *Lecture Notes in Computer Science - Selected Papers from AISB Workshop on Evolutionary Computing*, 865:131–194, 1994.

[124] S. A. Kauffman. *At Home in the Universe*. Oxford University Press, 1995.

[125] S. A. Kauffman. *Investigations*. Oxford University Press US, 2002. ISBN 0195121058, 9780195121056.

[126] P. J. Kearney and W. Merlat. Modelling market-based decentralised management systems. *BT Technology Journal*, 17(4):145–156, 1999.

[127] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, 1995.

[128] J. O. Kephart, J. E. Hanson, and A. R. Greenwald. Dynamic pricing by software agents. *Comput. Netw.*, 32(6):731–752, 2000.

[129] S. Kirn. *Gestaltung von Multiagenten-Systemen: Ein organisationszentrierter Ansatz*. Habilitation, Westfälische Wilhelms-Universität Münster, 1996.

[130] S. Kirn. Flexibility of multiagent systems. In S. Kirn, O. Herzog, P. Lockemann, and O. Spaniol, editors, *Multiagent Engineering*, International Handbooks on Information Systems, pages 53–70. Springer Berlin - Heidelberg, 2006.

[131] S. Kirn and T. Bieser. HoPIX: Simulationswerkzeug zur Analyse von Wertschöpfungsnetzwerken. In J. Banzhaf and H. Kuhnle, editors, *Entwicklungsperspektiven der Unternehmensführung und ihrer Berichterstattung*, pages 121–136. Gabler, Wiesbaden, 2006.

[132] S. Kirn, O. Herzog, P. Lockemann, and O. (Editors.). *Multiagent Engineering - Theory and Applications in Enterprises*. Springer Berlin Heidelberg, March 2006.

[133] H. Kornmayer, M. Stümpert, H. Gjermundrød, and P. Wolniewicz. g-eclipse — a contextualised framework for grid users, grid resource providers and grid application developers. In *ICCS '08: Proceedings of the 8th international conference on Computational Science, Part III*, pages 399–408, Berlin, Heidelberg, 2008. Springer-Verlag.

[134] V. K. Koumousis and C. P. Katsaras. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Trans. Evolutionary Computation*, 10(1):19–28, 2006.

[135] P. Koutsabasis, J. S. Darzentas, T. Spyrou, and J. Darzentas. Facilitating user-system interaction: The GAIA interaction agent. In *32nd Hawaii International Conference on System Sciences*, March 1999.

[136] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[137] K. W. Kratky. Die b̈eherrschbarkeitk̈omplexer systeme. In H. Carl Auer Verlag, editor, *Systemische Perspektiven. Interdisziplinäre Beiträge zu Theorie und Praxis*, pages 11–19. Karl W. Kratky, 1991.

[138] A. Kubik. Toward a formalization of emergence. *Artificial Life*, 9:41–65, 2003.

[139] F. Leal and J. Rodriguez. Message: Methodology for engineering systems of software agents. Technical report, Telecom Italia Lab and PT Inova
ccãl, 2001.

[140] H. L. Lee and S. Padmanabhan, V. und Whang. The bullwhip effect in supply chains. *Sloan Management Review*, 38(3):93–102, 1997.

[141] A. R. Leila Kallel, Bart Naudts. *Theoretical aspects of evolutionary computing*. Springer-Verlag, London, UK, 2001.

[142] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1997.

[143] S.-C. Lin, E. D. Goodman, and I. William F. Punch. Investigating parallel genetic algorithms on job shop scheduling problems. In *EP '97: Proceedings of the 6th International Conference on Evolutionary Programming VI*, pages 383–393, London, UK, 1997. Springer-Verlag.

[144] S.-C. Lin, W. Punch, and E. Goodman. Coarse-grain parallel genetic algorithms: categorization and newapproach. In *Sixth IEEE Symposium on Parallel and Distributed Processing (SPDP)*, pages 28–37, 1994.

[145] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. AgentLink, 2005.

[146] N. Luhmann and D. (Editor). *Einführung in die Systemtheorie*. Carl-Auer Verlag Heidelberg, 2004.

[147] S. Luke. Genetic programming produced competitive soccer softbot teams for robocup97. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 214–222, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.

[148] Z. S. Ma and A. W. Krings. Dynamic populations in genetic algorithms. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1807–1811, New York, NY, USA, 2008. ACM.

[149] R. H. MacArthur and E. O. Wilson. *The Theory of Island Biogeography*. Princeton University Press, 2001.

[150] M. A. Maluk Mohamed and D. Janakiram. Service composition based middleware architecture for mobile grid. In *Ubiquitous Computing, Proceedings of the 2nd International Workshop on Ubiquitous Computing, IWUC 2005, In conjunction with ICEIS 2005, Miami, USA, May 2005*, pages 39–46, 2005.

[151] P. Marrow, M. Koubarakis, R. van Lengen, F. Valverde-Albacete, E. Bonsma, J. Cid-Suerio, A. Figueiras-Vidal, A. Gallardo-Antolín, C. Hoile, T. Koutris, H. Molina-Bulla, A. Navia-Vázquez, P. Raftopoulou, N. Skarmeas, C. Tryfonopoulos, F. Wang, and C. Xiruhaki. Agents in decentralised information ecosystems: the diet approach. In *Proceedings of the Artificial Intelligence and Simulation Behaviour Convention*, pages 109–117, March 2001.

[152] D. Martin, A. Cheyer, and D. Moran. The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

[153] M. Matthies. Einführung in die Systemwissenschaft. Vorlesungsskript, http://www.usf.uos.de/archive/~vberding/syswi/skript10.pdf, 2002.

[154] K. Meffert. Jgap - java genetic algorithms and genetic programming package. http://jgap.sf.net/.

[155] N. Melab, S. Cahon, and E.-G. Talbi. Grid computing for parallel bioinspired algorithms. *Journal of Parallel and Distributed Computing*, 66(8):1052–1061, 2006.

[156] F. Menczer. *Life-like agents: internalizing local cues for reinforcement learning and evolution*. PhD thesis, University of California, San Diego, 1998. Chairperson-Richard K. Belew.

[157] H. W. Meuer. The top500 project: Looking back over 15 years of supercomputing experience. Internet, January 2008. University of Mannheim & Prometeus GmbH, Germany.

[158] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (2nd, extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1994.

[159] B. L. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. Technical report, Department of General Engineering, University of Illinois at Urbana-Champaign, 1995.

[160] B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, 1996.

[161] P. B. Mirchandani (Editor) and R. L. Francis (Editor). *Discrete Location Theory*. John Wiley and Sons, 1990.

[162] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. In *Physica D, 75*, pages 361–391, 1994.

[163] M. Mitchell and S. Forrest. Genetic algorithms and artificial life. *Artificial Life*, 1:267–289, 1994.

[164] S. Mostaghim, J. Branke, and H. Schmeck. Multi-objective particle swarm optimization on computer grids. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 869–875, New York, NY, USA, 2007. ACM.

[165] J. Mouritsen, A. Hansen, and C. Hansen. Inter-organizational controls and organizational competencies: episodes around target cost management/functional analysis and open book accounting. *Management Accounting Research*, 12(2):221–244, June 2001.

[166] T. Moyaux, B. Chaib-draa, and S. D'Amours. Multi-agent simulation of collaborative strategies in a supply chain. In *Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 52–59. ACM, 2004.

[167] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann, 1991.

[168] H. Mühlenbein. The breeder genetic algorithm - a provable optimal search algorithm and its application. *Colloquium on Applications of Genetic Algorithms, IEEE, London*, 67:5/1– 5/3, 1994.

[169] H. Mühlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm i. continuous parameter optimization. *Evolutionary Computation*, 1:25–49, 1993.

[170] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (bga). *Evolutionary Computation*, 1(4):335–360, 1993.

[171] J. P. Müller. *The Design of Intelligent Agents: A Layered Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[172] J. P. Müller and M. Pischel. The agent architecture InteRRaP: Concept and application. Technical Report 93-26, German Artificial Intelligence Research Center (DFKI), Saarbrücken, June 1993.

[173] C. Müller-Schloer. Organic computing - on the feasibility of controlled emergence. In *CODES+ISSS '04: Proceedings of the international conference on Hardware/Software Codesign and System Synthesis*, pages 2–5, Washington, DC, USA, 2004. IEEE Computer Society.

[174] C. Müller-Schloer, C. von der Malsburg, and R. P. Würtz. Organic computing. *Informatik Spektrum*, 27(4):332–336, 2004.

[175] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[176] M. Nowostawski and R. Poli. Parallel genetic algorithm taxonomy. In *Proceedings of the Third International Conference on Knowledge-based intelligent Information Engineering Systems KES'99*, pages 88–92. IEEE Computer Society, August 1999.

[177] H. S. Nwana and D. T. Ndumu. A perspective on software agents research. *The Knowledge Engineering Review*, 14(2):1–18, 1999.

[178] A. Oberschelp. *Rekursionstheorie*. BI-Wissenschaftsverlag, Mannheim, 1993. B035.

[179] R. Oestereich. *Potentiale und Risiken von Peer-to-peer-technologien*. GRIN Verlag, 2007.

[180] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Media, Inc., 1 edition, February 2001.

[181] J. Ottino. Complex systems. *American Institute of Chemical Engineers Journal*, 49(2):292–299, February 2003.

[182] S. Otto and S. Kirn. Adapting population size and strategies in multi agents systems using evolutionary computation. Technical report, Department of Information Systems II, Universität Hohenheim, , Internal research report, 2005.

[183] S. Otto and S. Kirn. Adaption in distributed systems: an evolutionary approach. In M. K. et al., editor, *Proceeding Genetic and Evolutionary Computation Conference (GECCO-06)*, volume 1, pages 199–206. ACM New York, NY, USA, 2006.

[184] S. Otto and S. Kirn. Evolutionary adaptation in complex systems using the example of a logistics problem. *International Transactions on Systems Science and Applications*, 2(2):157–166, 2006.

[185] S. Otto and G. Kókai. Decentralized evolutionary optimization approach to the p-median problem. In F. Rothlauf and A. Fink, editors, *in Proc EvoWorkshops 2008, Second European Workshop on Evolutionary Computation in Transportation and Logistics*, volume 4974, pages 659–668, Springer Berlin / Heidelberg, 2008. Springer Verlag.

[186] L. Padgham and M. Winikoff. Prometheus: a methodology for developing intelligent agents. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 37–38, New York, NY, USA, 2002. ACM.

[187] K. Park and W. Willinger. *The Internet as a large-scale complex system*. Oxford University Press US, 2005.

[188] J. L. Payne and M. J. Eppstein. Takeover times on scale-free topologies. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 308–315, New York, NY, USA, 2007. ACM.

[189] A. Petcu. Dynamic distributed optimization for planning and scheduling. *American Association for Artificial Intelligence (www.aaai.org)*, 2005.

[190] A. Petcu and B. Faltings. A-dpop: Approximations in distributed optimization. In *poster in Principles and Practice of Constraint Programming CP 2005*, pages 802–806, Sitges, Spain, October 2005.

[191] H. Pfohl. *Logistiksysteme*. Springer Verlag Berlin Heidelberg, 2000.

[192] A. Picot. Transaktionskostenansatz in der Organisationstheorie: Stand der Diskussion und Aussagewert. *Die Betriebswirtschaft: DBW*, 42(2):267–283, 1982.

[193] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: Implementing a BDI-infrastructure for jade agents. *EXP - in search of innovation (Special Issue on JADE)*, 3(3):76–85, 9 2003.

[194] T. Posch, K. Birken, and M. Gerdom. *Basiswissen Softwarearchitektur*. dpunkt-Verlag, Heidelberg, 1nd ed. edition, Oktober 2004.

[195] M. Preusser, C. Almeder, R. F. Hartl, and M. Klug. Lp modelling and simulation of supply chain. In H.-O. Günther, D. C. Mattfeld, and L. Suhl, editors, *Supply Chain Management und Logistik: Optimierung, Simulation, Decision Support*, pages 95–113. Physica-Verlag, 2005.

[196] M. J. Quinn. *Parallel computing (2nd ed.): theory and practice*. McGraw-Hill, Inc., New York, NY, USA, 1994.

[197] I. Rechenberg. Cybernetic solution path of an experimental problem. Technical report, Royal Air Force Establishment, 1965.

[198] J. Reese. Methods for solving the p-median problem: An annotated bibliography. Technical report, Department of Mathematics, Trinity University, 2005.

[199] M. Resnick. *Turtles, termites, and traffic jams: explorations in massively parallel microworlds*. MIT Press, Cambridge, MA, USA, 1994.

[200] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for organic computing. In C. Hochberger and R. Liskowsky, editors, *INFORMATIK 2006 – Informatik für Menschen*, volume P-93 of *GI-Edition – Lecture Notes in Informatics*, pages 112–119, Bonn, Germany, Sept. 2006. Köllen Verlag.

[201] F. F. Rivera, M. Bubak, A. G. Tato, and R. Doallo, editors. *Grid Computing, First European Across Grids Conference, Santiago de Compostela, Spain, February 13-14, 2003, Revised Papers*, volume 2970 of *Lecture Notes in Computer Science*. Springer, 2004.

[202] R. Rojas. *Neural Networks - A Systematic Introduction*. Springer-Verlag, Berlin, New-York, 1996.

[203] F. Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer, 2006.

[204] G. Rudolph. On takeover times in spatially structured populations: array and ring. In *Proceedings of the Second Asia-Pacific Conference on Genetic Algorithms and Applications*, pages 144–151. Global-Link Publishing Company, 2000.

[205] S. Russell and P. Norvig. *Artificial Intelligene: A Modern Appoach*. Prentice Hall, 1995.

[206] T. W. Sandholm. Distributed rational decision making. In *Multiagent systems: a modern approach to distributed artificial intelligence*, pages 201–258. MIT Press, Cambridge, MA, USA, 1999.

[207] J. Sarma and K. D. Jong. An analysis of the effects of neighborhood size and shape on local selection algorithms. In *Lecture Notes In Computer Science, Vol. 1141, Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 236–244. Springer-Verlag, 1996.

[208] K. Sastry and D. E. Goldberg. Modeling tournament selection with replacement using apparent added noise. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 781, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[209] J. Schaffer and L. Eshelman. On Crossover as an Evolutionary Viable Strategy. In R. Belew and L. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 61–68. Morgan Kaufmann, 1991.

[210] H. Schmeck. Organic computing. *Künstliche Intelligenz*, 05(3):68–69, JUL 2005.

[211] H. Schmeck. Organic computing- a new vision for distributed embedded systems. In *Proceedings Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005), 18-20 May 2005, Seattle, WA, USA*, pages 201–203. IEEE, IEEE Computer Society 2005, MAY 2005.

[212] R. H. Schmidt and E. Freese (Editor). *Handwörterbuch der Organisation. Enzyklopädie der Betriebswirtschaftslehre, Band 2*, pages 1854–1865. Schäffer-Poeschel, Stuttgart, 3 edition, 1992.

[213] U. Schöning. *Theoretische Informatik - kurzgefasst*. Spektrum Akademischer Verlag GmbH Heidelberg Berlin, 3rd edition, 1997.

[214] U. Schreiber. *Handlexikon Wirtschaft*. Wilhelm Heyne Verlag, Gmbh & Co. KG, München, 4 edition, 1990.

[215] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[216] H.-P. Schwefel. Evolutionary learning optimum-seeking on parallel computer architectures. In A. Sydow, S. G. Tzafestas, and R. Vichnevetsky, editors, *Systems Analysis and Simulation*, volume 1, pages 217–225. Akademie-Verlag, Berlin, 1988.

[217] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[218] C. R. Shalizi. Methods and techniques of complex systems science: An overview, 2006.

[219] H. T. Siegelmann and E. D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[220] R. E. Smith, C. Bonacina, P. Kearney, and T. Eymann. Integrating economics and genetics models in information ecosystems. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 959–966, La Jolla Marriott Hotel La Jolla, California, USA, 6-9 2000. IEEE Press.

[221] R. E. Smith and N. Taylor. A framework for evolutionary computation in agent-based systems. In C. Looney and J. Castaing, editors, *Proceedings of the 1998 International Conference on Intelligent Systems*, pages 221–224. ISCA Press, 1998.

[222] R. G. Smith. The contract net protocol: high-level communication and control in a distributed problem solver. In *Distributed Artificial Intelligence*, pages 357–366. Morgan Kaufmann Publishers Inc., 1988.

[223] W. M. Spears, K. A. D. Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In P. B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667, pages 442–459, Vienna, Austria, 1993. Springer Verlag.

[224] J. Sprave. A unified model of non-panmictic population structures in evolutionary algorithms. Technical Report CI-55/99, University of Dortmund - Dept. of Computer Science/XI, 44221 Dortmund - Germany, January 1999. ISSN 1433-3325.

[225] J. D. Sterman. Modeling managerial behavior: misperceptions of feedback in a dynamic decision making experiment. *Management Science*, 35(3):321–339, 1989.

[226] F. Streichert and H. Ulmer. Javaeva - a java based framework for evolutionary algorithms. Technical report, Centre for Bioinformatics Tübingen (ZBIT), 2005.

[227] T. Stützle and M. Dorigo. ACO algorithms for the traveling salesman problem. In K. Miettinen, P. Neittaanmäki, M. M. Mäkelä, and J. Périaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, chapter 9, pages 163–183. John Wiley & Sons, Chichester, UK, 1999.

[228] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT Press, Cambridge, Massachusetts, 1998.

[229] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, December 1996.

[230] Szilvia Zvada, Gabriella Kókai, Hans Holm Frühauf, and Robert Vanyi. evolFIR: Evolving redundancy-free FIR structures. In Tughrul Arslan, editor, *Proceedings NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pages 439–446, 2007.

[231] D. Talbot. Ein exponentielles Wachstum der Komplexität. http://www.heise.de/tr/artikel/103166, 2008.

[232] A. S. Tanenbaum and M. van Steen. *Verteilte Systeme. Grundlagen und Paradigmen*. Pearson Studium, 1 edition, April 2003.

[233] R. Tanese. Parallel genetic algorithms for a hypercube. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 177–183, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[234] L. Tesfatsion. Agent-based computational economics: A constructive approach to economic theory, July 2005.

[235] D. Thierens and D. E. Goldberg. Mixing in genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 38–47, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[236] S. B. Thrun. Efficient exploration in reinforcement learning. Technical report, Computer Science Department, Pittsburgh, PA, USA, 1992.

[237] I. J. Timm. *Dynamisches Konfliktmanagement als Verhaltenssteuerung Intelligenter Agenten*. PhD thesis, Fachbereich Mathematik und Informatik, Universität Bremen: Bremen, 2003.

[238] I. J. Timm, P. Knirsch, H.-J. Müller, M. Petsch, N. Abchiche, P. Davidsson, Y. Demazeau, F. J. Garijo, O. Herzog, S. Kirn, C. Petrie, and C. Tessier, editors. *Agent Technologies and Their Application Scenarios in Logistics*, 2000.

[239] I. J. Timm and P.-O. Woelk. Ontology-based capability management for distributed problem solving in the manufacturing domain. In M. S. et al., editor, *Multiagent System Technologies - Proceedings of the First German Conference, MATES 2003, Erfurt, Germany*, Lecture Notes in Artificial Intelligence, pages 168–179. Springer, September 2003.

[240] A. Tiwari, G. Goteng, and R. Roy. *Advances in Evolutionary Computing for System Design*, volume 66 of *Studies in Computational Intelligence*, chapter Evolutionary Computing within Grid Environment, pages 229–248. Springer Berlin / Heidelberg, 2007.

[241] H. R. Varian. Economic mechanism design for computerized agents. In *WOEC'95: Proceedings of the 1st conference on USENIX Workshop on Electronic Commerce*, pages 2–2, Berkeley, CA, USA, 1995. USENIX Association.

[242] VDE / ITG / GI-Arbeitsgruppe Organic Computing. Organic Computing - Computer- und Systemarchitektur im Jahr 2010. http://www.gi-ev.de/fileadmin/redaktion/Presse/VDE-ITG-GI-Positionspapier_20Organic_2 2003.

[243] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961.

[244] D. Vrajitoru. Parallel genetic algorithms based on coevolution. In E. D. Goodman, editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 45–457, San Francisco, California, USA, 9-11 2001.

[245] M. Waldrop. *Complexity: The Emerging Science at the Edge of Order and Chaos*. A Touchstone BookSimon & Schuster, 1992.

[246] M. Weiser. The computer for the 21st century. In *Human-computer interaction: toward the year 2000*, pages 933–940. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[247] G. Weiss. *Mulitagent Systems - A Modern Appoach to Distributed Artificial Intelligence*. The MIT Press Cambridge, Massachusetts, London England, 2001.

[248] J. B. Weissman, S. Kim, and D. England. Supporting the dynamic grid service lifecycle. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 808–815, Washington, DC, USA, 2005. IEEE Computer Society.

[249] R. Whitaker. A fast algorithm for the greedy interchange for large-scale clustering and median location problems. *INFOR 21*, pages 95–108, 1983.

[250] D. C. Whybark and B. M. Khumawala. *A Survey of Facility Location Methods*. Veröffentlicht von Herman C. Krannert Graduate School of Industrial Administration, Purdue University, 1972.

[251] U. Wilensky. Netlogo. http://ccl.northwestern.edu/netlogo, 1999. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL.

[252] P.-O. Woelk, H. Rudzio, R. Zimmermann, and J. Nimis. *Agent.Enterprise in a Nutshell*, pages 73–90. International Handbooks on Information Systems. Springer Berlin Heidelberg, March 2006.

[253] S. Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96:15–57, 1984.

[254] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, West Sussex, UK, March 2002.

[255] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[256] M. Wooldridge and N. R. Jennings. Software engineering with agents: Pitfalls and pratfalls. *IEEE Internet Computing*, 3(3):20–27, 1999.

[257] M. Wooldridge and A. Lomuscio. Multi-agent vsk logic. In *JELIA '00: Proceedings of the European Workshop on Logics in Artificial Intelligence*, pages 300–312, London, UK, 2000. Springer-Verlag.

[258] M. J. Wooldridge and N. R. Jennings. Agent Theories, Architectures, and Languages: A Survey. In M. J. Wooldridge and N. R. Jennings, editors, *Workshop on Agent Theories, Architectures & Languages (ECAI'94)*, volume 890 of *Lecture Notes in Artificial Intelligence*, pages 1–22, Amsterdam, The Netherlands, Jan. 1995. Springer-Verlag.

[259] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.

[260] K. Zhang and W. Pan. The two facets of the exploration-exploitation dilemma. In *IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 371–380, Washington, DC, USA, 2006. IEEE Computer Society.

[261] D. Ziegenbein. *A Compositional Approach to Embedded System Design.* Phd thesis, TU Braunschweig, Fakultät für Maschinenbau und Elektrotechnik der Technischen Universität Carolo-Wilhelmina zu Braunschweig, 2002.

Stichwortverzeichnis