technische universität
dortmund

Anwendung evolutionärer Algorithmen auf ein semantisches Web

Kay Thielmann

Algorithm Engineering Report
TR08-2-009
Dez. 2008
ISSN 1864-4503

Fakultät für Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

technische universität
dortmund

Diplomarbeit

Anwendung evolutionärer Algorithmen auf ein semantisches Web

Kay Thielmann

24. Dezember 2008

Betreuer:
Prof. Dr. Günter Rudolph
Dr. Christian Schumer, Materna GmbH Dortmund

Fakultät für Informatik
Algorithm Engineering (Ls11)
Technische Universität Dortmund
http://ls11-www.cs.uni-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Informationen für Web 2.0 Applikationen können über die vielfältigsten Schnittstellen erfasst werden. Dabei sind nicht nur direkte Schnittstellen zu einem Content-Management-System denkbar, sondern auch benutzereigene Blogs, Wikis, Foren und ähnliches. Diese Vielfältigkeit ermöglicht es dem einzelnen Benutzer, die für ihn optimale Form der Verwaltung seiner Daten zu finden. Allerdings besteht die Gefahr, dass das Gesamtsystem dadurch unübersichtlich wird. In einem wachsenden Netz mit mehr und mehr Datenquellen wird das bereits erfasste Wissen nur noch aufwändig wieder gefunden oder gerät sogar ganz in Vergessenheit. Die Pflege alter Daten und deren Einbindung in neue Kontexte sowie die Suche nach bestimmten Artikeln verursacht einen beträchtlichen Zeitaufwand, der zusätzlich noch durch die hohe Dynamik der Systeme erweitert wird. Durch die Anwendung evolutionärer Algorithmen als Basis für eine selbstlernende Navigation soll dies abgefangen werden. Hierbei liefert bereits die Art und Weise, wie ein Anwender durch das Web navigiert, wertvolle Informationen über dessen Interessen und sein Wissen. Daraus „lernt“ das Web und kann weitere passende Objekte gezielt anbieten sowie das Wissen dieses Anwenders an anderen geeigneten Stellen weiter verbreiten. Wissen wird so auf kurzen Wegen im Web erreichbar und die Recherche nach weiterführenden Informationen wird unterstützt.

Alle in einem solchen System gesammelten Informationen stehen in einer gewissen Beziehung zueinander. Nicht nur die reine durch Links aufgebaute Vernetzung der Seiten zu einem Graph gibt Informationen über deren Beziehungen. Auch Zuordnungen enthalten wichtige Daten, wie beispielsweise vom Nutzer erfasste Themen, Kategorisierungen oder Tags. Viele dieser Beziehungen erfasst das Web einmalig beim Anlegen eines Artikels. Der Ersteller oder spätere Nutzer ist nicht oder nur selten verpflichtet, diese Einordnungen weiter zu pflegen. So veralten die Daten und neu entstehende Kontexte, in denen die bereits

erfassten Informationen wieder interessant sein könnten, sind im ungünstigsten Fall bereits gar nicht mehr damit verbunden.

Wünschenswert wäre es, die Interessen und das Wissen eines Benutzers aus seinem täglichen Nutzungsverhalten abzuleiten und das so gesammelte Wissen zu nutzen. Einerseits könnte eine zusätzliche Navigationsansicht dem Benutzer selbstständig neue Objekte vorschlagen, die mit seiner aktuellen Arbeit verbunden sind. Andererseits könnten auch Dritte auf diese Informationen leichter zugreifen. Im Wesentlichen lässt sich also ein evolutionärer Algorithmus auf folgende Weise anwenden: Aus dem Verhalten eines Benutzers und den Pfaden, die er im Web gegangen ist, lassen sich dessen Interessen und Wissen ableiten. Daraus generiert das Web neue Pfade zu weiterführenden Objekten, die den Anwender bei seiner Arbeit unterstützen.

1.1.1 Evolutionärer Algorithmus

Der Aufbau eines evolutionären Algorithmus, der solch eine Aufgabe übernehmen könnte, besteht im Wesentlichen aus den Stufen Rekombination, Mutation und Selektion, die von einer Menge von Individuen in mehreren Iterationen durchlaufen werden (siehe Abbildung 1.1). Ein Individuum repräsentiert dabei eine mögliche Lösung des bearbeiteten Problems. Dabei müssen die Besonderheiten eines semantischen Web im Design des evolutionären Algorithums Berücksichtigung finden. Abgesehen von der Möglichkeit mit Hilfe evolutionärer Algorithmen gute Lösungen für Probleme von hoher Komplexität und vielen Nebenbedingungen zu finden, liegt eine weitere Stärke in ihrer schnellen Reaktions- und Anpassungsfähigkeit an dynamische Problemstellungen. Diese Fähigkeit in Webapplikationen oder anderen Anwendungen zu nutzen ist bis jetzt eine Seltenheit.

Die Rekombination erschafft aus einer Menge von Individuen ein oder mehrere neue Individuen, die die Eigenschaften der Elterngeneration in abgewandelter Weise widerspiegeln. Die Mutation selbst ist eine randomisierte Abwandlung der Individuen, die die Diversität in einer Generation der Individuen noch erhöhen kann. In der Regel wird die Stärke der Mutation mit Hilfe eines Faktors begrenzt. Dieser Faktor kann wahlweise vom Algorithmus selbst optimiert werden (Selbstadaption) oder er wird fest vorgegeben. Die Selektion legt schließlich die Individuen der entstehenden Kindergeneration fest, die in die folgende Elterngeneration übernommen werden. Wesentliches Entscheidungskriterium hierfür ist der Fitnesswert, der dem einzelnen Individuum zugeordnet ist. Der Fitnesswert wird lediglich als Kriterium für die Wahrscheinlichkeit genutzt, mit der ein Individuum in die neue Elterngeneration aufrücken darf. Dies verhindert, dass Individuen, die zwar in der aktuellen Phase einen schlechten Fitnesswert haben, aber sich vielleicht auf einem „guten Weg“ befinden, während der Selektion grundsätzlich aus der Generation entfernt werden. Die Auswahl selbst findet also auch randomisiert statt und ermöglicht es so auch Indivi-

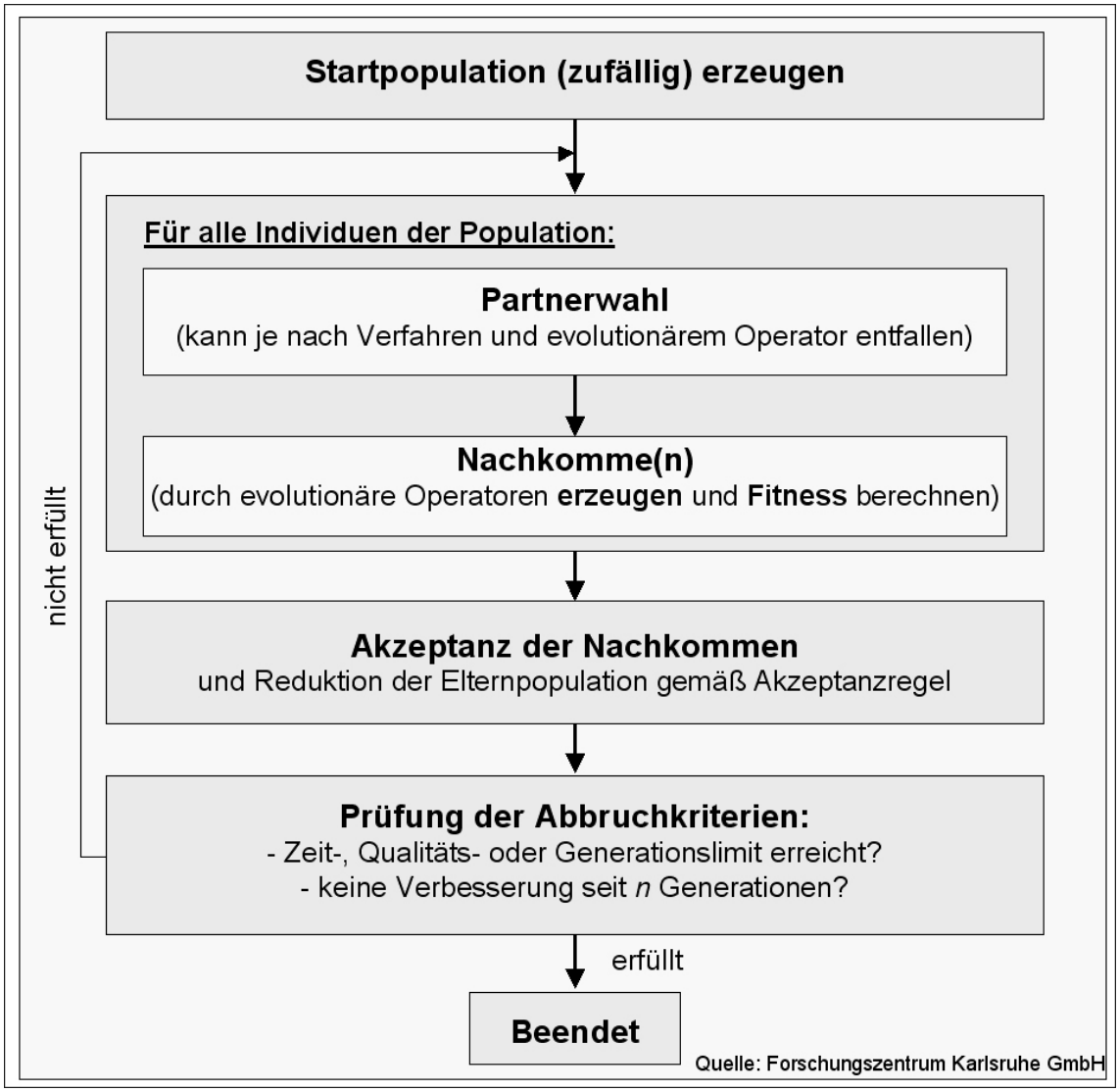

Abbildung 1.1: Schematischer Aufbau eines einfachen evolutionären Algorithmus entnommen aus http://www.iai.fzk.de/www-extern/index.php?id=237
Datum: 14.10.2008

duen mit schlechten Fitnesswerten, mit einer gewissen Wahrscheinlichkeit übernommen zu werden.

Anwendung eines evolutionären Algorithmus

Die Darstellung der Individuen ist der zentraler Punkt, um einem evolutionären Algorithmus die Arbeit auf einem semantischen Web zu ermöglichen. Das einzelne Individuum muss alle Informationen enthalten, die einerseits für den Algorithmus notwendig sind und andererseits eine gute Darstellung für den Benutzer erlauben. Die durch ein semantisches Web aufgebaute Datenstruktur bildet einen gerichteten Graphen mit Kanten, denen ein

bestimmter Typ zugeordnet ist (siehe Abbildung 2.1). Es entsteht also ein Netzwerk aus Zeigern zwischen den Objekten des Webs. Jeder Knoten des Graphen stellt ein aufrufbares Objekt dar. Jede gerichtete Kante entspricht einer Assoziation des semantischen Web. Objekte in diesem Graph sind also durch einen Weg von Assoziationen erreichbar. Ein Individuum ließe sich als Pfad in Form einer Abfolge von Assoziationen darstellen.

Als Ausgangspunkt für die Interpretation eines Pfades auf den Graphen der Objekte bieten sich die Interpretation der Pfade auf Basis der Typen der einzelnen Instanzen des Pfads an. In einem semantischen Web können allen Instanzen Typen zugeordnet werden. Als Ausgangspunkte einer ganzen Reihe von Pfaden wären also alle Instanzen eines bestimmten Typs denkbar. Ein Individuum besteht also zusammengefasst aus einem Pfad, wobei zu jeder Instanz des Pfads eine Liste von Typen gepflegt wird.

Aufbauend auf dieser Art der Darstellung eines Individuums kann der wesentliche Teil der Rekombination an bereits erprobte Rekombinationsverfahren anderer kombinatorischer Probleme angelehnt werden, wie etwa dem aus der theoretischen Informatik bekannten Traveling-Salesman-Problem [5]. Dabei werden beispielsweise Ausschnitte der Pfade von zwei oder mehreren Individuen miteinander zufällig verknüpft, um so ein neues Individuum zu erhalten. Dabei ist es explizit nicht die Aufgabe des an dieser Stelle angewendeten Verfahrens, darauf zu achten, dass die entstehenden Individuen Pfade darstellen, die grundsätzlich auf den Graphen anwendbar sind. Diese Nebenbedingungen können im Rahmen eines evolutionären Algorithmus wesentlich besser durch entsprechende Bestrafungsfunktionen abgedeckt werden.

Auch aus der Mutation, deren Stärke bei solchen Anwendungsfällen aus Erfahrung mit anderen kombinatorischen Problemen gering gehalten wird, können Individuen entstehen, deren Pfad nicht im Graph existiert.

Die Fitnesswerte

Die für die Selektion der Individuen, die in die neue Elterngeneration aufrücken dürfen, benötigten Fitnesswerte beeinflussen die Leistungsfähigkeit eines evolutionären Verfahrens. Zum einen lässt sich das Interesse aller Benutzer an einem Individuum bewerten, zum anderen ist das Interesse jedes einzelnen Benutzers eine eigene Größe. Ein Ansatz, der mehrere Kriterien gleichzeitig berücksichtigt, lässt sich hier jedoch nicht anwenden, denn die Menge der Benutzer ist keine feste Größe und wäre für die meisten Verfahren wesentlich zu hoch. Die Anwendung auf ein semantisches Web könnte folgendermaßen aussehen: Für jeden Benutzer existiert ein kleiner Pool an Individuen, der nach seinem benutzerspezifischen Verhalten optimiert wird.

Um die Fitnesswerte der Individuen zu ermitteln, bieten sich die Eingaben des angemeldeten Benutzers direkt an. Diese lassen sich über einen Servlet-Filter des Web-Servers erfassen und daraus Pfade generieren. Diese Informationen ermöglichen es, die Pfade eines

Benutzers zu verfolgen und seine benutzerspezifischen Individuen danach zu bewerten, wie häufig sie vorkommen. Eine Auswahl der besten Individuen aus dieser Bewertung kann nun dem Benutzer in seiner Oberfläche angeboten werden. Die entsprechende Darstellung als Link, zum Beispiel in einem Servlet oder Portlet, kann dann mit einer Funktion verbunden sein, die bei einem Klick darauf den Fitnesswert des entsprechenden Individuums weiter erhöht.

Gerade der Start einer solchen Umgebung gestaltet sich jedoch schwierig. Damit der Algorithmus sich verbessern kann, muss die Benutzerakzeptanz vorhanden sein. Wenn aber, wie hier angedacht, durch einen Servlet-Filter alle Eingaben in den Algorithmus eingehen, sollte sich das Bild der angebotenen Links von einem randomisiert erscheinenden zu einem zunehmend benutzerspezifischen Bild verbessern.

1.1.2 Ziele

Eine auf einem semantischen Web aufbauende Web-Applikation, die mit Hilfe eines lernfähigen Systems auf evolutionären Algorithmen basiert, soll eine einfache Benutzerführung gewährleisten. Dies erzeugt beim Benutzer keinen weiteren Aufwand. Ohne Weiteres zu tun, werden die Informationen des Einzelnen gezielt in die Gruppe aller Benutzer befördert. Zusätzlich unterstützt das System den einzelnen Benutzer bei der Suche nach weiteren geeigneten Themen. Nach einiger Zeit des „Lernens“ verbessert sich so die Qualität der angebotenen Hilfen eigenständig. Sämtliche Themenwechsel einzelner Anwender können zukünftig zunehmend schneller in der Benutzerführung berücksichtigt werden, ohne dass etwas manuell geändert werden muss. Die Navigation im semantischen Web erhält hierdurch seinen dynamischen Bestandteil, der den Eigenschaften eines semantischen Webs und seiner Nutzungsweise entspricht. Eindeutig vergebene Assoziationen sind für das Funktionieren eines evolutionären Algorithmus nicht zwingend notwendig und müssen somit nicht übermäßig reglementiert oder gepflegt werden. Insgesamt entsteht ein System, das den Anwender bei seiner täglichen Arbeit unterstützt.

Um ein solches System umzusetzen soll eine Reihe von Anpassungen und Erweiterungen von bekannten evolutionären Algorithmen für diese Problemstellung erfasst und gestestet werden. Für die einzelnen Komponenten eines evolutionären Algorithmus werden dessen Möglichkeiten erfasst für diese Problemstellung angepasst zu werden. Die Auswirkungen der Kombination und Konfiguration der einzelnen Komponenten auf die Leistungsfähigkeit bzw. den erwarteten Nutzen soll erfasst werden, um schließlich sagen zu können welche Anpassungen nötig sind um ein solches Verfahren in dieser Umgebung sinnvoll nutzen zu können.

1.2 Aufbau der Arbeit

Der Erstellung dieser Arbeit ging eine Recherche- und Implementierungsphase voraus. Ausgehend von den in der Einleitung dargestellten Ideen und Zielen wurde ein System gesucht, das die notwendigen Voraussetzungen für eine Umsetzung bietet. Dieses musste sowohl eine gute Erweiterbarkeit als auch ein breites Spektrum bereits implementierter Funktionen bieten. Durch die Integration des Spring-Frameworks in die Serversoftware IkeWiki konnte in der Implementierungsphase ein Modulbaukasten entwickelt werden, durch den die einzelnen Komponenten einer Evolutionsstrategie konfigurativ zu einem funktionierenden Gesamtsystem zusammengeführt werden können. Dafür wurden in einem ersten Schritt die Basismodule für die Initialisierung, Rekombination, Mutation, Selektion und Fitnessberechnung erstellt. Diese orientieren sich in wesentlichen Teilen an der bekannten Evolutionstrategie. In diesem Schritt wurden durch die Umsetzung der zentralen Ideen für die Gestaltung der Individuen sowie der Fitnessberechnung bereits die ersten Anpassungen an das Umfeld der Evolutionsstrategie vorgenommen.

Auf Basis dieser ersten grundlegenden Implementierung wurden Ideen für weitere Module oder Varianten von diesen gesammelt. Durch kurze Funktionstests entstanden Ideen für die Anpassung des Algorithmus an die Anforderungen, die durch den Benutzer und das umgebende System entstehen. Diese Varianten wurden als alternative Module implementiert und dokumentiert.

Die folgenden Kapitel sollen nun den Rahmen dieser Arbeit und deren Ergebnisse wiedergeben. Dazu gibt Kapitel 2 einen grundlegenden Überblick über die Technologien auf denen das genutzte System aufbaut. Hier wird eine Einführung in die Begriffe des semantischen Web sowie die technischen Grundlagen des Web 2.0 gegeben. Diese stellen die Basis für die Entstehung und Entwicklung von Systemen wie dem für diese Arbeit genutzten IkeWiki dar.

Kapitel 3 verschafft dann einen näheren Einblick in die Details der genutzten Software. Der Aufbau und das Zusammenspiel der einzelnen Komponenten wird näher erläutert und deren Konfigurationsmöglichkeiten näher beschrieben. Dabei wird im speziellen die Konfiguration und Erweiterbarkeit von IkeWiki, sowie die Architektur der semantischen Navigation erläutert.

Das folgende Kapitel 4 ist dann speziell der Darstellung der Algorithmen sowie den Feinheiten der einzelnen Module gewidmet. Dazu werden in einem ersten Schritt die aufeinander aufbauenden Implementierungen der $(\mu/\rho + \lambda)$-Evolutionsstrategie und des Cluster Based Niching EA näher erläutert, um dann auf die gemeinsam genutzten Module für die Initialisierung, Rekombination, Mutation und Fitnessberechnung sowie einige Bestrafungsfunktionen im Detail einzugehen.

Die grafische Darstellung des IkeWiki sowie der semantischen Navigation sind Inhalt des Kapitel 5. Hier werden die Eingabemasken von IkeWiki vorgestellt, die über die nor-

malen Funktionen eines Wikis hinaus gehen. Dabei stehen vor allem die Entwicklung von Ontologien im Wiki sowie die Eingabe der Semantik der Inhalte des Wikis im Mittelpunkt. Weiterhin wird die Oberfläche der semantischen Navigation in ihrer aktuellen Ausprägung beschrieben und Hinweise für eine mögliche Gestaltung bei der Nutzung durch echte Benutzer gegeben. Durch die grafische Repräsentation der semantischen Navigation ergeben sich weitere Möglichkeiten für den Benutzer Feedback an den Algorithmus weiterzuleiten. Bereits genutzte sowie weitere angedachte Feedbackmöglichkeiten werden hier beschrieben.

Kapitel 6 enthält eine detailierte Beschreibung der Testumgebung, die genutzt wurde um die implementierten Algorithmen mit unterschiedlichen Modulen zu testen. Dem Aufbau der Testumgebung, den Tests und ihrer Auswertung selber, sowie der für die Tests genutzen Software werden hier eigene Unterkapitel gewidmet. Hier wird der erwartete Nutzen den tatsächlich erreichten Werten der einzelnen Module gegenüber gestellt und so deren Eingung für den Einsatz in einem semantischen Web bewertet.

Abschließend soll in Kapitel 7 ein Überblick über die Ergebnisse dieser Arbeit gegeben werden. Die auftretenden Probleme bei der Anwendung evolutionärer Algorithmen auf ein semantisches Web sowie mögliche Lösungen und deren Leistungsfähigkeit werden zusammengefasst. Offene und weiterführende Fragestellungen zu dem hier behandelten Thema bilden den Abschluss dieser Arbeit.

Kapitel 2

Technologien des zugrundeliegenden Webs

Dieses Kapitel behandelt die grundlegenden Technologien, auf denen diese Arbeit aufbaut. Es soll ein Überblick über die Funktionen und den Nutzen für den Anwender gegeben werden.

2.1 Das semantische Web

Zusätzlich zu dem reinen Netz aus Links zwischen verschieden Seiten im Web liefert ein semantisches Web [9] weitere Informationen zu den Beziehungen zwischen den Objekten. Die Objekte, die mit zusätzlichen Informationen ausgestattet werden sollen, werden im Rahmen eines semantischen Webs als Instanzen bezeichnet. Jede beliebige Instanz kann zu jeder anderen getypte Assoziationen besitzen, die sich kontinuierlich verändern können oder ergänzt werden. Zwei Instanzen können so in maschinenlesbarer Weise ihre Beziehung zueinander beschreiben. Sowohl die Arten der Assoziationen als auch deren Orientierung und Menge können sich in einer stark genutzten Anwendung stetig wandeln.

2.1.1 Typen

Jeder Instanz im Web lassen sich eine Reihe von Typen zuorden, die diese Instanz beschreiben. Diese Typen können von einer Webanwendung teilweise schon automatisch vergeben werden. Der wesentlich interessantere Teil ergibt sich aber durch die Zuordnungen, die der Benutzer selber tätigt. Diese geben ein maschinenlesbares Abbild davon in welchen Context der Benutzer eine Instanz im Web einordnet. Typen selber können in einer Vererbungshierarchie zueinander stehen, die der Klassenhierarchie in objektorientierten Programmiersprachen ähnelt. So kann beispielsweise ein Typ "Vorgesetzter" dafür sorgen, dass das Objekt mit diesem Typ immer auch den Typ "Person" durch die Vererbung zugeordnet bekommt.

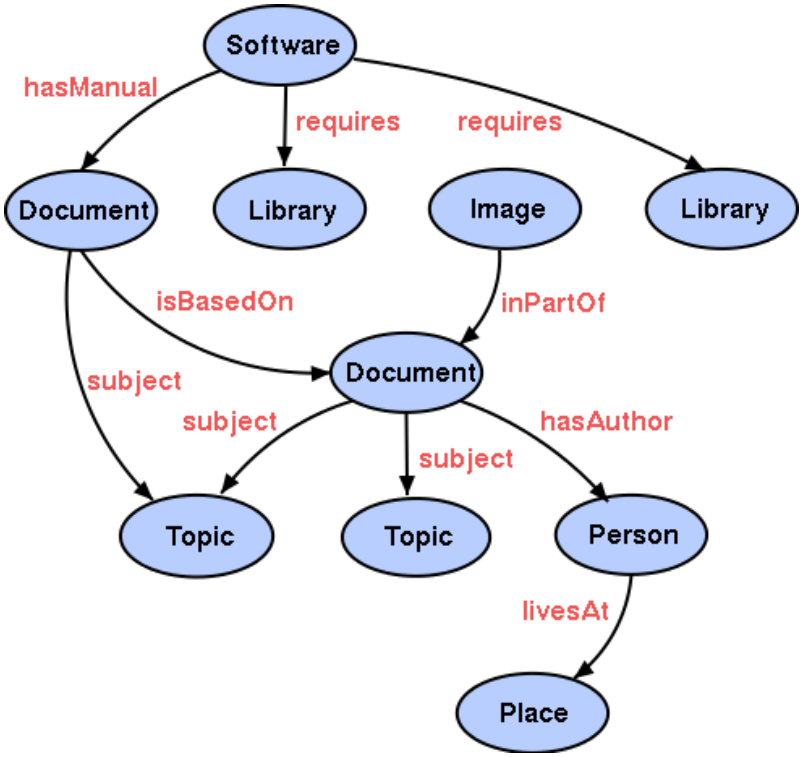

Abbildung 2.1: Verknüpfungen in einem semantischen Web. Entnommen aus http://www.w3.org/2004/Talks/0120-semweb-umich/semanticweb.png

Ein Basistyp dient als Wurzel dieser Vererbungshierarchie und ist somit automatisch jedem Objekt zugeordnet.

2.1.2 Semantische Verknüpfungen

Semantische Beziehungen zwischen den Instanzen lassen sich als ein Tripel aus einem Subjekt, einem Prädikat und einem Objekt darstellen. Sowohl Subjekt als auch Prädikat und Objekt sind selber auch als Instanzen im semantischen Web vorhanden. Sie können also selbst wieder einem oder mehreren Typen zugeordnet sein und in Beziehung zu weiteren Objekten stehen. Die Prädikate, die zwischen zwei Objekten genutzt werden können, sind abhängig von den Typen, die diesen zugeordnet werden. Jedes Objekt im Web, dass als Prädikat zwischen zwei Objekten genutzt werden soll, zeichnet sich durch den Typ "owl:ObjectProperty" aus. Außerdem enthält es über die Attribute "Domain" und "Range" Angaben darüber Instanzen welchen Typs als Ausgangs- und Zielpunkte dieses Prädikats zugelassen sind. Als weiteres Attribut des Prädikats kann angegeben werden welches andere Prädikat das inverse Element zu diesem darstellt.

2.1.3 Ontologien

Eine Ontologie im Sinne des semantischen Webs stellt ein Abbild aller möglichen Typen, deren Hierarchie, sowie der Beziehungen zwischen diesen dar. In Abhängigkeit von der Größe des Themengebiets, das von einer Ontologie beschrieben werden soll, kann diese sehr allgemein gehalten werden, oder auch Details näher beschreiben. So gibt es im Internet bereits vorgefertigte Ontologien für sehr spezielle Themengebiete wie z.B um die EXIF-Informationen eines Bildes (http://www.w3.org/2003/12/exif/ns/) zu verwalten oder um Dokumente um Informationen über Signaturen (http://xmlns.com/wot/0.1/) zu ergänzen. Andere Ontologien unterstützen die Bereitstellung wesentlich allgemeinerer Informationen. So steht mit Dublin Core (http://purl.org/dc/elements/1.1/) eine Ontologie zur Verfügung, die allgemein Metainformationen zu Dokumenten und anderen Objekten im Internet enthält.

Solche Ontologien lassen sich in Webanwendungen nutzen. Eine grundsätzliche Frage dabei ist immer, ob es einem Benutzer möglich sein soll diese Ontologien an seine eigenen Bedürfnisse anzupassen und sie zu ergänzen, oder ob eine starr vorgegeben Ontologie einen höheren Nutzen durch eine gesicherte Ordnung verspricht. Sicherlich bieten die bereits erwähnten vorhandenen Ontologien schon ein breites Spektrum an Möglichkeiten Beziehungen von Objekten einer Webanwendung zu erfassen. Durch die Kombination mehrerer solcher Ontologien lässt sich aber auch für spezielle Anwendungsfälle z.B. in einem Firmenintranet eine gute Basis schaffen. Allerdings ergeben sich je nach Anwendungsfall einer Webanwendung immer auch Einzelfälle, die in die semantische Struktur aufgenommen werden sollten. Diese Sonderfälle können teilweise durch eingehende Analysen im Vorfeld der Entwicklung einer Webanwendung ermittelt werden. Ob auf diese Weise wirklich alle gewünschten semantischen Beziehungen ermittelt werden, die zukünftigt benötigt werden könnten, ist nie sicher zu sagen. So entsteht die Anforderung die der Webanwendung zugrunde liegenden Ontologien pflegen und ergänzen zu können, um später auftretende Anforderungen mit zu berücksichtigen und dem Anwender ein Höchstmaß an Autonomie zu bieten.

Eine hoch dynamische Webanwendung, in der viele Benutzer auf die unterschiedlichsten Arten ihr Wissen und Daten verwalten und pflegen, kann schnell eine hohe Anzahl verschiedener semantischer Beziehungen entwickeln, die für die Benutzer von Interesse sind. Die Detailstufe, mit der die Typen verschiedener Instanzen erfasst werden, kann sehr hoch werden, was zu einer unübersichtlichen Anzahl von Typen und Assoziationen führt, deren Eindeutigkeit nicht mehr gegeben sein muss. Eine von jedem Benutzer erweiterbare Ontologie birgt somit die Gefahr, dass die semantischen Beziehungen, die eigentlich gerade dazu gedacht sind von Algorithmen auswertbar zu sein, keine brauchbaren Informationen mehr liefern, da die Eindeutigkeit nicht mehr gegeben ist. Mehrere Typen mit unterschiedlichen Namen, die den gleichen Sachverhalt beschreiben oder ein nicht eindeutig gewählter Name

einer Beziehung, die an mehreren Stellen mit unterschiedlicher Bedeutung genutzt wurde, machen ein solches System leicht unbrauchbar. Für den Aufbau und die Pflege der Ontologien bedarf es also leicht nachvollziehbarer Regeln, die die Übersichtlichkeit gewährleisten. Durch eigene Namespaces für jede genutzte Ontologie kann die Übersichtlichkeit bereits deutlich verbessert werden.

Eine gute grafische Schnittstelle, die den Benutzer bei der Pflege der Ontologien unterstützt und die Gefahr von Uneindeutigkeit und Verdopplungen minimiert, ist in der Software, die in dieser Arbeit genutzt werden soll, nicht vorhanden. Um hier die Funktionsfähigkeit eines evolutionären Algorithmus auf einem semantischen Web zu zeigen, wird von ausreichend versierten Benutzern des Systems ausgegangen. In diesem Kontext stellt diese Bedingung keine wesentliche Einschränkung der Aussagefähigkeit dar, da mit einem entsprechenden System zur geführten benutzerspezifischen Erweiterung von Ontologien in absehbarer Zeit zu rechnen ist.

2.2 Web 2.0

Der Begriff Web 2.0 kommt aus dem Umfeld des Verlegers Tim O'Reilly und wurde bei der Suche nach einem Begriff für den empfundenen Evolutionssprung, der im Internet auszumachen ist, geprägt. Tim O'Reilly beschreibt in seinem Artikel "What is Web 2.0" (http://www.oreilly.de/artikel/web20.html Datum: 06.11.2008) einen Wandel der im Internet angebotenen Dienste, deren Aufbau und Technologie und stellt alte und neue Dienste vergleichend gegenüber. Trotzdem ist der Begriff Web 2.0 im heutigen Sprachgebrauch nicht klar gefasst, da er von vielen aufgegriffen und in den unterschiedlichsten Kontexten verwendet wurde. Im wesentlichen wird versucht unter diesem neuen Begriff die Funktionen des Internet zusammenzufassen, die über die vormals weitgehend übliche statische Darstellung von Inhalten hinaus gehen. Viele Dienste, die heute als Beispiele für das Web 2.0 dienen, wären zu Zeiten der New Economy schon alleine wegen der technischen Begebenheiten wie der Bandbreiten gängiger Internetanschlüsse, sowie der Kompetenz der Internetnutzer mit dem Web gar nicht möglich gewesen [8].

Dem Benutzer einer Web 2.0-Anwendung wird es ermöglicht selber einen Teil des Internet zu gestalten und diese Inhalte zu vernetzen. Diese lassen sich in interaktiven Prozessen von den individuellen Benutzern anpassen. Dabei tritt die optische Repräsentation gegenüber dem Inhalte weiter in den Hintergrund. In großen weit verteilten Gruppen wird es ermöglicht gemeinsam Inhalte zu erarbeiten und Wissen in strukturierter Form zu speichern. Das Web wird somit aktiv gemeinsam gestaltet und wird zu einer Plattform die als alternative zum eigenen lokalen Rechner für den Ablauf von Prozessen genutzt werden kann.

Die Prinzipien und Zusammenhänge der einzelnen Bestandteile des Web 2.0 werden in der MindMap von Markus Angermeier (siehe Abbildung 2.2) dargestellt.

Abbildung 2.2: Mindmap zum Thema Web 2.0, entnommen aus http://www.nerdwideweb.com

Im Rahmen dieser Arbeit sollen aber vor allem die technischen Merkmale hervorgehoben werden, die für ein Web 2.0 nötig waren, da diese auch eine Grundlage für die hier entwickelten Ideen darstellen.

Ajax Ajax steht für Asynchonous Javascript and XML und besteht zu wesentlichen Teilen aus dem schon vorher bekannten XMLHttpRequest. Mit Hilfe von DOM (Document Object Model) manipulationen durch Javascript werden dynamisch geladene Teile in einer Website geändert. Der vormals gängige Request-Response-Zyklus, der Seite für Seite nacheinander läd, interpretiert und für den Benutzer darstellt, wird damit aufgebrochen. Ganze Abläufe und Prozesse können auf einer Seite stattfinden. Durch die asynchrone Abarbeitung der ausgelösten XMLHttpRequests wird die Nutzeroberfläche einer solchen Webapplikation nicht unnötig blockiert und vermittelt dem Benutzer eine hohe Reaktionsfreudigkeit. Diese Art flexibler dynamischer Oberflächen ist dem Bild nachempfunden, das ein Benutzer von seinen Desktopapplikationen kennt, in denen auch nicht für jede Änderung der komplette Fensterinhalt neu aufgebaut werden muss, sondern nur einzelne Bereiche ausgetauscht werden.

Nachteile ergeben sich bei der Nutzung von Ajax darin, dass die Browserhistory nicht mehr automatisch gepflegt werden kann. Der Programmierer einer ajaxbasierten Anwendung muss somit entweder auf diese Funktion des Browsers explizit verzichten oder auch

via Javascript selber Einträge im Stack der Browserhistory ergänzen, die den Änderungen auf der gleich gebliebenen Seite entsprechen. Außerdem ist die Entwicklung barrierefreier Webanwendungen durch direkte Manipulation des DOM sehr schwierig, da diese von vielen Screenreadern nicht erfasst werden kann.

RSS und Atom RSS in seinen vielen Varianten und das weniger verbreitete aber nach RFC 4287 standardisierte Atom stellen einfachste Webservices dar. Diese bieten die Möglichkeit mit aktuellen Browsern oder entsprechenden Newsreadern Informationen zu abonnieren. Die wesentliche Neuerung besteht dabei darin, dass über diese Schnittstelle, aktiv Daten von der Website zu den Abonnenten transportiert werden und geht in dieser Hinsicht weit über die Funktionalität einfacher Links oder Bookmarks hinaus. RSS-Feeds stellen somit auch einen wesentlichen Bestandteil von Blogs dar, die wiederum ein zentrales Beispiel für eine Web 2.0 Anwendung sind.

Einbeziehung des Nutzers in die Entwicklung Die Entwicklung einer Web 2.0 Anwendung ist ein Prozess der sich nicht grundsätzlich in releasebasierte Entwicklungzyklen pressen läßt. Es entsteht oftmals ein dauerhafter Beta-Status, der den Nutzern bereits eine nahzu vollständige Nutzerumgebung bietet, die aber fortwährend weiterentwickelt und ergänzt wird. Dabei übernimmt der Nutzer an vielen Stellen die Position eines Testers. Dabei geht es nicht nur um den Test der reinen Funktionen sondern vielmehr um die Benutzerakzeptanz, die ermittelt werden soll.

Kapitel 3

Architektur

Dieses Kapitel beschreibt in einem ersten Schritt welche Rahmenbedingungen das für diese Arbeit genutzte System bieten muss. Darauf aufbauend werden die gewählten Systeme, deren Aufbau sowie das Zusammenspiel der einzelnen Komponenten, die im Rahmen dieser Arbeit genutzt oder erzeugt werden, erläutert.

3.1 Anforderungen an das System

Um die Anforderungen eines Systems, auf dem die Diplomarbeit aufbauen soll, zu ermitteln wurde von einer Software ausgegangen, wie sie einem aktuellen Kundenwunsch für eine interne Webapplikation, wie z.B. einem Firmenintranet, entsprechen könnte. Diese sollte für eine einfache Bedienbarkeit und um hohen Installationsaufwand zu vermeiden webbasiert sein. Um den einzelnen Benutzern ihren Aufgaben und Rechten entsprechende Oberflächen zu bieten, sollte die Anwendung im Sinne des Web 2.0 mit Hilfe eines Login individualisierbar sein. Jedem Nutzer wird dabei ein maximal mögliches Ausmaß an Autonomie in der Gestaltung und Nutzung seiner Oberfläche sowie bei der Erfassung eigener Inhalte geboten. Übergreifend über alle Bereiche der Anwendung sollen semantische Verknüpfungen zwischen Objekten der Anwendung definiert werden können. Diese werden von den Benutzern selber gepflegt und erweitert.

Die Software sollte in Java geschrieben sein und einfache Schnittstellen für die Abfrage der verwalteten Daten bieten. Die Erweiterbarkeit der grafischen Oberfläche sowie der internen Prozesse sollte konfigurativ möglich sein.

Anforderungen:

- Java als grundlegende Programmiersprache
- Konfigurative Erweiterbarkeit
- Webbasiert

- Individualisierbar (Web 2.0)
- Unterstützt semantisches Web mit erweiterbaren Ontologien
- Hohe Autonomie seitens des Benutzers

3.2 Aufbau der Software

Im ersten Teil dieses Kapitels wird die gewählte Serversoftware IkeWiki beschrieben. Diese dient als exemplarische Basis dieser Arbeit und erfüllt die oben genannten Anforderung weitestgehend. Im zweiten Teil wird dann auf die Architektur der semantischen Navigation eingegangen.

3.2.1 Die genutzte Serversoftware : IkeWiki

Da eine Software wie in der Einleitung skizziert, die in einem Portal sowohl Wikis als auch Blogs, Foren und CMS unter einer Oberfläche zur Verfügung stellt und auf einem semantischen Web basiert in der Form leider noch nicht verfügbar ist, musste im Rahmen dieser Diplomarbeit auf ein weniger komplexes System gesetzt werden, das aber möglichst viele der gewünschten Funktionen abdecken sollte. Die Wahl viel dabei auf ein semantisches Wiki, da dieses viele Strukturen eines Forums oder eines Blogs zumindest nachbilden kann. Als javabasiertes Opensourceprojekt konnten durch IkeWiki[6] die meistens Anforderungen dieser Arbeit abgedeckt werden.

IkeWiki ist ein semantisches Wiki, das unter der Leitung von Sebastian Schaffer an der Salzburg Research Forschungsgesellschaft entwickelt wurde. Der Fokus lag dabei darauf ein Wissensmanagementsystem auf Basis eines semantischen Webs zu schaffen. Dies wurde dadurch verwirklicht, dass das Grundsystem eines normalen Wikis um Funktionen erweitert wurde, die es Benutzern mit speziellen Rechten erlauben, die semantischen Beziehungen zwischen Artikeln des Wikis zu ergänzen und zu verwalten.

Artikel werden im IkeWiki wie in anderen Wikis auch in Form von Texten in einer angepassten Wikisyntax erfasst, die in diesem Fall der bekannten Syntax des MediaWiki entspricht, welches unter anderem von der Online-Enzyklopädie Wikipedia (http://www.wikipedia.org) genutzt wird. Diesen Artikeln werden automatisch bereits Typen zugeordnet. So können bereits beim Anlegen eines Artikels die Typen für Bilder, Multimediainhalte oder einfache Texte vergeben werden. Jeder Benutzer hat eine eigene benutzerspezifische Seite, auf der er seine privaten Links und Informationen verwalten kann. Diesem Artikel wird automatisch der Typ für einen Benutzer "ikewiki:user" zugeordnet. Alle weiteren Artikel die ein Benutzer anlegt erhalten eine "ist Autor von"-Beziehung zu der entsprechenden Benutzerseite.

IkeWiki baut dabei auf einer Reihe von Ontologien auf, die bei der Installation des Systems direkt eingerichtet werden. Enthalten sind dabei eigene Ontologien zur Verwaltung

von Namespaces, Benutzern, sogenannten Wiklets und einiger spezieller Navigationsstrukturen wie z.B. Tutorials und Diskussionen. Diese bauen auf im Internet frei verfügbaren Ontologien auf, die durch die Vererbungshierarchie der Typen entsprechend miteinander verknüpft sind. So entspricht beispielsweise der Typ "ikewiki:user" immer auch dem Typ "foaf:Person". In IkeWiki verpflichtend enthalten sind die folgenden Ontologien:

RDF Schema Definiert grundlegende Konzepte wie "class", "resource", und "property", sowie die Eigenschaften "type", "subclass", und "comment".

OWL "Ontology Web Language" Erlaubt das definieren komplexerer Ontologien.

Dublin Core Erlaubt das Beschreiben von Meta-Eigenschaften von textbasierten Inhalten, wie "creator", "publisher", "rights", etc.

Friend of a Friend (FOAF) Ontologie Wird von IkeWiki für das Speichern nutzerspezifischer Daten genutzt.

SIOC (Semantically Interlinked Online Communities) Bietet Methoden zur Verknüpung verschiedener Diskussionsplattformen wie Blogs, Foren und Mailinglisten.

DILIGENT Argumentations Ontologie Hilft bei der Entwicklung von Ontologien Abhängigkeiten und Diskussionen festzuhalten.

IkeWiki Base Ontology Beinhaltet Typen für die interne Datenrepräsentation von IkeWiki (z.B. Bilder).

IkeWiki Tutorial Navigation Erlaubt die Naviagion in einem Tutorial, beispielsweise das IkeWiki-Hilfesystem.

IkeWiki Help System Stellt die Dokumentation und Hilfen zur Verfügung.

Simple Knowledge Organisation System (SKOS) Stellt ein Model für die grundlegende Struktur und Inhalte von Konzepten wie Thesauri, Taxonomien, Glossaren und ähnlichem.

Die Zuordnung von Typen zu einzelnen Artikeln des Wikis oder den Links in dem zugehörigen Artikel geschieht im IkeWiki über eine eigene Oberfläche, die vom eigentlichen Editor für die Texte getrennt wurde. Diese ist über den Tab "Annotations" erreichbar. Unter der Überschrift des Artikels werden alle bereits zugeordneten Typen eines Artikels angezeigt. Über die Buttons "+" und "-" lassen sich weitere Typen hinzufügen oder entfernen. Auch hinter jedem nicht automatisch generierten Link des Artikels erscheinen diese Buttons. Hier lassen sich die semantischen Beziehungen zu den jeweils verlinkten Artikeln ergänzen. Dabei werden dem Benutzer nur solche Beziehungen zur Auswahl angeboten, die mit den Typen der verlinkten Artikel konform sind. Ontologien lassen sich auch über die

Oberfläche des Wikis neu erstellen oder ergänzen. Dazu muss der Benutzer über die Links "Create Class" oder "Create Property" die einzelnen Typen und Eigenschaften erfassen. Komplette Ontologien lassen sich aber auch in verschieden Formaten einfach importieren und wieder exportieren.

Um die Oberfläche von IkeWiki anzupassen, stehen einem Entwickler mehrere leicht zu konfigurierende Ansätze zur Verfügung. Hier soll aber nur der für diese Arbeit genutzte Weg beschrieben werden.

Portlets Ein Portlet im Sinne des IkeWiki entspricht nicht dem im JSR-168 (http://jcp.org/aboutJava/communityprocess/final/jsr168/ Datum:31.10.2008) verwendeten Begriff eines Portlets in einer Portalumgebung. Da IkeWiki keine Portalumgebung nach dem JSR-168 bietet, ist der Begriff hier irreführend. Dennoch wird ein Portlet im Context des Wikis auf ähnliche Weise genutzt und stellt hier einen abgeschlossenen Bereich an Rand der Wikioberfläche dar, der dem Benutzer Navigationsmöglichkeiten oder weitere contextsensitive Informationen zu dem betrachteten Wikiartikel bietet. Portlets werden im IkeWiki bereits für die vorhandene Navigation, eine Anzeige der Verknüpfungen eines Wikiartikels, sowie für diverse administrative Aufgaben genutzt.

Weitere Portlets lassen sich über die Konfigurationsdatei /WEB-INF/portlets.xml ergänzen. In dieser wird für jedes Portlet ein weiterer Eintrag vorgenommen, der die Position, die zugehörige JSP, sowie einige ergänzende Parameter enthält.

```
1 <portlet id="semantic_navigation">
2   <jsp>components/portlets/semnav.jsp</jsp>
3   <acceptable-parameter>uri</acceptable-parameter>
4   <acceptable-parameter>title</acceptable-parameter>
5   <acceptable-parameter>position</acceptable-parameter>
6   <render-lazy>true</render-lazy>
7   <position>right</position>
8 </portlet>
```

Algorithmus 3.1: Eintrag in die portlet.xml für ein neues Portlet

IkeWiki Architektur IkeWiki ist nach dem Schichtenmodell aufgebaut. Als Basis für die Datenhaltung wird eine PostgreSQL-Datenbank genutzt. Diese dient sowohl der Speicherung der Inhalte des Wikis, als auch derer semantischen Verknüpfungen.

Diese Inhalte sind über zwei Module abrufbar. Der Page Store liefert den Inhalt eines Artikels im Wiki in einem speziellen XML-Format namens WIF (Wiki Exchange Format). Dieses Format wurde speziell für den Austausch von Daten zwischen verschiedenen Wikis entwickelt.

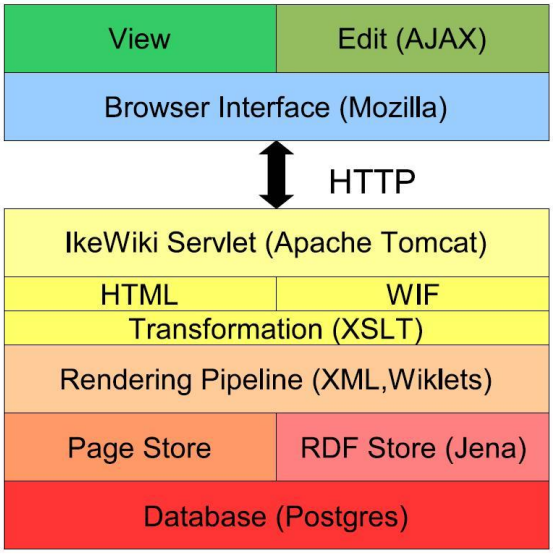

Abbildung 3.1: IkeWiki Architekur entnommen aus [6]

Im RDF Store wird auf Basis des Jena RDF Framework (http://jena.sourceforge.net) die Semantik des Wikis gespeichert. Über diese Schnittstelle erhält die für dieses Arbeit entwickelte semantische Navigation die notwendigen Daten über die semantischen Beziehungen einzelner Artikel. Da die Verbreitung von Jena unter den javabasierten semantischen Applikationen relativ hoch ist, kann durch eine entsprechende Umsetzung dieser Schnittstelle die semantische Navigation auch für andere auf Jena aufbauende Systeme genutzt werden. Zusätzlich zu der reinen Abfrage von semantischen Beziehungen bietet der RDF Store einen Interpreter für SPARQL. Dies ist eine an SQL angelehnte Abfragesprache, die es ermöglicht dynamische erzeugte Inhalte über die semantischen Beziehungen eines Objekts in einem Wikiartikel direkt abzufragen und darzustellen.

Über die Rendering Pipeline werden schließlich die Informationen aus Page Store und RDF Store kombiniert und mit Hilfe einer XSLT-Transformation das auszuliefernde HTML erzeugt.

3.2.2 Architektur der Komponenten der semantischen Navigation

Die semantische Navigation besteht aus mehreren Modulen, die mit Hilfe des Springframeworks konfiguriert und verbunden werden. Der Aufbau und das Zusammenspiel der einzelnen Module (siehe Abbildung 3.2) sollen in den folgenden Kapiteln beschrieben werden.

Die Schnittstelle zum semantischen Web

Um eine möglichst gute Basis für die Bewertung einzelner Individuen eines evolutionären Algorithmus zu haben, sollen möglichst alle Eingaben, die der Benutzer tätigt, erfasst

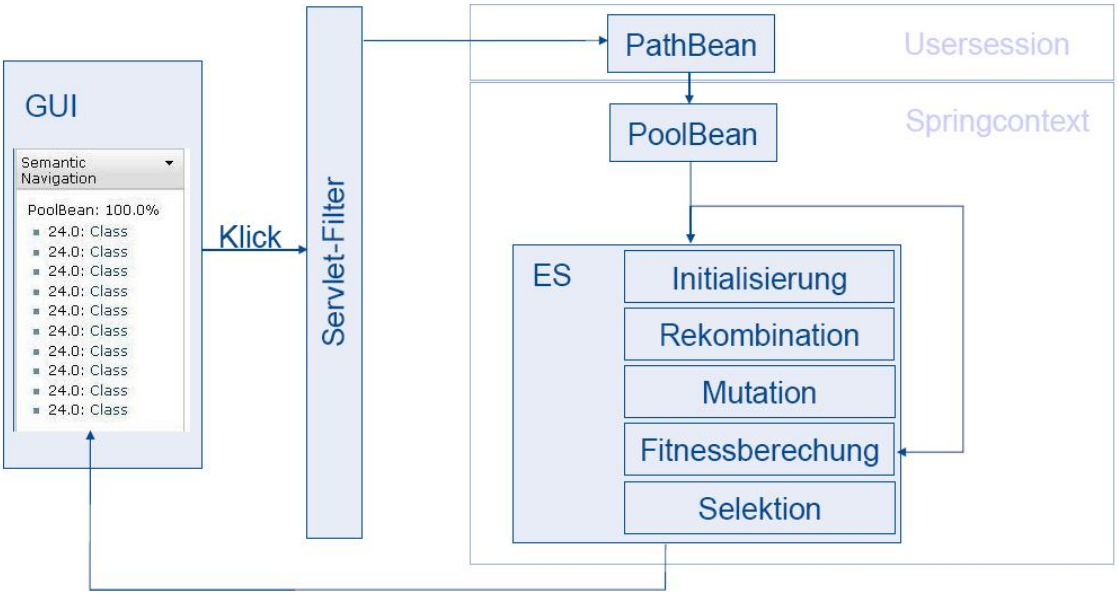

Abbildung 3.2: Architekur der semantischen Navigation

und genutzt werden. Um die Systemanforderung von IkeWiki zu erfüllen wird hier der Tomcat Server in der Version 6.0 als Implementierung der Servlet- und JavaServer Pages-Technologie genutzt. Die geeignete Stelle um die Eingaben des Benutzers abzufangen ist ein Filter, der sämtliche Aktionen registriert, die zum Laden einer neuen Instanz führen. Die Oberfläche von IkeWiki besteht aus eine Vielzahl einzelner JSPs, die parallel geladen und zu einer Oberfläche zusammengefügt werden. Um zu verhindern, dass der Filter für jede JSP, die zu der aktuellen Seite gehört ausgelöst wird, ist das Filtermapping auf genau eine JSP ausgrichtet (siehe Algorithmus 3.2, Zeile 7), die auf jeder Seite nur genau einmal vorkommt.

```
1 <filter>
2   <filter-name>LinkPathFilter</filter-name>
3   <filter-class>de.materna.semanticnav.filter.LinkPathFilter</filter-
        class>
4 </filter>
5 <filter-mapping>
6   <filter-name>LinkPathFilter</filter-name>
7   <url-pattern>/components/portlets/semnav.jsp</url-pattern>
8 </filter-mapping>
```

Algorithmus 3.2: Eintrag in die WEB-INF/web.xml für ein Filter

Der `LinkPathFilter` hat die Aufgabe die Eingaben des Benutzers darauf zu prüfen, ob sie einen Pfad enthalten und diesen dann unter dem Namen des Benutzers in der

PoolBean abzuspeichern. Der Filter überprüft dabei immer wenn er ausgelöst wird, ob es eine semantische Verknüpfung zwischen der vorangegangen Instanz und der neuen Instanz gibt. Existiert diese, so wird die neue Instanz auf einen Stack in der PathBean gelegt. Existiert sie nicht wird überprüft ob in der PathBean bereits ein Pfad liegt, der mindestens die Länge zwei haben muss. Dieser wird dann in der PoolBean gespeichert. Die neue Instanz bildet den Anfang eines neuen Pfads und wird somit als erstes Objekt auf einen neuen Stack in der PathBean gelegt. Die PathBean selber liegt in der Session des jeweiligen Benutzers und kann so dessen aktuellen Pfad speichern.

Bei der Erkennung ob zwei Instanzen eine Beziehung zueinander haben, ist es nicht sinnvoll alle möglichen Relationen als Teil eines Pfads zuzulassen. Da IkeWiki einige Relation automatisch vergibt, würden so unendlich lange Pfade entstehen. Um dies zu vermeiden wird die Menge der gefunden Relationen durch eine Reihe von frei konfigurierbaren Filtern (siehe Algorithmus 3.3) geschickt, die für diesen Anwendungsfall ungünstige Relationen entfernen. Dabei kann sowohl auf der Basis einzelner Relationen gefiltert werden, als auch auf der Basis ganzer Namespaces.

```
1  <bean id="semanticResourceFilterList" class="java.util.ArrayList" >
2    <constructor-arg>
3    <list>
4      <ref local="de.materna.semanticnav.util.filter.
         ResourceTitleFilter" />
5      <!-- ref local="de.materna.semanticnav.util.filter.
         NamespaceFilter" /-->
6    </list>
7    </constructor-arg>
8  </bean>
```

Algorithmus 3.3: Liste der anzuwendenden Filter in WEB-INF/applicationContext.xml

Wie in Algorithmus 3.4, Zeile 4 und 5 zu sehen werden die Relationen "ikewiki:NavigationalLink" sowie "ikewiki:hasAuthor" in den Pfaden nicht berücksichtigt. In beiden Fällen handelt es sich um Relationen die automatisch von IkeWiki vergeben werden und durch ihr extrem häufiges vorkommen, das Verhalten des evolutionären Algorithmus verfälschen würden. Der Filter bezüglich der Namespaces wird in der hier verwendeten Konfiguration nicht gebraucht. Sollte ein Unternehmen die Ergänzungen der Ontologie seitens der Benutzer in einem Namespace sammeln, könnte es hier interessant sein, die Pfade speziell auf diesen Namespace einzugrenzen.

PoolBean Die PoolBean dient der semantischen Navigation als Speicher für die Pfade, die ein Benutzer wirklich genutzt hat. Zu jedem Benutzer wird eine fest konfigurierte (siehe Algorithmus 3.5, Zeile 2) Anzahl an Pfaden gespeichert, wobei der älteste Eintrag aus

```
1  <bean id="de.materna.semanticnav.util.filter.ResourceTitleFilter"
       class="de.materna.semanticnav.util.filter.ResourceTitleFilter" >
2    <property name="bannedTitles" >
3      <list>
4        <value>ikewiki:NavigationalLink</value>
5        <value>ikewiki:hasAuthor</value>
6      </list>
7    </property>
8  </bean>
9  <bean id="de.materna.semanticnav.util.filter.NamespaceFilter" class="
       de.materna.semanticnav.util.filter.NamespaceFilter">
10   <property name="allowedNamespaces">
11     <list>
12       <value>http://xmlns.com/foaf/0.1/</value>
13     </list>
14   </property>
15 </bean>
```

Algorithmus 3.4: Filterkonfiguration in WEB-INF/applicationContext.xml

der Liste entfernt wird, sobald die Liste die maximale Größte überschreitet. Die PoolBean Größe stellt dabei eine Art Erinnerungsvermögen der semantischen Navigation dar. Umso höher der Wert gewählt wird, desto länger nehmen Pfade, die älteren Interessen des Benutzers entsprechen, Einfluss auf die aktuelle Fitnessbewertung.

```
1 <bean class="de.materna.semanticnav.bean.PoolBean" id="de.materna.
    semanticnav.bean.PoolBean">
2   <property name="size" value="30"/>
3 </bean>
```

Algorithmus 3.5: Konfiguration der PoolBean in WEB-INF/applicationContext.xml

Für eine Implementierung, die im Umfeld eines Unternehmens eingesetzt werden soll, liegt hier Optimierungsbedarf, da in der aktuellen Implementierung, alle Pfade aller Nutzer im Speicher gehalten werden. Diese Datenmenge sollte in einem konkreten Anwendungsfall in eine Datenbank ausgelagert werden.

Aufbau und Konfiguration der zentralen Serverkomponente

Die zentrale Komponente der semantischen Navigation übernimmt die Aufgabe die Daten aus der PoolBean zu überwachen, diese an einen beliebigen evolutionären Algorithmus weiterzuleiten und die Ergebnisse in Form einer vorbestimmten Zahl von Links an die grafische Oberfläche zu übermitteln.

```
1 <bean class="de.materna.semanticnav.bean.SemanticNavBean" id="de.
     materna.semanticnav.bean.SemanticNavBean">
2   <property name="poolBean" ref="de.materna.semanticnav.bean.PoolBean">
       </property>
3   <property name="numberOfLinks" value="10"/>
4   <property name="ea" ref="de.materna.semanticnav.ea.CBNES"/>
5 </bean>
```

Algorithmus 3.6: Konfiguration der SemantivNavBean in WEB-INF/applicationContext.xml

Diese Aufgabe übernimmt die SemanticNavBean, die wiederum mit Hilfe des Spring Framework konfiguriert wird. Sobald die PoolBean für den aktuellen Benutzer Daten enthält löst die SemanticNavBean bei jeder Aktion des Benutzers, die durch den beschrieben `LinkPathFilter` erkannt wird, einen Iterationsschritt des vorgegebenen evolutionären Algorithmus aus. Dieser muss das einfache Interface `EAInterface` implementieren um hier als evolutionärer Algorithmus eingehängt werden zu können (siehe Algortihmus 3.6, Zeile 4).

Außer der Referenz auf die PoolBean und den zu verwendenden evolutionären Algorithmus bietet die SemanticNavBean noch eine einfache Möglichkeit die maximale Anzahl der Links, die dem Benutzer angezeigt werden sollen, einzustellen (siehe Algorithmus 3.6, Zeile 3). Diese Anzahl sollte nicht zu hoch gewählt werden um zu verhindern, dass dem Benutzer zu viele Links angezeigt werden, die noch keinem gut optimierten Ergebnis entsprechen.

Aufbau des evolutionären Algorithmus Die hier gewählte Implementation eines evolutionären Algorithmus entspricht dem Aufbau einer Evolutionsstrategie [7]. Die Individuen durchlaufen nach einer einmaligen Initialisierung in jedem Iterationsschritt die Phasen der Rekombination, Mutation und Selektion. Die Fitnesswerte aller Individuen müssen dabei in jeder Runde neu berechnet werde, da das Interesse des Benutzers an verschiedenen Themen über die Zeit variabel ist. Somit besteht eine Evolutionsstrategie, wie in Abbildung 3.2 skizziert, aus den folgenden Modulen:

- Initialisierung
- Rekombination
- Mutation
- Berechnung des Fitnesswerts
- Selektion

Diese sind durch entsprechende Interfaces und eine abstrakte Oberklasse für jedes Modul leicht austauschbar, so dass verschiedene Vorgehensweisen auf ihre Eignung für die semantische Navigation getestet werden können.

Der Zufallsgenerator Für alle Module steht ein Zufallsgenerator zur Verfügung, der als Singleton konfiguriert ist und somit für das ganze System gleich ist. Dies dient dazu über einen festen Randomseed sämtliche Vorgänge in den Modulen reproduzierbar zu machen. Der Seed wird konfigurativ (siehe Algorithmus 3.7, Zeile 3) beim Start der Webapplikation festgelegt und veranlasst den Zufallgenerator nach einen Neustart immer die gleiche Folge von Zufallszahlen zu generieren.

```
1 <!-- Weitere Util-Klassen -->
2 <bean class="java.util.Random" id="java.util.Random">
3   <constructor-arg type="long" value="31415"/>
4 </bean>
```

Algorithmus 3.7: Konfiguration des Zufallsgenerators in WEB-INF/applicationContext.xml

Kapitel 4

Aufbau des Evolutionären Algorithmus

Grundsätzlich ist es möglich fast jede Art von evolutionären Algorithmen für die hier bearbeitete Problemstellung entsprechend anzupassen und zu nutzen. Aufgrund ihrer hohen Anpassungsfähigkeit [2] an Probleme mit verschiedenen Rahmenbedingungen wurde hier eine Evolutionsstrategie gewählt. Diese wurde später um einen Cluster-Based-Niching-Ansatz erweitert. Alle Algorithmen, die für die semantische Navigation genutzt werden sollen, müssen das Interface EAInterface implementieren, welches zwei einfache Methoden enthält. Diese dienen der semantischen Navigation zum Anstoßen eines einzelnen Optimierungsschritts, sowie dem Abruf der anzuzeigenden Links.

In den folgenden Kapiteln sollen nun die implementierten Anpassungen und Varianten der verschiedenen Module näher erklärt werden.

4.1 $(\mu/\rho + \lambda)$ Evolutionsstrategie

Bei der $(\mu/\rho+\lambda)$ Evolutionsstrategie durchlaufen die Individuen nacheinander die Schritte der Rekombination, Mutation und Selektion. Dabei sind in der Elterngeneration immer μ Individuen enthalten, aus denen jeweils ρ zufällig gewählte Individuen in der Rekombination zu einem oder mehreren neuen Individuen vereinigt werden. Dies wird solange wiederholt, bis insgesamt λ neue Individuen erschaffen wurden. In der Phase der Mutation werden die neuen Individuen mit einem zufallsbasierten Mutationsoperator bearbeitet, der für eine bessere Streuung der Individuen im Suchraum zuständig ist. Da es für diese Anwendung wichtig ist alte Individuen zu erhalten, um dem Benutzer der semantischen Navigation kein zu sprunghaft wechselndes Bild zu bieten, wurde hier die "+"-Variante der Evolutionstrategie gewählt, in der die Individuen der Elterngeneration nicht verworfen, sondern vor der Selektion mit der Menge der neuen Individuen vereinigt werden. Um dem möglicherweise stetig wechselnden Interesse des Benutzers gerecht zu werden, muss

im nächsten Schritt für jedes Individuum der Fitnesswert neu berechnet werden. Der Algorithmus arbeitet also auf einer über die Zeit dynamischen Fitnesslandschaft. Auf Basis der aktuellen Fitnesswerte können dann verschiedene Arten der Selektion durchgeführt werden, durch die die μ Individuen für die nächste Elterngeneration bestimmt werden. Dabei wird hier nicht nur die diskriminierende Selektion nach dem Fitnesswert genutzt, wie sie für einen solche Evolutionsstrategie üblich ist, sondern auch die Möglichkeit gegeben, alternative, nicht diskrimierende Selektionsverfahren zu nutzen, die im Kapitel 4.8 "Selektionsverfahren" näher beschrieben werden.

4.2 Clustering Based Niching EA (CBN-EA)

Der Sinn eines Clustering Based Niching EA (CBN-EA)[4] liegt darin, die Diversität eines evolutionären Verfahrens deutlich zu erhöhen und so auch möglichst viele lokale Optima in einem Suchraum zu finden. Einfache evolutionäre Algorithmen sind dafür gedacht nach möglichst kurzer Zeit ein als global angenommenes Optimum zu finden. Alle Individuen konvergieren also bereits nach kurzer Zeit gegen dieses Optimum. Im Kontext der semantischen Navigation bedeutet dies, dass dem Benutzer nur ein Link vorgeschlagen werden kann, der seinem Hauptinteresse entsprechen würde. Wesentlich interessanter wäre es allerdings zu jedem Interessengebiet mindestens einen Link zu finden.

Durch den CBN-EA wird die Population, auf der eine Evolutionsstrategie arbeitet, in Gruppen aufgeteilt. Diese Gruppen ergeben sich aus dem Abstand der Individuen im Suchraum zueinander. Individuen innerhalb einer Gruppe entsprechen im Kontext eines evolutionären Prozess einer Spezies, die sich dadurch auszeichnet, dass die zugehörigen Individuen sich nur noch untereinander fortpflanzen und es keine Vermischung des Genpools mit Individuen anderer Spezies mehr gibt. Die Aufteilung und Bearbeitung der einzelnen Gruppen wird dabei in drei Phasen unterteilt:

Evolutionsphase (siehe Algorithmus 4.1, ab Zeile 3) In dieser ersten Phase wird auf jede vorhandene Spezies ein evolutionionäres Verfahren angewendet, welches auf das Auffinden eines einzelnen Optimums ausgerichtet ist. Hier wird die bereits beschriebene Evolutionsstrategie genutzt.

Differenzierungsphase (siehe Algorithmus 4.1, ab Zeile 6) In der Differenzierungsphase werden alle Spezies darauf getestet, ob sich weitere Subspezies durch den letzten Optimierungsschritt gebildet haben. Diese Subspezies werden im Folgenden dann einzeln weiter behandelt. Einzelne Individuen, die keiner besonderen Spezies angehören, werden immer der ersten Spezies D_0 zugeordnet (siehe Algorithmus 4.1, Zeile 10), die eine Art "Sammelbecken" darstellt.

Konvergenzphase (siehe Algorithmus 4.1, ab Zeile 12) Diese letzte Phase ermittelt aus jeder Spezies ein repräsentatives Individuum und fügt dieses in eine leere Menge

ein. In dieser Menge sollte jedes Element einer eigenen Spezies entsprechen. Wenn sich mehrere repräsentative Individuen zu einer Spezies zusammenfassen lassen, so werden die Spezies, aus denen die Individuen stammen, zu einer neuen kombiniert, da vermutlich beide gegen das selbe Optimum konvergieren.

```
1  D_0 = createInitialPop();
2  do{
3    //Species evolution phase
4    for (int i=0; i < AnzahlCluster;i++)
5      simulateEAGeneration(D_i);
6    //Species differentation phase
7    if (numOfClusters(D_0)>1) split(D_0);
8    for (int i=1; i < AnzahlCluster; i++){
9      if (numOfClusters(D_i)>1) split(D_i);
10     D_0.add(D_i.getLoners());
11   }
12   //Species convergence phase
13   TLP = createEmptyPop();
14   for (int i=1; i < AnzahlCluster; i++)
15     TLP.addCentroidOf(D_i);
16   if (numOfClusters(TLP)<TLP.size()) mergeDemes();
17 }while(!maxGenerationReached())
```

Algorithmus 4.1: Aufbau des Clustering Based Niching EA, entnommen aus [4]

Um eine Generation in Cluster unterteilen zu können muss also eine Möglichkeit geschaffen werden die Distanz zwischen zwei Individuen zu ermitteln. Dafür wird bei der Fitnessberechnung für jeden Typ eines Pfades protokolliert, ob dieser zu dem Fitnesswert beigetragen hat. Daraus ergibt sich für jedes Individuum eine Liste der fitnessrelevanten Typen. Umso ähnlicher sich die fitnessrelevanten Typen zweier Individuen sind, desto geringer ist ihr Abstand. Die Ähnlichkeit wird dabei als das Verhältnis zwischen gleichen und ungleichen Typen definiert. Um den Grenzwert für den Abstand zu bestimmen wird ein weiterer Parameter σ_{share} eingeführt. Für zwei Individuen gilt dann, dass sie zur gleichen Spezies gehören, wenn:

$$\frac{\text{Anzahl ungleicher Typen}}{\text{Anzahl gleicher Typen}} < \sigma_{share}$$

Weiterhin muss es möglich sein ein repräsentatives Individuum einer Spezies zu ermitteln. Dies kann beim Aufbau einer neuen Spezies leicht erreicht werden, da das Individuum, das eine neue Spezies begründet und somit zu allen weiteren Individuen dieser Spezies einen entsprechend geringen Abstand hat, immer als erstes in der Liste der Individuen stehen

muss. Als repräsentatives Individuum kann also einfach das erste Element der jeweiligen Liste gewählt werden.

4.3 Repräsentation der Individuen

Die Individuen (siehe Abbildung 4.1), mit denen ein evolutionärer Algorithmus arbeitet, stellen eine mögliche Lösung im Suchraum dar. Im Fall der semantischen Navigation sollte ein Individuum also ein Pfad im semantischen Web sein, der nach Möglichkeit einem Interesse des jeweiligen Benutzers entsprechen sollte. Jede Instanz im semantischen Web ist im IkeWiki als ein Objekt der Klasse WikiResource abrufbar. Über dieses lassen sich die Typen der Instanz abrufen. Dabei wird zwischen den Typen, die der Benutzer selbst zugeordnet hat und denen, die von IkeWiki automatisch vergeben wurden, unterschieden. Diese Unterscheidung kann bei der Mutation sowie der Rekombination genutzt werden.

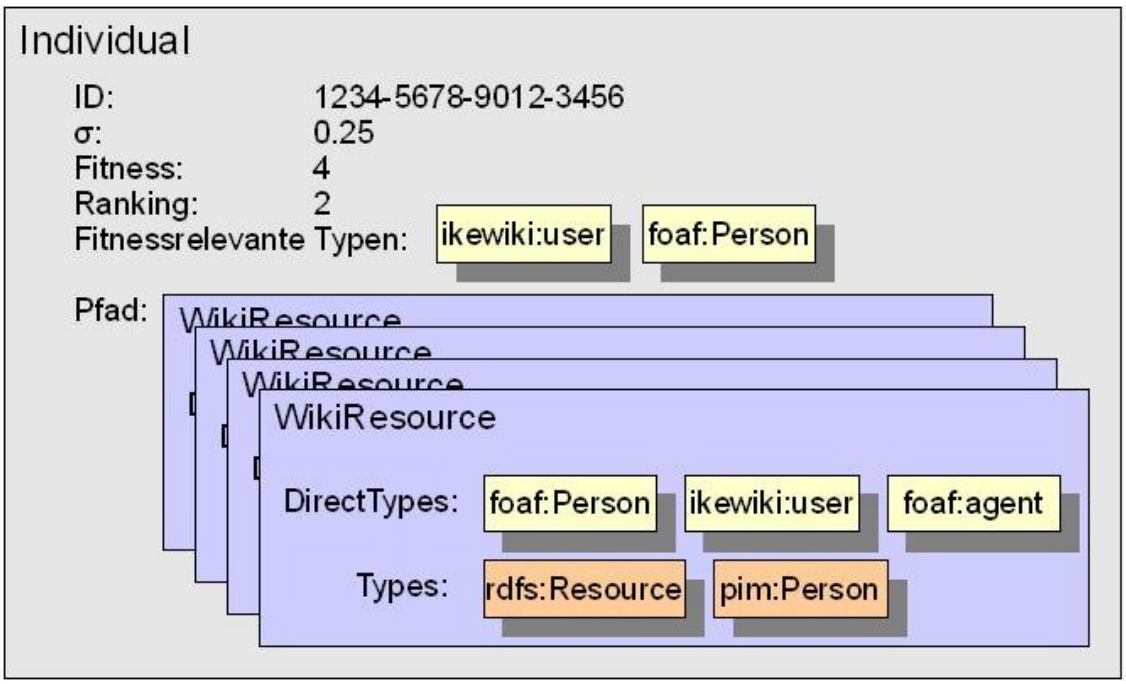

Abbildung 4.1: Aufbau eines Individuums

Zusätzlich zu der Liste der Instanzen enthält ein Individuum eine ID, die für ein direktes positives Nutzerfeedback genutzt werden kann. Über den im Individuum abgelegten Wert σ wird die Stärke der Mutation gesteuert. Des weiteren lassen sich der jeweilige Fitnesswert sowie ein für die Selektionsverfahren genutztes Ranking speichern. Bei der Berechnung des Fitnesswerts können die Typen, die zu einer Erhöhung des Werts führen in der Liste der fitnessrelevanten Typen gespeichert werden. Diese werden später für die Bestimmung von Abständen zwischen Individuen benötigt.

4.4 Initialisierung

Die Initialisierung der Startpopulation findet einmalig beim Start der semantischen Navigation statt. Um einen möglichst guten Ausgangspunkt für eine Suche zu liefern gibt es verschiedene im Folgenden erläuterte Möglichkeiten.

Random Walk Ausgehend von einer Instanz des semantischen Webs wird nach allen Tripeln (Subjekt, Prädikat, Objekt) gesucht, in denen diese Instanz das Subjekt darstellt. Aus diesen wird zufällig gleichverteilt ein Tripel gewählt. Das zugehörige Objekt wird an den zufälligen Pfad angehängt, wenn es vom Typ WikiResource ist. Dies stellt sicher, dass die Pfade nicht durch den Verweis auf ein Literal frühzeitig enden. Um die Länge eines Pfads dennoch einzuschränken, wird (siehe Algorithmus 4.2, Zeile 3) eine maximale Pfadlänge angegeben. Ein so entstandener Pfad stellt sicher, dass die zugehörigen semantischen Verknüpfungen im Wiki existieren und somit der Benutzer die Möglichkeit hat diesen Pfad auch zu gehen.

Die Wahl des Ausgangspunktes für einen solchen Pfad ist schwierig. Einen möglichen Ansatzpunkt bildet die zufällige Auswahl einer beliebigen Instanz des semantischen Webs. Um aber schon am Anfang eine möglichst den Benutzerinteressen entsprechende Füllung der ersten Generation zu erhalten, kann das Benutzerobjekt als Instanz selbst eine Möglichkeit bieten. Wenn von einem bereits in Benutzung befindlichen System ausgegangen wird, kann der jeweilige Benutzer seine Bookmarks, Interessen und ähnliches bereits von seiner eigenen benutzerspezifischen Seite aus verlinkt und mit entsprechender Semantik versehen haben. Dies kann unter dieser Voraussetzung also ein guter Startpunkt für einen Pfad sein.

```
1 <!-- Initialisierung -->
2 <bean class="de.materna.semanticnav.ea.initialisation.
    MueRhoLambdaInitialisation" id="de.materna.semanticnav.ea.
    initialisation.MueRhoLambdaInitialisation">
3   <property name="maxPathLength" value="5"/>
4   <property name="rand" ref="java.util.Random"/>
5 </bean>
6 <bean class="de.materna.semanticnav.ea.initialisation.
    MueRhoLambdaInitialisationWarmsurfen" id="de.materna.semanticnav.ea
    .initialisation.MueRhoLambdaInitialisationWarmsurfen">
7   <property name="rand" ref="java.util.Random"/>
8   <property name="poolBean" ref="de.materna.semanticnav.bean.PoolBean">
      </property>
9 </bean>
```

Algorithmus 4.2: Konfiguration der Initialisierung in WEB-INF/applicationContext.xml

Initialisierung durch Benutzereingaben Eine weitere Möglichkeit der Initialisierung ergibt sich durch die Eingaben des Benutzers selbst. Durch den LinkPathFilter werden bereits alle Benutzereingaben abgefangen, analysiert und in Form von Pfaden in der PoolBean abgelegt. Diese dienen im wesentlichen dazu die Interessen des Benutzers zu repräsentieren und die Berechung eines Fitnesswerts zu ermöglichen. Für eine Initialisierung wäre es aber möglich die Pfade der PoolBean in Individuen des Algorithmus umzuwandeln und diese als Startpopulation zu nutzen.

Für den Benutzer ergibt sich dadurch ein geändertes Verhalten der semantischen Navigation, da diese erst in der Lage ist Links darzustellen, wenn die PoolBean einen Füllstand erreicht hat, der der Größe der Startpopulation entspricht. Grundsätzlich muss dafür natürlich die Vorbedingung gegeben sein, dass die Größe der PoolBean mindestens der Größe der Elterngeneration μ entspricht.

4.5 Berechnung des Fitnesswerts

Der Fitnesswert eines Individuums stellt eine Funktion dar, die je nach Art des zu bearbeitetenden Problems maximiert oder minimiert werden soll. Sie setzt sich zum einen aus einem Wert zusammen, der die aktuelle Qualität eines Individuums beschreibt. Dieser Wert kann beispielsweise umso höher sein, desto besser die Lösung ist, die das Individuum repräsentiert. Von diesem Qualitätswert wird auf der anderen Seite ein Bestrafungsterm abgezogen, der die Gesamtfitness von Individuen verringert, die z.B. eine Nebenbedingung verletzten.

In den folgenden Kapiteln wird nun eine Möglichkeit beschrieben das Interesse eines Benutzers in einem Fitnesswert zu erfassen und dabei mit Hilfe verschiedener Bestrafungsfunktionen die durch das semantische Web gegebenen Rahmenbedingungen zu erfüllen.

4.5.1 Bewertung der Güte eines Individuums

Die Bewertung eines Individuums soll wiedergeben wie hoch das Interesse eines Benutzers an dem Thema ist, welches durch die semantischen Verknüpfungen und Typen des Pfades repräsentiert werden. Wie das Interesse eines Benutzers aus seinem Klickverhalten abzuleiten ist und welche Teile eines semantischen Webs dafür genutzt werden können, ist zunächst einmal eine Frage der Interpretation.

Die PoolBean enthält für jeden Benutzer eine Reihe von Pfaden, die dieser tatsächlich gegangen ist. Diese geben ein Bild der aktuellen Interessen und Themenbereiche dieses Benutzers wieder. Um nun ein Individuum dahingehend zu bewerten, ob es diesen Interessengebieten entspricht soll hier ein Vergleich auf Basis der Typen durchgeführt werden. Jedes Individuum, das bewertet werden soll, wird mit jedem Pfad aus dem Pool verglichen und gezählt, wie lang die Teilstücke des Pfads sind, bei denen die Instanzen gleiche Typen

aufweisen. Die Längen dieser Teilstücke werden aufsummiert und so ein Wert ermittelt, der die thematische Ähnlichkeit eines Individuums mit dem Pool darstellt.

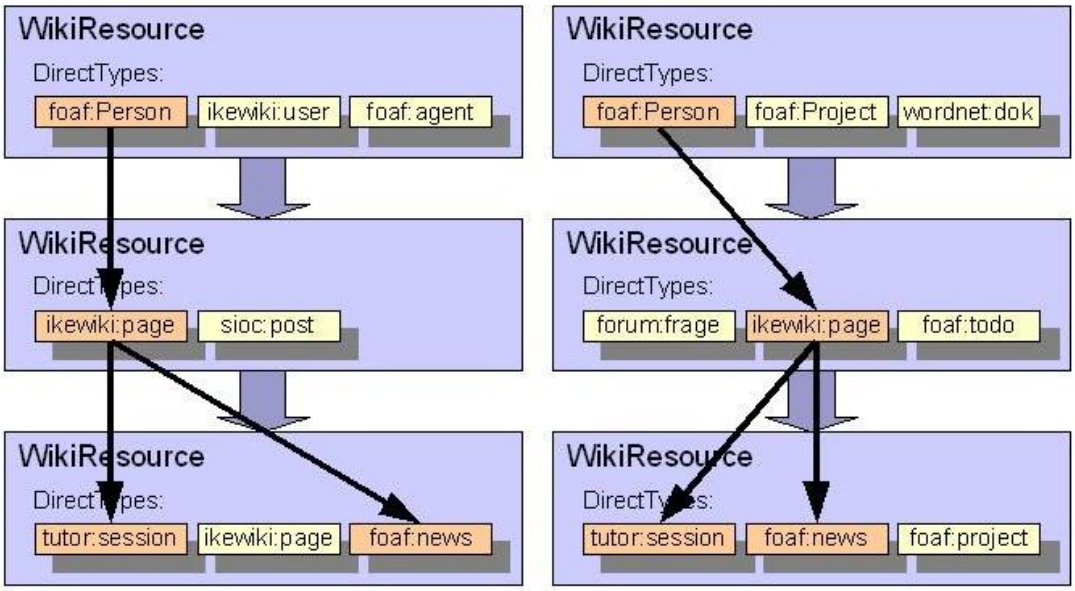

Abbildung 4.2: Fitnessberechung durch Vergleich auf Basis der Typen

Jedes Individuum besteht aus einer Liste von Instanzen. Zu jeder Instanz wiederum existiert eine Liste von Typen, die dieser Instanz zugeordnet wurden. Dabei wird zwischen den automatisch zugeordneten und den von Benutzern selbst vergebenen Typen unterschieden. Die Typen, die seitens der Benutzer gesetzt wurden, werden im Kontext des IkeWiki als "Direct Types" bezeichnet. Sie stellen für die Bewertung der Individuen die geeignetste Informationsquelle dar. Für die Typen des Individuums wird zuerst ein Index aufgebaut, der zu jedem Typ eine Liste aller Positionen enthält, in denen er vorkommt (siehe Algorithmus 4.3, Zeile 3). Über alle Pfade der PoolBean wird dann iteriert und die Menge aller gleichen Teilbäume (siehe Abbildung 4.2) ermittelt, die sich in den Typen finden lassen. Die Tiefe eines Teilbaums wird mit seiner Breite multipliziert und auf die Gesamtfitness eines Individuums aufsummiert (siehe Algorithmus 4.3, ab Zeile 14).

Wie die hohe Anzahl ineinander verschachtelter Schleifen (siehe Algorithmus 4.3) bereits vermuten lässt, ist die Fitnessberechnung auf diesem Weg sehr rechenzeitintensiv, wenn von großen Populationen und einer hohen Anzahl Pfade in der PoolBean ausgegangen wird. Da in jedem Schritt die Fitness aller Individuen neu berechnet werden muss, um die geänderten Interessen des Benutzers zu berücksichtigen, muss die Laufzeit ausreichend effizient sein, um dem Server Anwortzeiten in für Webanwendungen gewohnten Größenordnungen zu ermöglichen. Dies kann allerdings mit dem hier vorgesehenen Algorithmus auf heute gängiger Hardware ohne Probleme erreicht werden.

Dadurch, dass der Vergleich der Pfade auf Basis der Typen und nicht der Instanzen geschieht, werden durch diese Fitnessberechnung genau die Individuen einen hohen Fitnesswert erreichen, die von einem Benutzer durch die Typisierung in einen ähnlichen Kontext gesetzt wurden wie die Pfade der PoolBean. Diese entsprechen somit einem ähnlichen Themengebiet.

```
1  for (Individual ind : generation) {
2    //Index der Typen des aktuellen Individuums aufbauen
3    HashMap<Typ, Positionen> index = IndividualUtils.getIndex(ind);
4    double fitness = 0;
5    //Vergleiche mit jedem Pfad aus der PoolBean
6    for (List path : poolBean) {
7      //Fuer jedes Instanz des Pfads
8      for (WikiResource instanz : path) {
9        //Fuer jeden direkten Typ einer Instanz
10       for (Typ directType : instanz) {
11         //Positionen des Typs im Individuum bestimmen
12         List positions = index.get(directType);
13         for (int position : positions){
14           //berechne Teilbaeume gleicher Typen ab der Position
15           fitness += calculateTreeDimension(position);
16         }
17       }
18     }
19   }
20 }
```

Algorithmus 4.3: Fitnessberechung durch Vergleich auf Basis der Typen

Gefahren ergeben sich in diesem Zusammenhang allerdings dadurch, dass ein Individuum Kreise enthalten kann. Semantische Beziehungen besitzen in der Ontologie oft ein inverses Element, welches die Bildung von Kreisen begünstigt. Wechselt ein Individuum also oft zwischen zwei Instanzen hin und her, die in beide Richtungen über entsprechende semantische Verknüpfungen verfügen, so kann es seinen Fitnesswert beliebig erhöhen ohne die Qualität des Ergebnisses wirklich zu verbessern. Da diese Kreise aber in keinem Kontext einen Sinn ergeben, kann schon während der Rekombination darauf geachtet werden, dass nur kreisfreie Individuen entstehen.

Mehrere auf diesem Algorithmus aufbauende Varianten der Fitnessberechnung wurden im Rahmen dieser Arbeit implementiert. Diese sollen im Folgenden beschrieben werden.

Bestrafung nicht-semantischer Pfade Um sicherzustellen, dass die Individuen wirklich Pfaden entsprechen, die im semantischen Web möglich sind, wird vor der eigentlichen

Fitnessberechnung für je zwei aufeinander folgende Instanzen des Pfades überprüft, ob es Tripel der Form (Subjekt, Prädikat, Objekt) gibt, bei denen der erste Teil des Pfades dem Subjekt und der zweite Teil dem Objekt entspricht. Für jede Stelle im Pfad, für die kein solches Tripel gefunden werden kann, wird ein Zähler hochgesetzt. Der Fitnesswert des Individuums wird schließlich auf den negativen Wert des Zählers gesetzt. Nur gültige Individuen werden überhaupt mit der PoolBean verglichen und können so einen positiven Fitnesswert erreichen.

Fitness-Sharing Mit Hilfe von Fitness-Sharing [1] wird versucht die Diversität der Suchergebnisse bei einer multimodalen Fitnesslandschaft zu erhöhen. Da die Evolutionsstrategie möglichst viele verschiedene Themen für den Benutzer finden soll, ist es hier von Interesse die Diversität möglichst hoch zu halten. Fitness-Sharing basiert dabei auf der Idee Individuen mit einem entsprechenden Abzug auf den Fitnesswert $\phi(x_i)$ zu bestrafen, wenn es zu viele ähnliche Individuen gibt. Zu ähnlich sind sich zwei Individuen genau dann, wenn der Abstand im Suchraum kleiner ist als ein vorgegebener Wert σ_{share}. Umso höher die Anzahl ähnlicher Individuen, desto höher fällt auch die entsprechende Bestrafung aus.

$$\phi'(x_i) = \frac{\phi(x_i)}{\sum_{j=0}^{\mu} sh(x_i, x_j)}$$

$$sh(x_i, x_j) = \begin{cases} 1 - \left(\frac{\|x_i, x_j\|}{\sigma_{share}}\right) & : \text{if } \|x_i, x_j\| \leq \sigma_{share} \\ 0 & : \text{else} \end{cases}$$

Die Distanz zweier Individuen im Suchraum lässt sich hier auf die gleiche Art und Weise, wie bereits in Kapitel 4.2 beschrieben, über das Verhältnis zwischen unterschiedlichen und gleichen fitnessrelevanten Typen ermitteln.

Konfigurierbare Bestrafungsfunktionen Um weitere Bestrafungsfunktionen auf einfache Art und Weise in die Fitnessberechnung einfließen lassen zu können wurde eine weitere Variante implementiert. Diese führt für jedes Individuum den bereits beschriebenen Algorithmus aus und wendet erst dann eine konfigurierbare Liste von Bestrafungsfunktionen darauf an. Diese können den Fitnesswert abhängig von der vorigen Berechung verändern oder auch ganz neu setzen. Die Konfiguration der Fitnessberechnung sowie der einzelnen Bestrafungsfunktionen geschieht via Spring (siehe Algorithmus 4.4).

4.5.2 Bestrafungsfunktionen

Alle Bestrafungsfunktionen implementieren das einfache Interface PenaltyFunctionInterface und werden in der Reihenfolge, in der sie in der Liste (siehe Algorithmus 4.4, ab Zeile 4) eingeordnet sind, auf die einzelnen Individuen angewandt. Für einzelne Bestrafungsfunktionen exisitieren noch spezielle Konfigurationsmöglichkeiten. So kann für die Bestrafungsfunktion, die das Fitness-Sharing umsetzt, der Wert für σ_{share} gesetzt werden.

```
1  <!-- Fitnesswertberechung -->
2  <bean class="de.materna.semanticnav.ea.fitnesscalculator.
      MueRhoLambdaFitnessCalcPenalty" id="de.materna.semanticnav.ea.
      fitnesscalculator.CBNESFitnessCalcPenalty">
3    <property name="penaltyFunction">
4      <list>
5        <ref local="de.materna.semanticnav.ea.fitnesscalculator.
           MueRhoLambdaFitnessSharing"/>
6        <ref local="de.materna.semanticnav.ea.fitnesscalculator.
           MueRhoLambdaDeathPenalty"/>
7        <ref local="de.materna.semanticnav.ea.fitnesscalculator.
           UselessPathPartPenalty"/>
8      </list>
9    </property>
10 </bean>
11
12 <!-- Penalty Functions -->
13 <bean class="de.materna.semanticnav.ea.fitnesscalculator.
      MueRhoLambdaDeathPenalty" id="de.materna.semanticnav.ea.
      fitnesscalculator.MueRhoLambdaDeathPenalty"/>
14 <bean class="de.materna.semanticnav.ea.fitnesscalculator.
      UselessPathPartPenalty" id="de.materna.semanticnav.ea.
      fitnesscalculator.UselessPathPartPenalty"/>
15 <bean class="de.materna.semanticnav.ea.fitnesscalculator.
      MueRhoLambdaFitnessSharing" id="de.materna.semanticnav.ea.
      fitnesscalculator.MueRhoLambdaFitnessSharing">
16   <property name="sigmaShare" value="1.5"/>
17 </bean>
```

Algorithmus 4.4: Konfiguration der Bestrafungsfunktionen in WEB-INF/applicationContext.xml

Die hier einsetzbaren Bestrafungsfunktionen unterscheiden sich in ihrer Arbeitsweise nicht zu den fest implementierten Varianten, die in den einzelnen Modulen zur Fitnessberechung genutzt werden. Allerdings wird hier unabhängig von der Ausgabe der Bestrafungsfunktionen erst ein normaler Fitnesswert berechnet, der dann bearbeitet werden kann. Es ist also zu Gunsten leichterer Konfigurierbarkeit mit einem Performanceverlust zu rechnen, der sich in Tests aber als nicht relevant herausgestellt hat.

Die folgenden Bestrafungsfunktionen wurden als Module implementiert:

DeathPenalty Setzt den Fitnesswert auf die negative Anzahl der Stellen im Pfad, die keine semantische Verknüpfung zulassen. Bei einem diskriminierenden Selektionsverfahren werden diese Individuen meist sofort aus der Population entfernt. (Entspricht Beschreibung in 4.5.1 "Bestrafung nicht semantischer Pfade").

UselessPathPartPenalty Ermittelt alle Instanzen eines Pfades, die keinen Anteil an der Berechung des Fitnesswertes hatten. Je mehr solcher Instanzen enthalten sind, desto höher fällt die Bestrafung aus. Individuen, die unnötige Pfadteile gerade am Anfang oder Ende haben, werden so mit der Zeit aussortiert. Pfadteile, die keine Auswirkung auf die Fitness haben, können sonst beliebige Längen annehmen und so das in der Oberfläche angezeigte Ergebnis stark beeinflussen.

Fitness-Sharing Entspricht der Umsetzung der in 4.5.1 beschriebenen Bestrafungsfunktion Fitness-Sharing, um die Diversität zu erhöhen.

4.6 Rekombinationsverfahren

Die Rekombination, bei der aus ρ zufällig gewählten Eltern ein oder mehrere neue Individuen erschaffen werden, wurde in unterschiedlichen Varianten implementiert. Während die Rekombination einer Evolutionsstrategie eine Richtung geben soll, hat die Mutation die Aufgabe die Streuung der Individuen zu verbessern und so den ganzen Suchraum erreichbar zu machen. Die hier angewandten Rekombinationsoperatoren lehnen sich alle an gängige Crossover-Verfahren an, die aber an die speziellen Eigenschaften des semantischen Webs und den daraus resultierenden Nebenbedingungen mehr oder weniger stark angepasst wurden.

Deterministische Rekombination In dieser einfachen Variante wird jedes der ρ Individuen in wiederum ρ gleich große Teile zerlegt. Das neue Individuum entsteht dann aus einer Konkatenation des jeweils i-ten Teils des jeweils i-ten Individuums. Jedes Individuum bringt damit einen seiner Länge entsprechenden Teil des Pfads mit ein. Das entstehende Individuum entspricht in seiner Länge dem Durchschnitt der ρ Individuen.

Randomisierte Rekombination Aus jedem der ρ Elternindividuen wird ein zufälliges Stück ausgeschnitten und diese Stücke zu einem neuen Individuum zusammengefügt.

Randomisierte Rekombination mit Nachbearbeitung Nachdem das neue Individuum aus zufällig gewählten Teilen der Eltern zusammengesetzt wurde, wird überprüft ob sich Kreise im Pfad befinden. Dies ist immer dann der Fall, wenn die gleiche Instanz im Pfad mehrfach auftaucht. Diese Kreise verbessern die Qualität eines Individuums nicht, allerdings können die Typen der sich auf dem Kreis befindlichen Instanzen, durchaus zu einem besseren Fitnesswert beitragen. Um dies zu verhindern werden in einem Nachbearbeitungschritt alle Instanzen, die auf dem Kreis liegen, aus dem Individuum entfernt.

Typbasiertes Crossover Eine weitere Eigenschaft von Pfaden in einem semantischen Web findet in dieser Variante Berücksichtigung. Die entstehenden Pfade sollen möglichst im semantischen Web existieren. Das heißt, dass zu je zwei aufeinander folgenden Instanzen ein Tripel (Subjekt, Prädikat, Objekt) gefunden werden kann, bei dem die erste Instanz dem Subjekt und die zweite Instanz dem Objekt entspricht. Solche Tripel sind abhängig von den Typen der Instanzen, denn nur zwischen Instanzen mit bestimmten Typen lassen sich entsprechende Prädikate verwenden. Um für den Crossover-Operator also eine geeignete Stelle zu finden wird in beiden Individuen nach Instanzen mit dem gleichen Typ gesucht (siehe Abbildung 4.3).

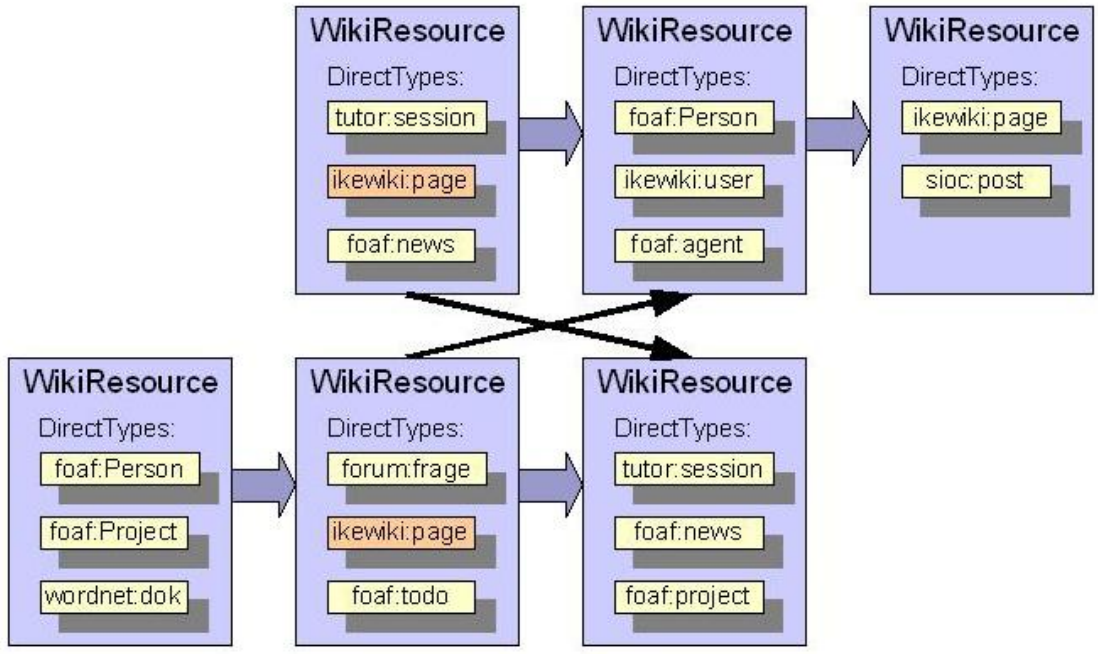

Abbildung 4.3: Typbasierter Crossover-Operator

Eine solche Position lässt sich aber nicht grundsätzlich in den vom Benutzer vergebenen Typen finden. In diesem Fall muss auch in den vom System und durch Vererbung vergebenen Typen gesucht werden. In diesen läßt sich aufgrund des Basistyps "rdfs:Resource" immer mindestens eine solche Position finden. Aus diesen wird dann zufällig eine Position gewählt, hinter der dann ein einfaches Single-Point-Crossover durchgeführt wird. Von dem gleichen Typ ausgehend kann so sichergestellt werden, dass zu den folgenden Instanzen eine semantische Verknüpfung möglich ist.

Auch in diesem Fall werden in einem Nachbearbeitungsschritt alle Kreise aus den neuen Individuen entfernt.

4.7 Mutationsverfahren

Ein guter Mutationsoperator sollte im Wesentlich drei Bedingungen [3] erfüllen:

Erreichbarkeit Ausgegangen von jedem Zustand der Objekt- und Strategievariablen muss jeder andere Zustand durch eine endliche Schrittfolge von Mutationsschritten erreichbar sein.

Keine Verfälschung Die Selektion dient dazu die vielversprechendste Richtung der Suche anhand der Fitnesswerte zu finden, wohingegen die Mutation für eine gute Streuung im Suchraum sorgen soll. Der Fitnesswert sollte bei der Mutation also nicht genutzt werden.

Skalierbarkeit Die Skalierbarkeit kann durch die durchschnittliche Länge eines Mutationsschritts gegeben werden. Man nimmt an, dass kleine Änderungen der Objektvariablen auch kleine Änderungen des Fitnesswerts ergeben. Die Skalierbarkeit ist für kombinatorische Suchprobleme wie in diesem Fall gegeben schwer zu erreichen, da hier minimale Mutationsschritte existieren, die nicht unterschritten werden können.

Für die Mutation eines Pfades in einem semantischen Web kann für jede Position bestimmt werden, ob diese verändert werden soll. Dies geschieht in Abhängigkeit von einer Variable σ die für jedes Individuum gespeichert wird. σ enstpricht dabei der Wahrscheinlichkeit mit der eine Position des Pfads verändert wird. Die Änderung selbst geschieht auf Basis der Typen (siehe Abbildung 4.4) der gewählten Instanz. Dazu werden alle Typen bestimmt und daraus zufällig einer gewählt. Zu diesem Typ werden dann alle verfügbaren zugeordneten Instanzen ermittelt und aus diesen zufällig eine gewählt, die dann die Instanz im Pfad ersetzt. Durch die Berücksichtigung der Typen der geänderten Instanz erhöht sich die Wahrscheinlichkeit ein Individuum zu erzeugen, das bezüglich der semantischen Verknüpfungen zwischen den Instanzen weiterhin die Bedingungen eines gültigen Pfades erfüllt.

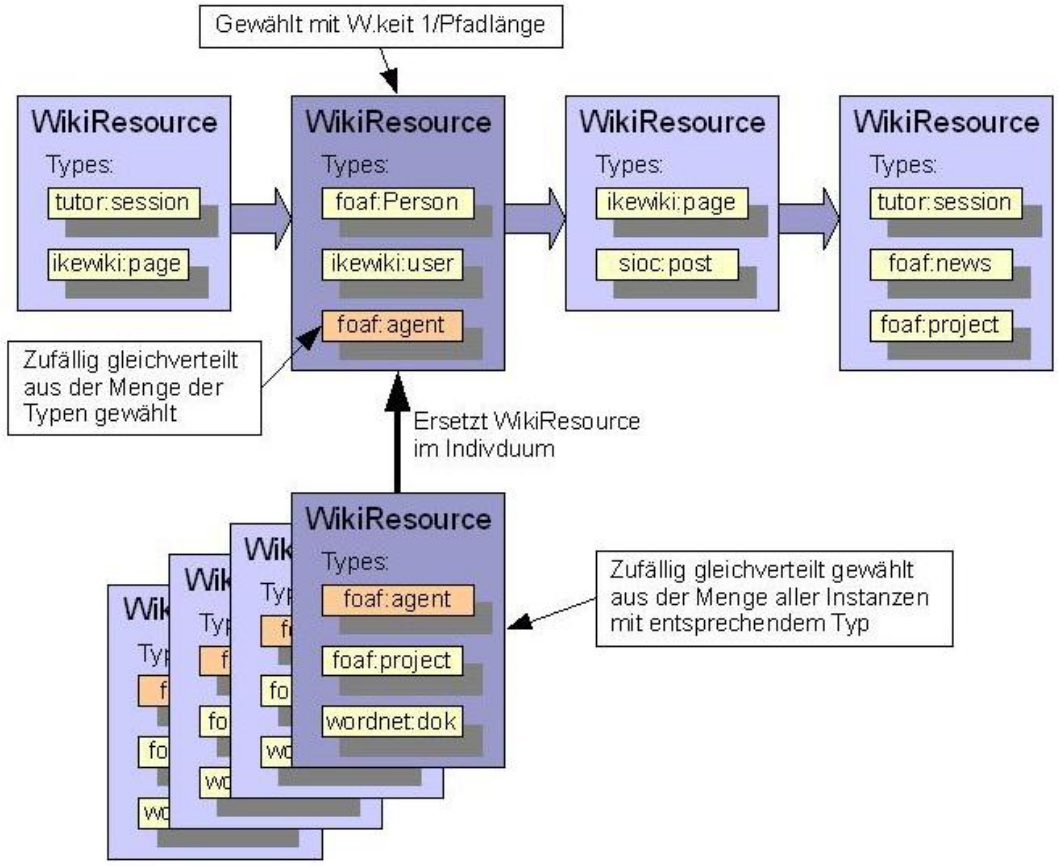

Abbildung 4.4: Typbasierter Mutationsoperator

Trotz der Nutzung der Typen für die Mutation ist die Bedingung der Erreichbarkeit hier erfüllt, da für jede Instanz alle zugeordneten Typen berücksichtigt werden. Wird also der als Wurzel in der Vererbungshierarchie stehende Typ "rdfs:resource" gewählt, entspricht die Menge aller Instanzen, die diesem Typ zugeordnet sind, der Menge aller Instanzen. Durch die Mutation kann in einer entsprechenden Anzahl von Schritten also jedes andere Individuum entstehen.

4.8 Selektionsverfahren

Die zur Verfügung stehenden Selektionsverfahren teilen sich in diskriminierende und nicht diskrimierende. Nur bei den nicht diskriminierenden Verfahren haben Individuen die Möglichkeit sich trotz anderer möglicherweise besser bewerteter Individuen durchzusetzen und in die nächste Generation übernommen zu werden. Als diskriminierend werden die Verfahren bezeichnet, die alleine auf Basis einer Sortierung nach dem aktuellen Fitnesswert die Individuen für die nächste Elterngeneration bestimmen. Abgesehen von der diskrimi-

nierenden Selektion, die normalerweise in einer Evolutionsstrategie genutzt wird, wurden drei weitere nicht diskriminierende Verfahren implementiert.

Selektion auf Basis eines Rankings Um die Individuen zu ermitteln, die in die nächste Generation übernommen werden sollen, wird für jedes Individuum ein Ranking ermittelt (siehe Algorithmus 4.5). Dazu wird für jedes Individuum eine fest konfigurierte Anzahl an Wettkämpfen durchgeführt und die Anzahl der Siege gezählt. Schließlich werden alle Individuen nach dem erstellten Ranking sortiert und die besten μ in die folgende Generation übernommen. Da jedes Individuum nur mit einer bestimmten Anzahl zufällig gewählter Gegner verglichen wird, besteht somit auch für Individuen mit einem schlechten Fitnesswert die Möglichkeit einen höheren Wert im Ranking zu erreichen.

```
1  for (Individual individual : pool) {
2    int victoryCount=0;
3    for (int i=0; i<AnzahlDerWettkaempfe; i++){
4      //zufaellige Auswahl eines Gegners
5      Individual rival = waehleZufaelligGegner(pool);
6      //Sieger ermitteln
7      if (individual.getFitness()>rival.getFitness())
8        victoryCount++;
9    }
10   individual.setRanking(victoryCount);
11 }
12 //Sortierung nach Ranking, absteigend
13 Collections.sort(pool, new Comparator<Individual>());
14 //Rueckgabewert sind die mue besten Individuen nach Ranking
15 return pool.subList(0, mue);
```

Algorithmus 4.5: Selektion mit Ranking

Selektion mit fitnessbasierter Wahrscheinlichkeit (Monte-Carlo-Verfahren) Eine andere Möglichkeit ist die Selektion mit Hilfe einer auf dem Fitnesswert basierenden Wahrscheinlichkeit. Dafür wird eine Art Rouletterad genutzt, in dem für jedes Individuum ein Fach exisistiert, das in der Größe seinem Fitnesswert entspricht. Dieses Rad wird dann μ mal gedreht um die Individuen zu ermitteln, die in die nächste Generation übernommen werden.

Auch bei diesem Verfahren haben Individuen mit einem schlechten Fitnesswert eine Chance übernommen zu werden. Somit eignet sich dieses Verfahren um die genetische Vielfalt aufrecht zu erhalten.

Turnierselektion Ähnlich wie der Selektion über ein Ranking wird bei der Turnierselektion eine vorkonfigurierte Anzahl an Individuen aus der Offspring-Generation gewählt. Von diesen wird das Individuum mit dem höchsten Fitnesswert ermittelt, welches dann direkt in die folgende Elterngeneration übernommen wird. Auch dieses Verfahren ist nicht diskriminierend und kann über die Anzahl der zu vergleichenden Individuen den Selektionsdruck anpassen.

Kapitel 5

Darstellung der Benutzeroberfläche

Die grafische Oberfläche von IkeWiki bietet verschiedene Möglichkeiten Inhalte semantisch einzuordnen und zu verknüpfen. Es existieren eigene Masken für die Pflege und Erstellung der verwendeten Ontologien. Im folgenden Kapitel sollen diese Schnittstellen für den Benutzer zu den semantischen Inhalten sowie die semantische Navigation kurz gezeigt und erläutert werden.

5.1 Pflegemasken für semantische Inhalte

Die Erfassung der Semantik für die Inhalte des Wikis gliedert sich in zwei Bereiche. Zum einen werden die Ontologien erstellt und bearbeitet. Auf der anderen Seite werden diese dann dazu genutzt um die Inhalte des IkeWiki einzuordnen und zu verknüpfen. Die dafür verwendeten grafischen Schnittstellen sind Inhalt der folgenden Unterkapitel.

5.1.1 Bearbeiten und Erstellen von Ontologien

Ontologien lassen sich unter IkeWiki über die Menüpunkte in der rechten Navigationsspalte unter der Überschrift "Edit" erweitern (siehe Abbildung 5.1).

Hier werden dem Administrator die notwendigen Links für das Anlegen und Löschen neuer Typen und Properties angezeigt. Darüber hinaus können hier direkte semantische Verknüpfungen des akuellen Artikels angelegt werden, die nicht durch einen Link im Artikel selbst repräsentiert werden. Die Templates, die über dieses Menü erstellt werden, können bei der Erstellung aller anderen Objekte genutzt werden, da alle Typen und Properties immer auch einen Wikiartikel darstellen. Die Masken für das Anlegen neuer Typen und Properties ermöglichen es dem Benutzer diese in einen Namespace und die Vererbungshierarchie einzuordnen (siehe Abbildung 5.2). Für die Properties lassen sich zusätzlich verschiedene Verknüpfungsarten einstellen, die sich dadurch unterscheiden, ob ein Property zwei Instanzen miteinander verbindet oder ob auf ein Literal verweist.

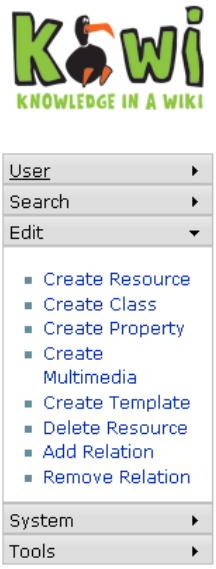

Abbildung 5.1: Bearbeiten der Ontologien

5.1.2 Semantische Verknüpfungen in Dokumenten

Um bereits existierende Artikel im Wiki mit anderen zu verknüpfen bieten sich die bereits vorhandenen Links des Artikels an. Diese lassen sich über den Karteireiter "Annotate" (siehe Abbildung 5.3) mit semantischen Verknüpfungen hinterlegen. Auch die Typen dieser Instanz lassen sich über diese Seite verwalten.

Über die "+" und "-" Buttons kann der Benutzer die entsprechenden Beziehungen auswählen. Dabei wird ihm eine Auswahl geboten, die den entsprechenden Typen der ausgehenden und eingehenden Instanzen entspricht (siehe Abbildung 5.4). Links, die nicht über solche Buttons verfügen, wurden mit Hilfe von SPARQL automatisch generiert. Da diese dynamisch erzeugt werden, können ihnen keine festen Werte zugeordnet werden. In der Regel werden die SPARQL-Statements aber auf bereits existierenden semantischen Beziehungen aufgebaut. So können zu einem Benutzer z.B. alle Artikel angezeigt werden, zu denen er eine "ikewiki:isAuthor"-Beziehung hat. In der oberen Zeile sind alle Typen der aktuellen Instanz aufgelistet. Dabei werden die geerbten Typen ausgegraut. Hier lassen sich

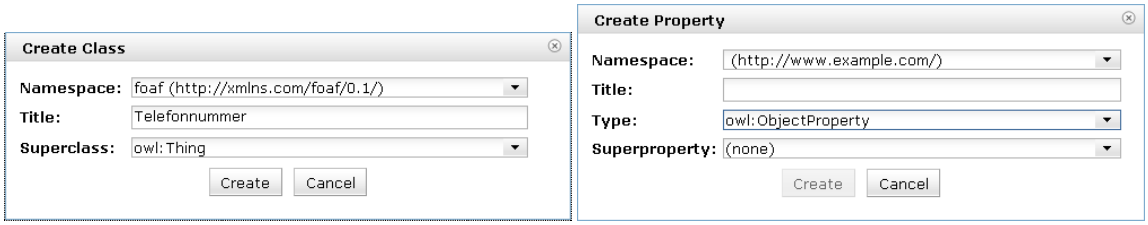

Abbildung 5.2: Anlegen von Typen und Properties

Abbildung 5.3: Bearbeiten der Links einer Instanz

nach dem gleichen Prinzip wie bei den Links auch weitere Typen hinzufügen und wieder entfernen.

Eine weitere Unterstützung des Benutzers durch das System existiert nicht. Jegliche Zuordnungen geschehen ohne eine weitere Überprüfung. Auch beim Anlegen von neuen Typen und Properties bietet das System keine Funktionen, die z. B. unnötige Verdopplungen erkennen und vermeiden könnten. An dieser Stelle wird dem Benutzer also eine hohe Disziplin und ausreichendes Wissen für den Umgang mit Ontologien abverlangt. Darin ist sicherlich ein wesentlicher Punkt zu sehen, warum das semantische Web noch relativ selten anzutreffen ist. Bis jetzt existieren noch keine ausreichenden Assistenzsysteme, die es auch einem ungeübten Benutzer ermöglichen ein solches semantisches Web zu pflegen.

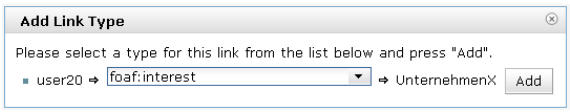

Abbildung 5.4: Auswahl der Properties zu einem Link

5.2 Darstellung der semantischen Navigation

In der rechten Navigationsspalte befindet sich die semantische Navigation. Diese wurde als eigenes Portlet im Sinne des IkeWiki integriert (siehe Abbildung 5.5). Diese zeigt neben den Links die jeweiligen für die Entwicklung interessanten Fitnesswerte, sowie darüber einen prozentualen Füllstand der PoolBean an. Zu jedem Link kann der zugehörige Pfad als Tooltip angezeigt werden.

Die Benutzerschnittstelle ist in dieser Form noch für die Entwicklung des Systems angepasst. Eine besser für die Nutzung optimierte Version sollte für eine bessere Über-

Abbildung 5.5: Portlet mit der semantischen Navigation

sichtlichkeit sowohl auf die Anzeige der Fitnesswerte als auch auf den PoolBean-Füllstand verzichten. Die Links selbst können nach den gefundenen Nischen sortiert präsentiert werden. Ein kurzer Teasertext unter der jeweiligen Überschrift sollte dem Benutzer die Einordung des angebotenen Links erleichtern. Dieser könnte um dem geringen Platz gerecht zu werden dynamisch ausgeklappt werden, wenn der Benutzer zusätzliche Informationen wünscht. Die Anzeige der Pfade ist aufgrund der schmalen Spalte für die Portlets auch nicht permanent möglich, dennoch könnte man mit Hilfe von z.B. Ajax-Komponenten eine optisch ansprechenderere Darstellung erreichen.

5.3 Kommunikation mit dem Algorithmus

Abgesehen von der permanenten Protokollierung der Klicks des Benutzers ist auch ein direktes positives Feedback an den Algorithmus über die Links der semantischen Navigation möglich. Klickt ein Benutzer auf einen Link in der semantischen Navigation so wird nicht nur durch den LinkPathFilter geprüft, ob der direkte Link einem semantischen Pfad entspricht, sondern zusätzlich die ID des entsprechenden Individuums ermittelt. Der Pfad des entsprechenden Individuums wird dann direkt in die PoolBean eingefügt, was zu einer deutlichen Erhöhung des zugehörigen Fitnesswertes führt. Dies entspricht in etwa einer Abkürzung und wird von dem Algorithmus ähnlich interpretiert, als wenn der Benutzer genau diesen zugehörigen Pfad noch einmal manuell durchgeklickt hätte. Da dieses Verhalten schlecht durch einen Webtest abzubilden ist, wurde in den folgenden Versuchen auf die

Nutzung dieser Links verzichtet. Die Eingaben durch die simulierten Benutzer geschehen also immer durch das Anklicken aller Links, die zu einem Pfad gehören.

Um diese Möglichkeit des Benutzerfeedbacks noch zu präzisieren könnten zusätzlich die jeweiligen Instanzen die Teil des Pfads sind als Links zur Verfügung gestellt werden. Auf diese Art und Weise könnten auch Teilpfade vom Benutzer positiv verstärkt werden. Dazu müsste die entsprechende Schnittstelle im LinkPathFilter um die Position des Links im Pfad erweitert werden.

Abgesehen von der Möglichkeit dem Algorithmus positives Feedback zu geben, könnte man auch entsprechende Buttons für die Bestrafung einzelner Individuen ergänzen. Dieser Ansatz wurde allerdings nicht verfolgt, da es für einen Benutzer nur schwer möglich ist zu entscheiden, wann ein angebotener Link wirklich ungeeignet ist. Die reine Repräsentation als Link kann von einem Benutzer im Bezug auf seine Nützlichkeit sicherlich bewertet werden, allerdings ist er nicht in der Lage zu erkennen, ob aus diesem Link innerhalb eines gewissen Zeitraums nicht genau der gewünschte Link entsteht, der vielleicht einem neuen Interesse entspricht. Diese evtl. auftauchenden Zwischenergebnisse einer willkürlichen Bestrafung seitens des Benutzers auszusetzen kann zu ähnlichen Ergebnissen führen wie der Einsatz der "UselessPathPartPenalty" (siehe Kapitel 6.3.4).

Eine weitere mögliche Ergänzung der semantischen Navigation für zusätzliches Benutzerfeedback ist die Anzeige eines Buttons zur Reinitialisierung des Algorithmus. Auch nach der optimierten Einstellung der exogenen Parameter des genutzten Algorithmus ist es aufgrund der zugrundeliegenden Randomisierung immernoch möglich, dass der Algorithmus in eine Sackgasse läuft. Wegen der hier genutzten Fitnesswertberechnung haben alle Pfade der Länge null auch immer den Fitnesswert null. Haben einmal alle Pfade einer Generation diese Länge erreicht, kann der Algorithmus sehr lange brauchen um dieses Plateau in der Fitnesslandschaft wieder zu verlassen. Da in diesem Fall davon ausgegangen werden kann, dass keine Individuen auf einem guten Weg sind schnell ein neues Optimum zu finden, könnte hier nach z. B. drei Iterationen mit ausschließlich Individuen mit Fitnesswert null und Pfadlänge null, ein Button eingeblendet werden, der es dem Benutzer ermöglicht die Initialisierung des Algorithmus erneut anzustoßen. So sollte eine unnötig lange Zeitspanne mit nicht brauchbaren Links in der Navigation vermieden werden.

Kapitel 6

Verhalten des Algorithmus in einer Testumgebung

Um die Qualität und Leistungsfähigkeit evolutionärer Verfahren auf einem semantischen Web beurteilen zu können muss eine Umgebung geschaffen werden, die es ermöglicht gezielt Ereignisse auszulösen und das Verhalten des evolutionären Algorithmus zu beobachten. Sowohl die Schnelligkeit mit der sich die Ausgaben der semantischen Navigation an neue Situationen anpassen, als auch die Vielfältigkeit der Auswahl sind dabei interessante Kriterien. IkeWiki soll dabei möglichst viele der in der Einleitung skizzierten Anwendungsfälle abdecken. Die Tests werden auf ein speziell für diesen Fall erzeugtes Firmenintranet angewendet, das eine entsprechende semantische Struktur bietet. Die Beschreibung der Strukturen des semantischen Webs sowie die geplanten Testszenarien und deren Auswertung werden in den folgenden Kapiteln beschrieben. Weiterhin wird eine Entwicklungsumgebung für Webtests vorgestellt, die für die Simulation von Benutzereingaben genutzt wurde.

6.1 Aufbau der Testumgebung

Um eine Testumgebung für die semantische Navigation zu schaffen wurde ein Firmenintranet auf Basis des IkeWiki aufgebaut, das im wesentlichen einige gängige Wünsche von Kunden an ein Intranet abdecken soll. Dazu gehören unter anderem Anforderungen wie die Abbildung der Firmenhierarchie und -struktur oder die Möglichkeit Aufgaben einzelnen Mitarbeitern zuordnen zu können. Die semantische Struktur der Testumgebung wurde unter Berücksichtigung dieser Wünsche im Wesentlichen aus den Ontologien des IkeWiki, sowie leicht erweiterten Versionen von FOAF (Friend Of A Friend) und SIOC (Semantically-Interlinked Online Communities) erstellt.

Es wurden insgesamt 22 Benutzer (siehe Abbildung 6.1) mit jeweils eigenen Benutzerseiten angelegt. Auf diesen Seiten befinden sich die Angaben über die Einordnung des

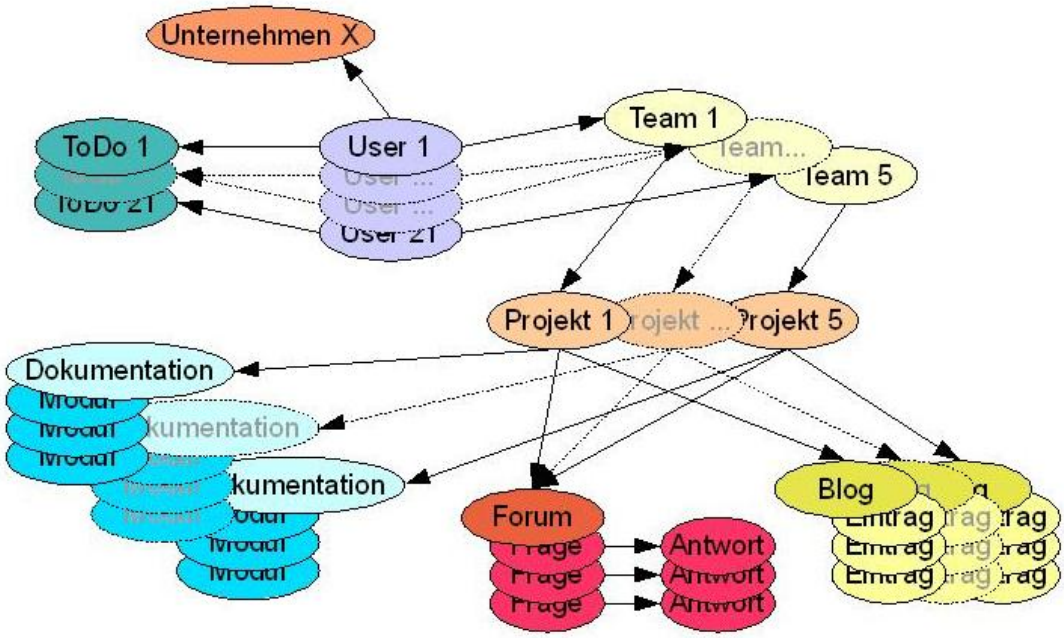

Abbildung 6.1: Grobe Übersicht über die semantische Struktur des Testintranets

Benutzers in die Unternehmenshierarchie und -struktur sowie dessen persönliche Interessen und Aufgaben. Die Benutzer teilen sich auf fünf Teams auf, die wiederum acht verschiedene Projekte bearbeiten. Jeweils drei Benutzer bearbeiten dabei Aufgaben, die dem selben Projekt zugeordnet sind. Die Verwaltung der Aufgaben von Mitarbeitern und Projekten wird auch über die Seiten des Wikis geregelt. Über einen Templatemechanismus lassen sich Instanzen vom Typ "foaf:ToDo" erzeugen, die sich dann einzelnen Benutzern oder Projekten zuordnen lassen. Zu jedem Projekt oder jeder Aufgabe lässt sich weiterhin der jeweilige Fortschritt erfassen. Für die Testumgebung wurde zu jedem Benutzer mindestens eine Aufgabe mit einem entsprechenden Fortschritt angelegt. Für jeden Benutzer besteht außerdem die Möglichkeit seine Position in der Firmenhierarchie durch entsprechende Verknüpfungen zu anderen Benutzern darzustellen. Hierzu wurde zwischen den Objekten vom Typ "foaf:Person" eine "foaf:isChiefOf"-Beziehung erstellt. Diese kann auch auf ganze Teams angewandt werden, da jedes Team durch Vererbung der Typen gleichzeitig auch eine Person darstellt. So wurde zu jedem Team ein Vorgesetzter in der Testumgebung eingerichtet.

Für die einzelnen Projekte existieren verschiedene Möglichkeiten Daten und Wissen zu sammeln und zu strukturieren. Zum einen lassen sich klassische Dokumentationen aus einzelnen Modulen aufbauen. Diese Struktur wird durch die von IkeWiki mitgelieferte Ontologie "tutorial" abgedeckt, die unter Anderem auch für das Hilfe-System genutzt wird. Durch die Ontologie SIOC lassen sich sowohl Foren als auch Blogs im Wiki nachstellen.

Einzelne Fragen und Antworten lassen sich hiermit zu Threads gruppieren und einzelnen Benutzern zuordnen (siehe Abbildung 6.2).

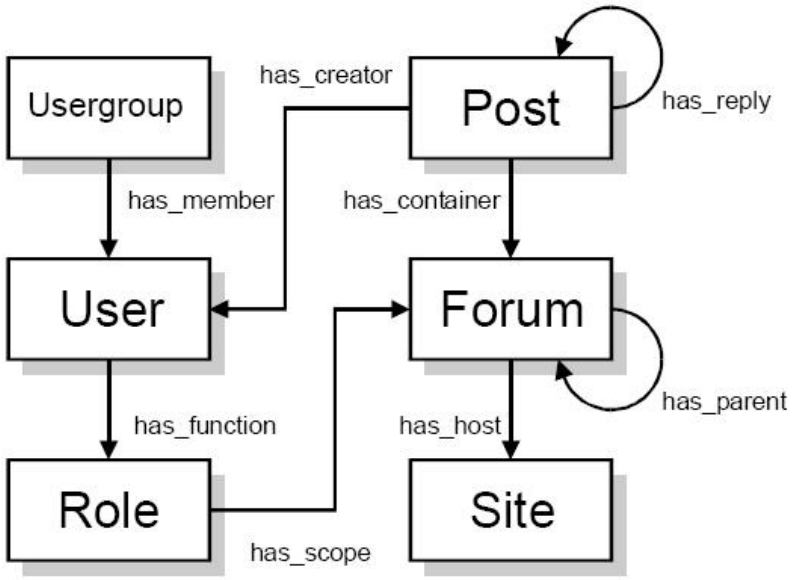

Abbildung 6.2: Grober Aufbau von SIOC, entnommen aus
http://sioc-project.org/files/3_a_sioc_developers_guide.pdf

Zu jedem Projekt wurde für diese Testumgebung mindestens eine solche Struktur angelegt und mit Dummy-Inhalten gefüllt. Speziell an dieser Stelle ergibt sich für die einzelnen Benutzer und Teams ein hohes Maß an Autonomie bei der Verwaltung und Erfassung der eigenen Daten. Gerade durch diese Freiheiten wird einem Benutzer die Suche nach Informationen in fremden Strukturen erschwert. Da die zugrundeliegende Semantik aber gleich oder wenigstens ähnlich sein sollte, wird hier ein hoher Nutzen der semantischen Navigation erwartet.

Weiterhin werden die Interessen eines Benutzers in Form von Verweisen von der Benutzerinstanz auf die jeweilige Instanz von Interesse erfasst. Diese Links können dem Benutzer als eine Art wikispezifische Bookmarks dienen. Auf der Startseite des Wikis wird zusätzlich eine Liste aller News automatisch generiert. Diese Liste enthält alle Instanzen vom Typ "foaf:News".

6.2 Testsoftware Selenium

Um Benutzereingaben automatisiert simulieren zu können wurde Selenium (http://selenium-ide.seleniumhq.org/) eingesetzt. Der ursprüngliche Einsatzzweck dieses Werkzeuges ist das Durchführen automatisierter Webtests. Da sich Selenium aber auch gut für Eingaben auf ajaxbasierten Oberflächen eignet und die Tests einen modularen Aufbau unterstützen, las-

sen sich hierdurch auch gut Benutzereingaben im Sinne von Lasttests simulieren. Selenium steht als Plugin für den Browser Firefox zur Verfügung (siehe Abbildung 6.3) und kann Eingaben in dem Browser aufzeichnen und wieder abspielen. Die aufgezeichneten Tests werden in Form von xhtml-basierten Dateien (siehe Algorithmus 6.1) abgelegt.

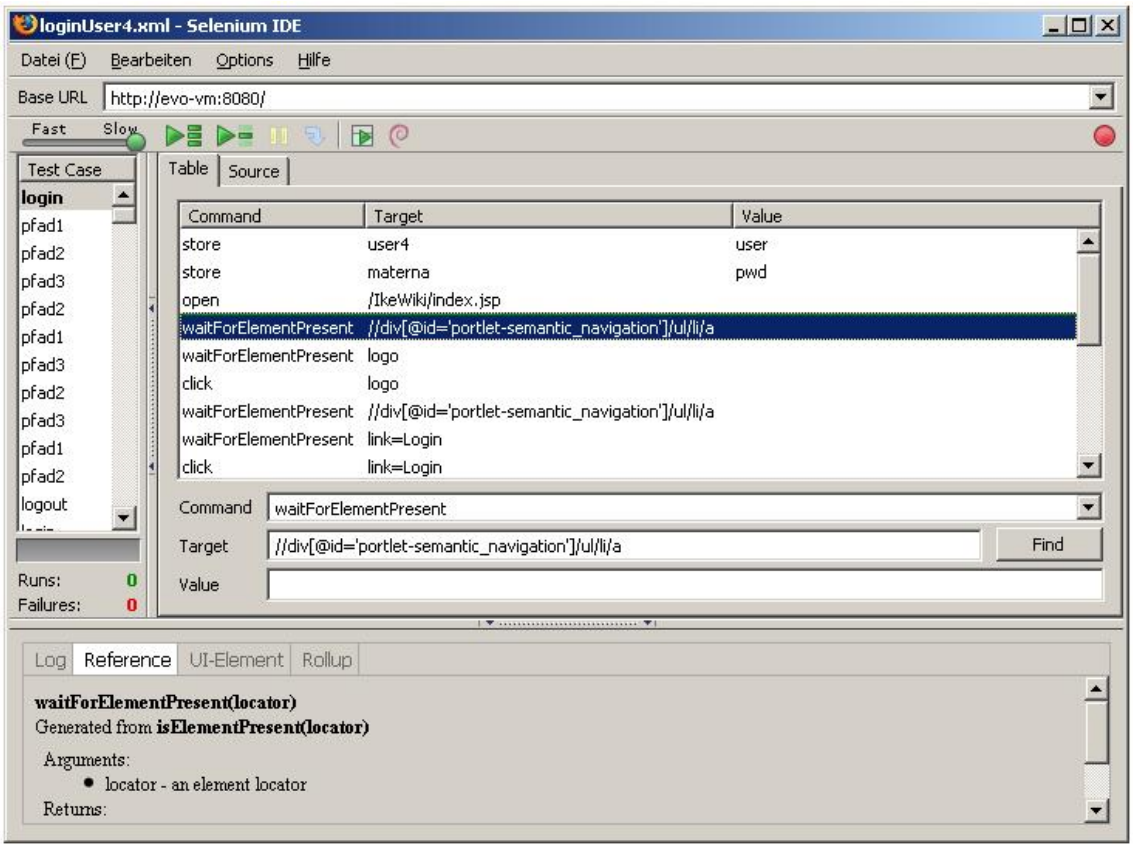

Abbildung 6.3: Selenium IDE als Firefox-Plugin

Selenium referenziert die HTML-Element wahlweise über den Namen, die ID oder eine XPath-Angabe und kann so auch dynamisch via Javascript generierte Elemente in den Tests erfassen und nutzen. Diese Tests lassen sich einzeln speichern und in so genannten Test-Suites zusammenfassen. Für die Benutzersimulation, die für die semantische Navigation durchgeführt werden soll, werden einzelne Pfade des semantischen Webs als Tests erfasst und in verschiedenen Kombinationen zu Test-Suites zusammengefasst, um z.B. das Benutzerverhalten über einen ganzen Arbeitstag zu simulieren.

6.3 Darstellung und Auswertung der Testszenarien

Mit Hilfe der Testszenarien sollen in einem ersten Schritt gute Werte für eine Konfiguration der Evolutionsstrategie gefunden werden. Aufbauend auf etablierten Werten werden dafür einige Wertebereiche der Einstellungsmöglichkeiten systematisch abgesucht und miteinan-

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD␣XHTML␣1.0␣Strict//EN" "http://www.w3
       .org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4  <head profile="http://selenium-ide.openqa.org/profiles/test-case">
5  <meta http-equiv="Content-Type" content="text/html;␣charset=UTF-8" />
6  <link rel="selenium.base" href="" />
7  <title>login</title>
8  </head>
9  <body>
10 <table cellpadding="1" cellspacing="1" border="1">
11 <thead>
12 <tr><td rowspan="1" colspan="3">login</td></tr>
13 </thead><tbody>
14 <tr>
15   <td>open</td>
16   <td>/IkeWiki/index.jsp</td>
17   <td></td>
18 </tr>
19 <tr>
20   <td>click</td>
21   <td>link=Login</td>
22   <td></td>
23 </tr>
24 <tr>
25   <td>type</td>
26   <td>login-password</td>
27   <td>geheim</td>
28 </tr>
29 <tr>
30   <td>click</td>
31   <td>form-login-submit</td>
32   <td></td>
33 </tr>
34 </tbody></table>
35 </body>
36 </html>
```

Algorithmus 6.1: Aufbau eines Selenium-Tests

der verglichen. In einem zweiten Schritt werden dann einzelne Module der Algorithmen ausgetauscht und die Auswirkungen auf die Testergebnisse erfasst, um deren Vor- und Nachteile zu dokumentieren. Aufbauend auf diesen Ergebnissen soll die Diversität durch den Einsatz eines Niching-Verfahrens gesteigert werden. Die Parameter des Niching-Verfahrens sollen entsprechend auf die Anwendung in einem semantischen Web abgestimmt werden. Schließlich soll die gefundene Konfiguration des Algorithmus bei verschiedenen Szenarien dem Benutzer eine Auswahl von Links anbieten, die sich seinen persönlichen Interessen automatisch anpassen.

6.3.1 Simulation von Benutzereingaben

Um verschiedene Benutzer mit unterschiedlichen Interessen mit Hilfe von Selenium simulieren zu können, wurden verschieden Tests erstellt, die die einzelnen Interessen eines Benutzers widerspiegeln. Dafür wurden insgesamt fünf Themengebiete ausgewählt.

Interesse	Anzahl Pfade	Anzahl Klicks	Beschreibung
Forum	4	14	Der Benutzer betrachtet mehrere Fragen und Antworten aus Foren verschiedener Projekte
TODO	3	11	Aufgaben von verschiedenen Benutzern und Projekten werden aufgerufen, bzw. die Benutzer ermittelt, die zu einer Aufgabe gehören.
Dokumentation	4	22	In der Dokumentation verschiedener Projekte wird mehrfach in den Kapiteln hin und her gesprungen.
Blog	3	15	Die Blogeinträge verschiedener Projekte werden durchgesehen.
pers. Interessen	5	16	Die einem Benutzer als persönliches Interesse zugeordneten Links auf der jeweiligen Benutzerseite werden verfolgt.

Um jetzt einen Benutzer über einen längeren Zeitraum simulieren zu können, werden diese Tests in beliebiger Reihenfolge auch mehrfach zu einer Testsuite zusammengesetzt, die dann mit Hilfe von Selenium abgearbeitet werden kann.

6.3.2 Einstellung der exogenen Parameter

Da sich im Rahmen dieser Diplomarbeit ein sehr neues Feld für die Anwendung von Evolutionsstrategien erschließt, empfiehlt es sich die sonst genutzten Einstellungen der exogenen Parameter μ, ρ und λ zu überprüfen. Im Umfeld eines semantischen Webs ergeben sich andere Anforderungen an das Verhalten des Algorithmus, sodass Änderungen der exogenen Parameter erforderlich sein können. Da in einer Web-2.0-Anwendung Benutzer eine gewisse Reaktionsgeschwindigkeit auf ihre Eingaben erwarten und die Fitnessberechung für die semantische Navigation eine sehr hohe Laufzeit aufweist, können die Parameter nur mit relativ niedrigen Werten getestet werden. Für die Versuche wurden alle Kombinationsmöglichkeiten aus folgenden Einstellungen getestet.

μ	ρ	λ
10	2	10
15	3	25
50	5	100
		200

Bei höheren Einstellungen würden die Wartezeiten nach einem Klick für den Benutzer umzumutbar, sobald mehrere Nutzer auf dem System gleichzeitig aktiv sind. Die Fitnesswerte der jeweiligen Durchläufe wurden für einen Benutzer erfasst, der die Pfade der in Kapitel 6.3.1 dargestellten Interessen, jeweils der Reihe nach zweimal abarbeitet. Auf diese Art und Weise werden insgesamt 197 Generationsschritte berechnet. Um das Verhalten auch bei komplett gefüllter PoolBean beobachten zu können, wurde diese auf einen relativ geringen Wert von nur 15 Pfaden eingestellt. Damit ist die PoolBean ab der 79ten Generation komplett gefüllt. Danach auftretende Schwankungen des Fitnesswerts sollten also hauptsächlich auf die sich wandelnden Interessen des Nutzers zurückzuführen sein.

Für jede Kombinationsmöglichkeit wurden mehrere Durchläufe mit jeweils geändertem Randomseed durchgeführt, die für die Auswertung gemittelt wurden. Bei der Auswertung der Ergebnisse wurden drei wesentliche Kriterien betrachtet:

Maximal erreichter Fitnesswert Mit welcher Kombination von μ, ρ und λ konnte im Durchschnitt der höchste Fitnesswert erreicht werden.

Laufzeit Umso höher die Werte von μ, ρ und λ sind, desto länger benötigt der Algorithmus um eine Generation zu berechnen. Dabei sollte der Nutzen der höheren Werte im Verhältnis zu ihrer Laufzeit stehen. Um eine Lauffähigkeit und Nutzbarkeit auf aktueller Hardware auch bei einer höheren Zahl von Nutzern zu gewährleisten, wurden die Werte entsprechend der obigen Tabelle gewählt.

Reaktionsgeschwindigkeit Bei einem Wechsel der angeklickten Themen des Benutzers sollten keine starken Verzögerungen bei der Reaktion seitens des Algorithmus auftre-

ten. Nach möglichst kurzer Zeit sollten die neuen Themen entsprechend berücksichtigt werden.

Für die Werte 10 und 15 für μ sowie die Werte 10 und 25 für λ wurden viele Läufe beobachtet in denen der Algorithmus während der ersten 200 Generationen keine Lösungen finden konnte, die einen Fitnesswert größer null hatten. Dadurch ergibt sich hier im Mittel ein sehr schlechtes Bild für die durchschnittliche maximal erreichte Fitness (siehe Abbildung 6.4).

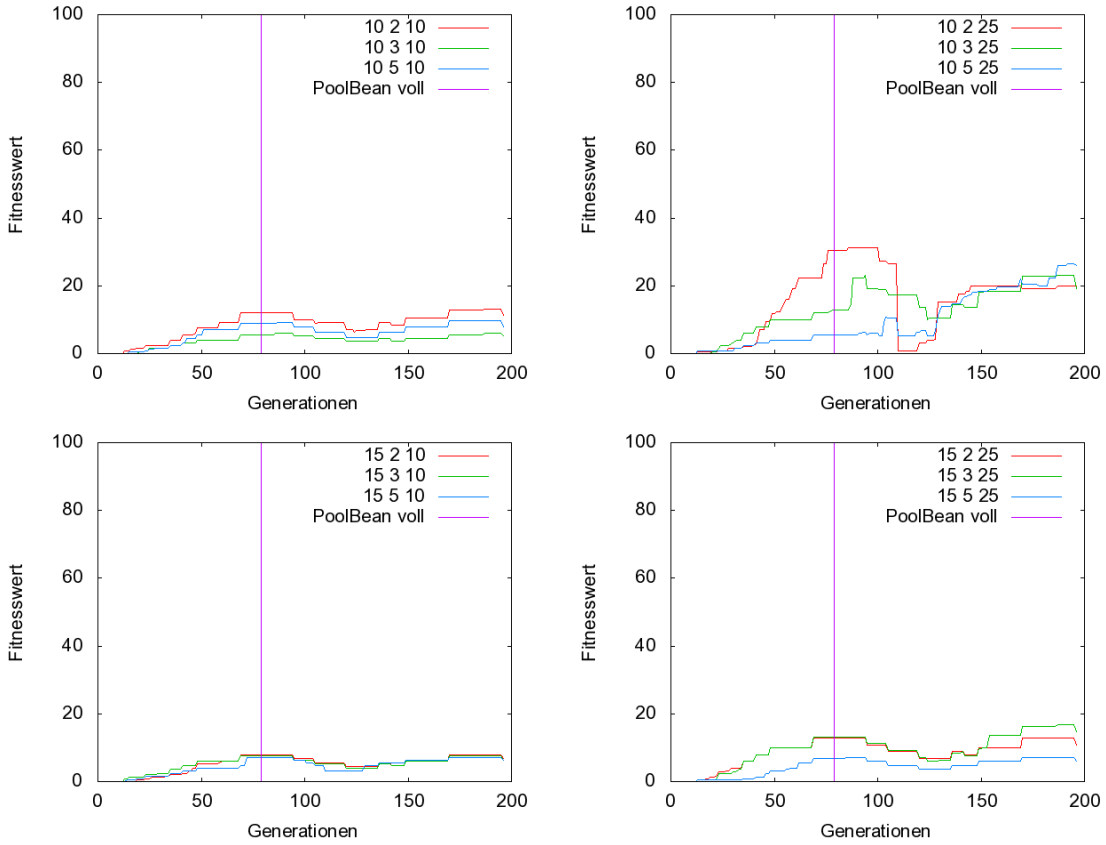

Abbildung 6.4: Niedrige Fitnesswerte bei niedrigen Werten für μ und λ

Bereits hier lässt sich aber gut erkennen, dass bis zur Generation 79 die Fitnesswerte der einzelnen Varianten stetig ansteigen. Sobald ab dieser Generation die PoolBean komplett gefüllt ist, ergibt sich ein langsames Schwanken des Fitnesswerts. Dieser bricht immer dann ein, wenn Pfade wieder aus der PoolBean entfernt werden, die zu einem Themengebiet gehören, welches als lokales Optimum vom Algorithmus erreicht wurde. Zu diesem Zeitpunkt kommt es teilweise sogar zu kompletten Zusammenbrüchen, die einem Neustart des Algorithmus gleich kommen. Dabei kehren alle Individuen auf den Fitnesswert null zurück bevor ein neues lokales Optimum gefunden werden kann. Diese Werte liegen also eindeutig zu niedrig um einen hohen Fitnesswert und eine gute Reaktionsgeschwindigkeit zu gewähr-

leisten. Außerdem ist zu beobachten, dass der Wert für ρ hier keinen nennenswerten Einfluß auf die Ergebnisse hat.

Die starke Stufenbildung im Verlauf der Fitness ist durch die Funktionsweise der Fitnessberechung zu erklären. Da bei jedem Lauf für eine Kombination der exogenen Parameter durch den Seleniumtest exakt die selben Eingaben simuliert wurden, wird in der PoolBean auch immer zum gleichen Zeitpunkt ein neuer Pfad ergänzt und ein alter Pfad entfernt. Dies führt bei allen Läufen jeweils an der selben Stelle zu einem sprunghaften Anstieg oder Abfall des Fitnessverlaufs, was sich auch in den durchschnittlichen Werten durch gut sichtbare Stufen bemerkbar macht.

Der höchste durchschnittliche Fitnesswert wurde mit den Werten $\mu = 50$, $\rho = 3$ und $\lambda = 200$ erreicht (siehe Abbildung 6.5). Hier sieht man ein deutlich unterscheidbares Verhalten in Abhängigkeit von ρ. Ein Wert von $\rho = 5$ führt hier zu einem etwas langsameren Anstieg der Fitness, dafür sind allerdings im mittleren Bereich und am Ende geringere Einbrüche der Fitness bei gefüllter PoolBean zu beobachten. Für den Benutzer ergibt sich so ein konstanteres Bild seiner Links, das nicht durch Phasen unterbrochen wird, in denen die Qualität deutlich absinkt.

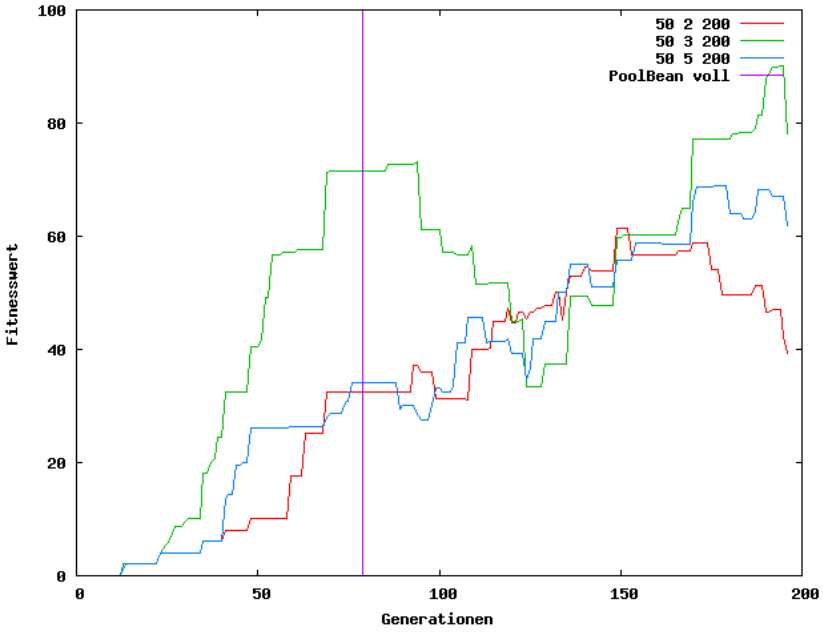

Abbildung 6.5: Durchschnitt maximal erreichter Fitnesswerte

Um die Wichtigkeit eines maximalen Fitnesswertes beurteilen zu können muss dessen Aussage genau betrachtet werden. Umso höher der Fitnesswert ist, desto mehr Pfade existieren in der PoolBean, deren Typen dem Pfad des Individuums entsprechen. Ein maximaler Fitnesswert wäre also durch eine Gleichheit der Typen zu erreichen. Dadurch würden dem Benutzer aufgrund der Typbasierung Ergebnisse angezeigt, die nicht komplett den von ihm gegangenen Pfaden entsprechen, deren Ähnlichkeit aber dennoch extrem hoch wäre.

Eine etwas stärkere Abweichung kann also durchaus im Interesse des Benutzers sein. Eine Möglichkeit auf Kosten der maximalen Fitness eine bessere Laufzeit zu erreichen bietet die Reduzierung des Wertes von λ von 200 auf 100 (siehe Abbildung 6.6).

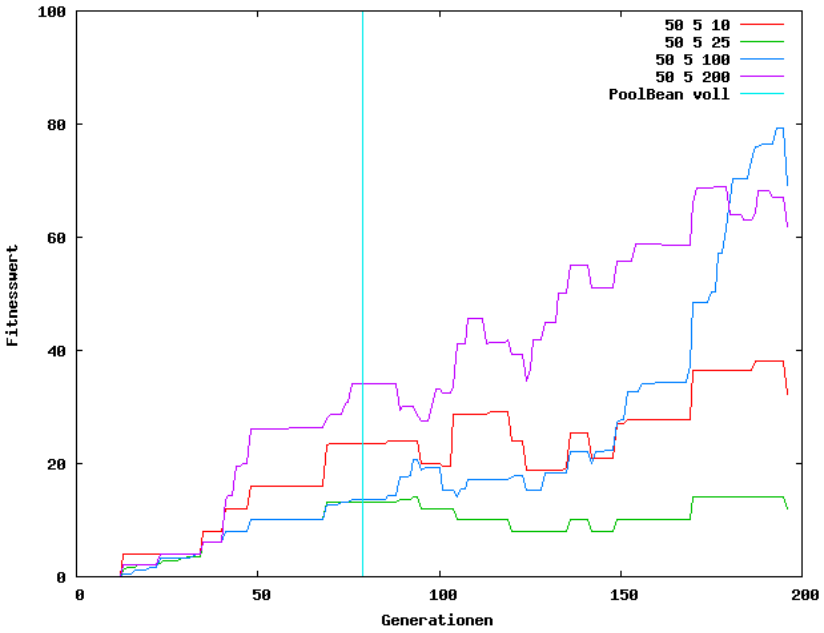

Abbildung 6.6: Reaktionsgeschwindigkeiten für unterschiedliche λ

Die für diese Einstellung zu beobachtenden Einbrüche werden innerhalb weniger Generationen wieder behoben. Trotz des in weiten Teilen etwas niedrigeren Fitnesswertes kann der Algorithmus hier schnell auf kurzfristige Änderungen im Benutzerverhalten reagieren. Durch die beträchtliche Reduzierung der zu berechnenden Individuen kann die Laufzeit so verbessert werden, dass sie auch auf sehr stark belasteten Systemen zu verantworten ist.

Bei hohen Werten von μ und λ zeigt sich die Wirkungsweise von ρ wesentlich deutlicher (siehe Abbildung 6.7).

Für niedrige Werte von ρ lassen sich stärkere Schwankungen der Fitness durch die dynamische Fitnesslandschaft erkennen. Eine stärkere Durchmischung der Individuen führt dazu, dass die Schwankungen der Fitnesslandschaft besser ausgeglichen werden. Während der Rekombination entstehen so häufiger Individuen, die in neue Bereiche des Suchraums vordringen. Diese unterstützen die Anpassung an die veränderte Fitnesslandschaft. Dass dieser Effekt erst bei hohen Werten von μ und λ auftritt, liegt im wesentlichen an der diskriminierenden Selektion, die solche stark veränderten neuen Individuen aufgrund anfänglich niedriger Fitnesswerte schnell aussortiert.

Um den oben genannten Anforderungen gerecht zu werden kann hier also davon ausgegangen werden, dass die Werte am besten auf $\mu = 50$, $\rho = 5$ und $\lambda = 100$ bei stark ausgelasteten Systemen und $\lambda = 200$ bei weniger ausgelasteten Systemen gesetzt werden

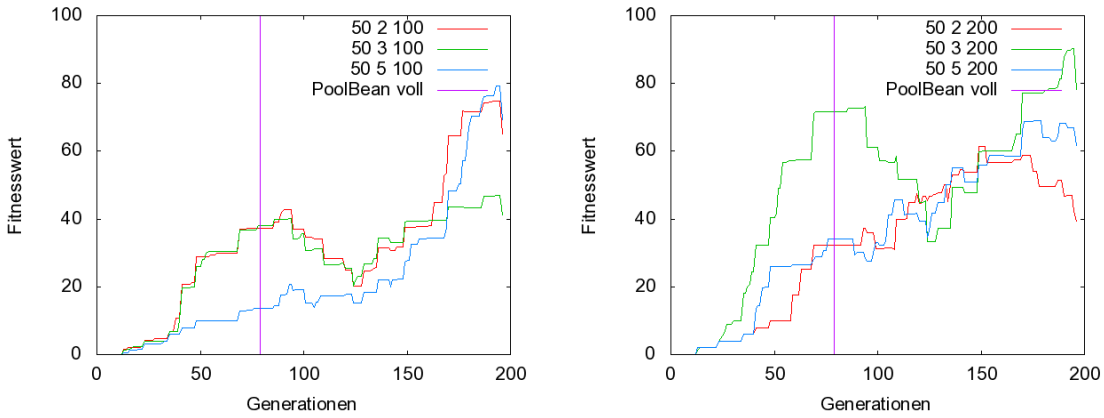

Abbildung 6.7: Auswirkung von ρ bei hohen Werten für μ und λ

sollten. Im folgenden wird hier von einem System ausgegangen, das mit der Einstellung $\lambda = 200$ gut zurecht kommt.

6.3.3 Größe der PoolBean

Die Schnelligkeit mit der eine Evolutionsstrategie mit einer dynamischen Fitnesslandschaft zurecht kommen muss hängt stark von der Größe der PoolBean ab. Die PoolBean enthält für jeden Benutzer eine Liste fester Länge, welche die Pfade speichert, die vom Benutzer im Web genutzt wurden. Sobald die PoolBean ihre maximale Auslastung erreicht hat verdrängt ein neuer Pfad den jeweils Ältesten. Immer wenn ein neuer Pfad aufgenommen und gegebenenfalls ein alter Pfad entfernt wird, kommt es somit zu einem Sprung in der Fitnesslandschaft. Diese Sprünge sind bei einer geringen Anzahl von Pfaden in der PoolBean wesentlich deutlicher zu merken, da dann der einzelne Pfad einen höheren Anteil an der Fitness eines Individuums hat. Je höher die Anzahl der Pfade in der PoolBean desto höher wird auch der maximal erreichbare Fitnesswert (siehe Abbildung 6.8).

Bei einer Poolgröße von zehn sind die Schwankungen der Fitnesslandschaft noch zu stark. Der Verlauf der Fitness bricht hier deutlich ein, da die PoolBean zu schnell ihren Inhalt wechselt. Ab einer Anzahl von 15 Pfaden ist hier deutlich zu sehen, dass die gewählte Einstellung des Algorithmus ohne starke Einbrüche auf die Themenwechsel des Benutzers reagiert. Für den hier simulierten Benutzer ist eine Poolgröße von über 25 bereits zu groß. Am Ende der Fitnesskurve geht der Fitnesswert leicht zurück. Dies ist darauf zurück zu führen, dass der Benutzer die Interessen "Forum", "TODO", "Dokumentation", "Blog" und "pers. Interessen" zweimal nacheinander durchgeht. Die PoolBean ist jetzt aber so groß, dass z. B. das Thema "Forum" noch nicht wieder ganz vergessen wurde bevor es wieder neu hinzugefügt wird. Der Algorithmus verlässt das lokale Optimum für das Thema "Forum" also gar nicht erst sondern wartet den kurzfristigen Einbruch der Fitness, der für dieses spezielle Thema relativ gering ausfällt, einfach ab. Für das hier angenommene

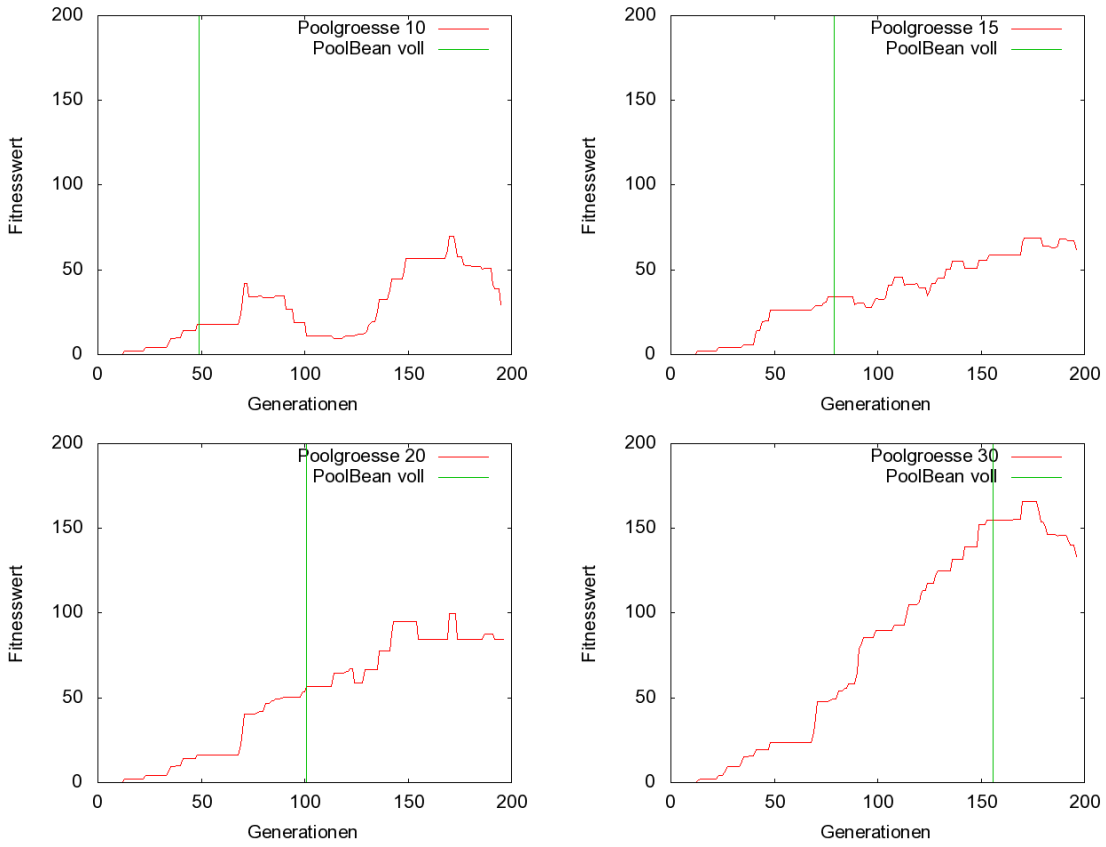

Abbildung 6.8: Verlauf der Fitness in Abhängigkeit von der Größe der PoolBean

Benutzerverhalten ist eine Poolgröße zwischen 15 und 20 also als ideal anzusehen. Um in einer Umgebung mit echten Benutzern ein optimales Verhalten zu erreichen sollte deren Klickverhalten analysiert und die Poolgröße dementsprechend angepasst werden. Da die Größe der PoolBean sich auch direkt auf die Laufzeit der Fitnessberechung auswirkt wird für die weiteren Tests der Wert auf 15 festgelegt.

6.3.4 Variation der Module

Im Folgenden soll die Wirkungsweise und der resultierende Nutzen der einzelnen Module getestet werden. Dafür wird für jedes Modul ein entsprechender Test durchgeführt, der die evtl. vorhandenen zusätzlichen Einstellungen variiert.

Initialisierung durch Benutzereingaben

Als Alternative zu der Initialisierung mit Hilfe zufälliger Pfade auf Basis des Benutzerobjekts wurde ein Verfahren implementiert, das direkt die vom Benutzer gegangenen Pfade aus der PoolBean für die neuen Individuen nutzt. Dabei wird gewartet bis durch Benutzereingaben genug Pfade gesammelt wurden um diese dann zu den μ ersten Individuen zu

machen. Voraussetztung dafür ist, dass die PoolBean mindestens μ Individuen pro Benutzer aufnehmen kann. Dies macht es schwierig mit den vorher ermittelten optimalen Werten zu arbeiten, da eine PoolBean mit der Größe 50 starken Einfluß auf die Laufzeit ausüben würde. Um zu demonstrieren wie sich der Algorithmus bei dieser Art der Initialisierung verhält, reicht es allerdings einen Wert von 15 Individuen zu betrachten. Das hier auftretende Verhalten wird durch höhere Werte für die PoolBean durch die daraus resultierenden höheren maximalen Fitnesswerte höchstens noch extremer.

Das gewünschte Verhalten, das von einer solchen Initialisierung erwartet würde, wäre es dem Benutzer eine Startphase zu ersparen, in der die ihm angebotenen Links der semantischen Navigation stark randomisiert und damit willkürlich erscheinen. Anstelle dieser Startphase würde dem Benutzer eine Zeit lang gar keine Auswahl an Links angeboten bis genug Daten gesammelt sind, um direkt einen Einstieg mit einer hohen Qualität zu bieten.

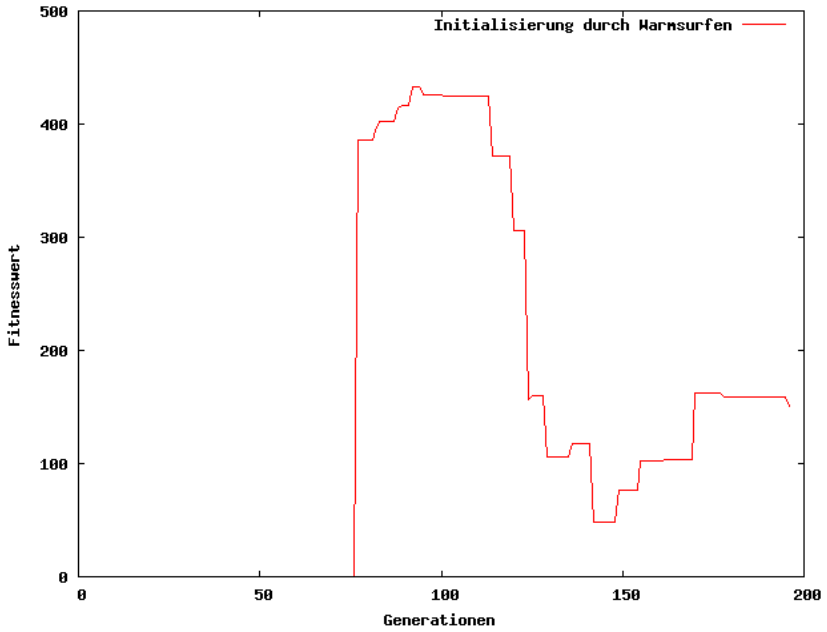

Abbildung 6.9: Initialisierung durch Benutzereingaben

Wie in Abbildung 6.9 zu sehen wird, sobald die PoolBean komplett gefüllt ist, die Initialisierung des Algorithmus vorgenommen. Durch die Funktionsweise der Fitnesswertberechnung kommen in der ersten Generation also nur Individuen mit extrem hohen Fitnesswerten vor, da es zu jedem Individuum in der Startpopulation mindestens eine exakte Entsprechung in der PoolBean gibt. Jedes Individuum stellt in diesem Moment also fast ein lokales Optimum dar. Dabei sind die Ergebnisse, die dem Benutzer präsentiert werden, nicht unbedingt in dem Sinne nützlich für den die semantische Navigation eigentlich gedacht sein soll. Die hier präsentierten Ergebisse entsprechen mehr einer Liste priorisierter Bookmarks und nicht den eigentlich gesuchten alternativen Artikeln mit ähnlichen Themengebieten.

Bei beiden möglichen Initialisierungen ist also das Bild, das sich dem Benutzer zu Beginn zeigt, nicht den eigentlichen Erwartungen entsprechend.

Der weitere Verlauf der Fitnesskurve zeigt allerdings, dass der Algorithmus mit dieser Initialisierung nicht gut zurechtkommt. Schon nach wenigen Iterationen bricht der durchschnittliche Fitnesswert stark ein. Einige Durchläufe fielen dabei sogar auf einen Wert von null. Danach ist eine Erholungsphase zu beobachten, bei der sich der Algorithmus auf etwa dem gleichen Niveau einpegelt, das er auch mit einer randomisierten Initialisierung nach dieser Zeit erreichen kann. Für das spätere Verhalten des Algorithmus bietet also keines der beiden Verfahren einen besonderen Vorteil. Die Frage, die bei einer Entscheidung für eines der beiden Systeme beantwortet werden muss, ist also vielmehr die, durch welche Variante eine höhere Benutzerakzeptanz erreicht werden kann. Die letztendliche Leistungsfähigkeit über einen längeren Zeitraum unterscheidet sich für die beiden Varianten nicht.

Um das Verhalten der Initialisierung durch Benutzereingaben zu verbessern sind mehrere Ansatzpunkte denkbar. Zum Ersten wäre die Voraussetzung, dass die PoolBean mindestens die Größe von μ haben muss, nicht unbedingt notwenig, wenn die PoolBean bis zur Initialisierung eine größere Menge Pfade sammeln könnte, als sie hinterher zur Bewertung für die Fitnessberechnung zur Verfügung stellt. Dadurch würden weiterhin relativ hohe Fitnesswerte für die erste Generation entstehen, die aber im Vergleich zu der jetzigen Implementierung etwas geringer ausfallen würden. Außerdem kann der Algorithmus durch die größere Generation vermutlich besser auf die Schwankungen reagieren. Der Einbruch der Fitness sollte weniger extrem ausfallen.

Einen weiteren Ansatzpunkt liefert die Art wie die Individuen der ersten Generation aus den Pfaden der PoolBean gewonnen werden. Ein Verfahren, das Beispielsweise nur Teilpfade auswählt oder auf anderen Wegen die Individuen generiert, könnte die anfänglichen Maximalwerte etwas reduzieren. Durch weitere Verbesserungen besteht hier also die Möglichkeit die Initialisierung durch Benutzereingaben zu einer sinnvollen Komponente zu machen.

Rekombination mit typbasiertem Crossover

Der Rekombinationsoperator für die Suche in einem semantischen Web soll neue Individuen erschaffen, die einer gewissen Richtung im Suchraum folgen. Eine solche Richtung kann im semantischen Web nur schwer ausgemacht werden. Die Idee für den typbasierten Crossover-Operator kombiniert ein normales Single-Crossover mit einer auf das semantische Web angepassten Bestimmung der Crossover-Position. Da gute Individuen hauptsächlich auf gültigen Pfaden vermutet werden, soll es dem Algorithmus hier ermöglicht werden mit erhöhter Wahrscheinlichkeit solche gültigen Individuen zu erzeugen. Dazu werden alle Positionen bestimmt, bei denen beide Individuen gleiche Typen haben. Dabei kommt jede Position mindestens einmal vor, da jede Instanz den Typ "rdfs:resource" hat. Umso

mehr Typen bei einer Position gleich sind, desto häufiger wird diese Position in einer Liste gespeichert. Wenn nun eine Position zufällig gleichverteilt aus der Liste der Positionen gewählt wird, ist die Wahrscheinlichkeit für die Wahl einer Position mit vielen gleichen Typen entsprechend erhöht. Da die möglichen semantischen Verknüpfungen zwischen zwei Instanzen von deren Typen abhängen, kann davon ausgegangen werden, dass hinter einer so gefundenen Position auch mit erhöhter Wahrscheinlichkeit wieder eine semantische Verknüpfung existieren kann, nachdem der Crossover-Operator durchgeführt wurde. Auch bei diesem Rekombinationsoperator muss darauf geachtet werden, dass keine Individuen mit Kreisen entstehen, da sonst beliebig hohe Fitnesswerte durch einfaches wiederholen eines Teilpfades möglich wären. Der Parameter ρ wurde für diese Art der Rekombination insofern berücksichtigt, als dass entsprechend viele Individuen in einem Schritt gewählt und auch die gleiche Anzahl neuer Individuen erzeugt werden. So wirkt sich ρ in diesem Fall nicht wesentlich auf die Laufzeit des Algorithmus aus, da bei steigendem ρ entsprechend weniger Rekombinationsschritte durchgeführt werden müssen.

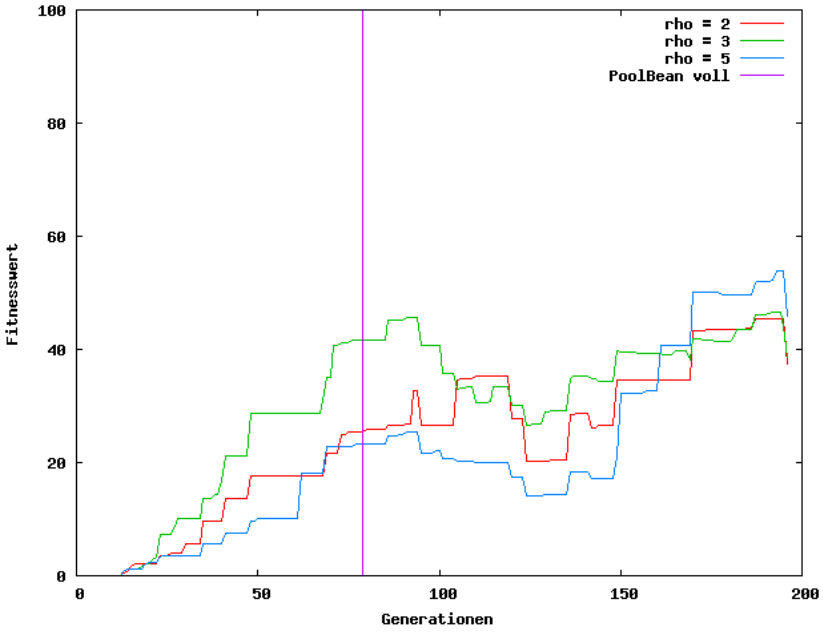

Abbildung 6.10: Fitnessverlauf bei typbasiertem Crossover

Im Gegensatz zu einer Rekombination, die ein neues Individuum aus zufällig gewählten Teilpfaden der Eltern erstellt, ergibt sich hier kein Vorteil (siehe Abbildung 6.10). Die maximal durchschnittliche Fitness liegt deutlich unterhalb der Ergebnisse, die mit dem einfachen Rekombinationsoperator erzielt werden konnten. Auch das Verhalten bei wechselnden Interessen des Benutzers wurde nicht verbessert. Für verschiedene Werte von ρ konnte keine Änderung des Verhaltens beobachtet werden, was sich dadurch erklären lässt, dass es keinen wesentlichen Unterschied macht ob eine Anzahl von Rekombinationen in einem Schritt durchgeführt werden oder in mehreren Aufrufen. Wird die typbasierte Re-

kombination noch verstärkt, indem nur die "Direct Types" berücksichtigt werden, lässt sich die Leistungsfähigkeit nicht verbessern. Nach wenigen Iterationen entstehen ausschließlich Individuen mit der Pfadlänge 1. Diese können mit der hier genutzten Fitnessberechnung keinen Fitnesswert größer null erreichen. Der Algorithmus stagniert also über seine komplette Laufzeit bei null.

Eine größere Leistungsfähigkeit könnte sich bei der Nutzung in Generationen mit einer höheren Diversität ergeben. Dazu wird dieses Verfahren in Kombination mit dem Fitness-Sharing-Verfahren (siehe Abbildung 6.16) noch einmal verwendet. In einer Generation mit geringer Diversität werden durch die Rekombination mit typbasiertem Crossover nur noch sehr ähnliche Individuen erzeugt. Diese bringen den Algorithmus nur noch in sehr kleinen Schritten vorwärts. Auch die schlechte Reaktionsfähigkeit auf die dynamische Fitnesslandschaft läßt sich dadurch erklären.

Fitnessberechnung mit verschiedenen Bestrafungsfunktionen

Die Fitnessberechnung kann mit Hilfe des Moduls MueRhoLambdaFitnessCalcPenalty mit einer Liste von Bestrafungsfunktionen versehen werden. Diese werden in der angegebenen Reihenfolge auf jedes Individuum angewendet, dessen Fitnesswert bestimmt werden soll. Dabei kann der Fitnesswert sowohl absolut als auch relativ angepasst werden. Standard für alle Fitnessberechnungen ist die DeathPenalty, die dafür sorgt, dass für jeden Übergang zwischen zwei Instanzen des Pfads auch eine semantische Verknüpfung existiert. Jeder Übergang, der keine solche Verknüpfung erlaubt, wird aufsummiert und diese Summe schließlich als negativer Fitnesswert gesetzt. Negative Fitnesswerte entstehen somit ausschließlich durch diese Bestrafungsfunktion und führen in der Regel dazu, dass das Individuum während der Selektion nicht in die Folgegeneration übernommen wird.

Zusätzlich zu dieser Bestrafungsfunktion, die eine Nebenbedingung der semantischen Navigation umsetzt, wurden noch zwei weitere Bestrafungsfunktionen als Module mit unterschiedlichen Zielen implementiert.

Die UselessPathPartPenalty entstand aus der Beobachtung, dass Individuen mit langen Pfaden entstehen können, bei denen nur ein kleiner Teilpfad für den Fitnesswert verantwortlich ist, während der restliche Pfad den Nutzer an Stellen im semantischen Web führt, die keiner seiner Interessen entsprechen. Da diese überflüssigen Teilpfade sich nicht auf die Fitness des Individuums auswirken, sollte mit Hilfe der UselessPathPartPenalty eine Bestrafungsfunktion geschaffen werden, die für jede Instanz kontrolliert ob diese mindestens einen Typ besitzt, der während der Fitnessberechnung zu einer Erhöhung des Fitnesswerts geführt hat. Alle Instanzen, die keinen solchen Typ besitzen, werden aufsummiert und der erreichte Fitnesswert folgendermaßen verändert:

$$\phi'(x_i) = \frac{\phi(x_i)}{e^{\text{Anzahl nicht fitnessrelevanter Instanzen}}}$$

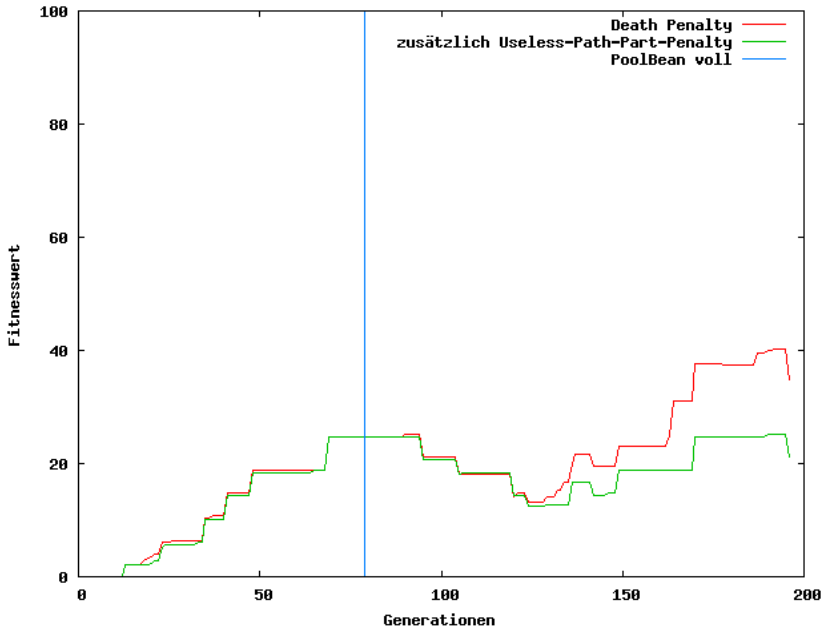

Abbildung 6.11: Fitnessverlauf mit und ohne Nutzung der UselessPathPartPenalty

Die Unterstützung der Suche durch die UselessPathPartPenalty konnte allerdings nicht bestätigt werden (siehe Abbildung 6.11). In der Startphase beeinflusst die zusätzliche Bestrafungsfunktion den Verlauf des Algorithmus gar nicht, da die zu Beginn gefundenen Pfade in der Regel kurz sind und fast alle Instanzen zum Fitnesswert beitragen. Im späteren Verlauf entwickeln sich aber immer mehr längere Pfade, die über entsprechende Teilpfade verfügen. Der Fitnesswert sinkt allerdings durch die Bestrafungsfunktion nur ab. Neue Individuen mit höheren Fitnesswerten, die keine überflüssigen Pfadteile enthalten, werden dadurch nicht schneller gefunden. Das hier zu beobachtende Verhalten des Algorithmus lässt darauf schließen, dass die Pfadteile, die nicht zum Fitnesswert beitragen, durchaus einen wichtigen Teil der Suche darstellen, da diese in neuen Kombinationen oder durch Mutation zu besseren Suchergebnissen führen können. Der Einsatz der UselessPathPartPenalty hemmt die Suche an dieser Stelle also dadurch, dass Individuen auf dem Weg zu einem guten Suchergebnis unnötig bestraft werden.

Nicht im Sinne einer eigentlichen Bestrafungsfunktion arbeitet das Fitness-Sharing. Dieses Verfahren zielt auf eine höhere Diversität innerhalb einer Generation ab. Für jedes Individuum wird seine Ähnlichkeit zu allen anderen Individuen der Generation bestimmt. Diese Ähnlichkeit wird durch einen Vergleich der fitnessrelevanten Typen (siehe Kapitel 4.5.1 und Kapitel 4.2) definiert. Umso mehr Individuen exisitieren, die sich ähnlich sind und umso stärker diese Ähnlichkeit ist, desto stärker fällt auch die Bestrafung des Fitnesswertes aus. So soll es neuen Individuen erleichtert werden sich in einer Generation durchzusetzen, die schon stark gegen ein lokales Optimum konvergiert.

Die wesentliche Einstellungsmöglichkeit dieses Moduls ist der Wert σ_{share}. Dieser entspricht dem Verhältnis zwischen ungleichen und gleichen fitnessrelevanten Typen, welches unterschritten werden muss, damit zwei Individuen als "ähnlich" gelten.

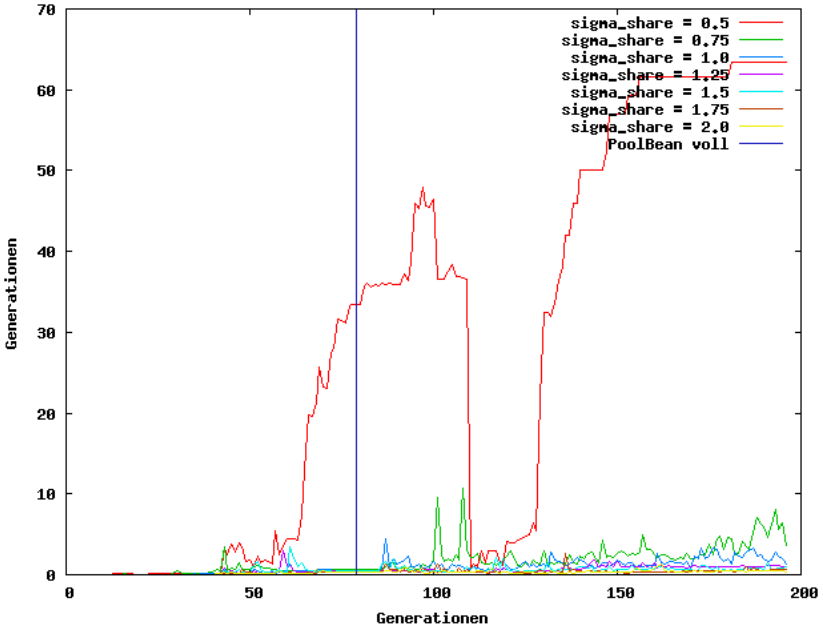

Abbildung 6.12: Fitnessverlauf mit verschiedenen Werten von σ_{share}

Mit steigendem σ_{share} werden die einzelnen Individuen schneller als " ähnlich " bewertet und stärker bestraft. Erst bei einem Wert von $\sigma_{share} = 0.5$ sieht man, dass hier auch höhere Fitnesswerte erreicht werden. Trotz der niedrigen Werte bleibt der Algorithmus aber funktionsfähig und liefert gute Ergebnisse. Ohne den Fitnessverlauf für $\sigma_{share} = 0.5$ ergibt sich bei geänderter Skalierung der Fitnesswerte ein deutlicheres Bild davon, wie sich das Verhalten des Algorithmus ändert (siehe Abbildung 6.13).

Anstatt der in vorangegangenen Abbildungen üblichen Treppen ergeben sich in dieser Grafik Spitzen, die durch einzelne Individuen verursacht werden, die sich aus der Masse hervorheben und somit der Bestrafungsfunktion entgehen. Diese Individuen übernehmen innerhalb kürzester Zeit die komplette Generation und sinken dadurch sofort wieder auf einen sehr niedrigen Fitnesswert. Dieses Verhalten wird bei steigendem σ_{share} immer deutlicher und ermöglicht einen rasanten Wechsel zwischen verschiedenen Themen. Das ursprüngliche Ziel, die Erhöhung der Diversität, wird allerdings für Werte von $\sigma_{share} \geqslant 1.0$ nicht erreicht. Für einen einzelnen Lauf des Algorithmus ergibt sich als Beispiel folgendes Bild (siehe Abbildung 6.14).

Deutlich zu sehen sind einzelne Individuen, deren Fitnesswerte stark nach oben abweichen. Kurz darauf zeigen sich meist zwei bis drei erhöhte Werte, die dann schnell wieder in der Masse der Individuen verschwinden. Die meisten Indidivuen halten sich dabei relativ konstant auf einem Wert. Das schnelle Auswechseln aller Individuen lässt sich für

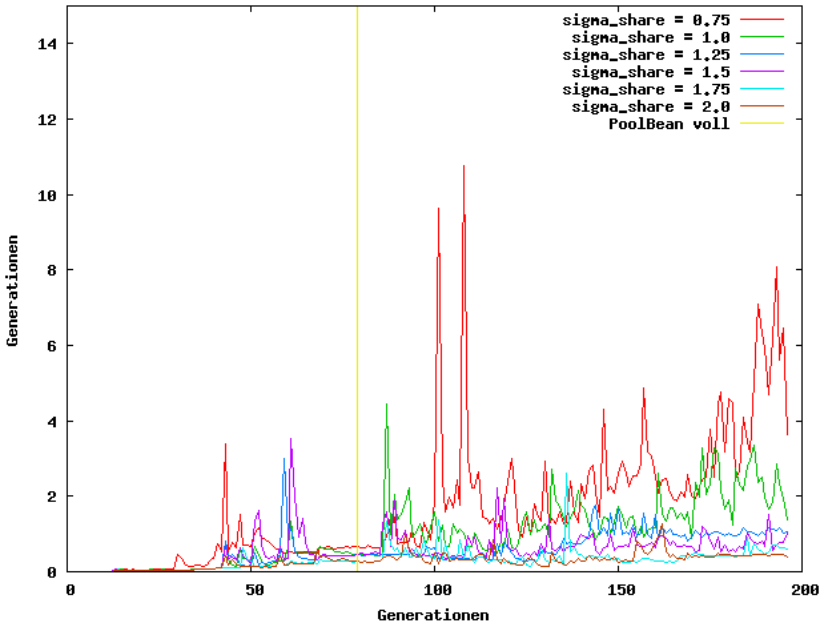

Abbildung 6.13: Fitnessverlauf mit verschiedenen Werten von σ_{share} ohne die Werte für $\sigma_{share} = 0.5$

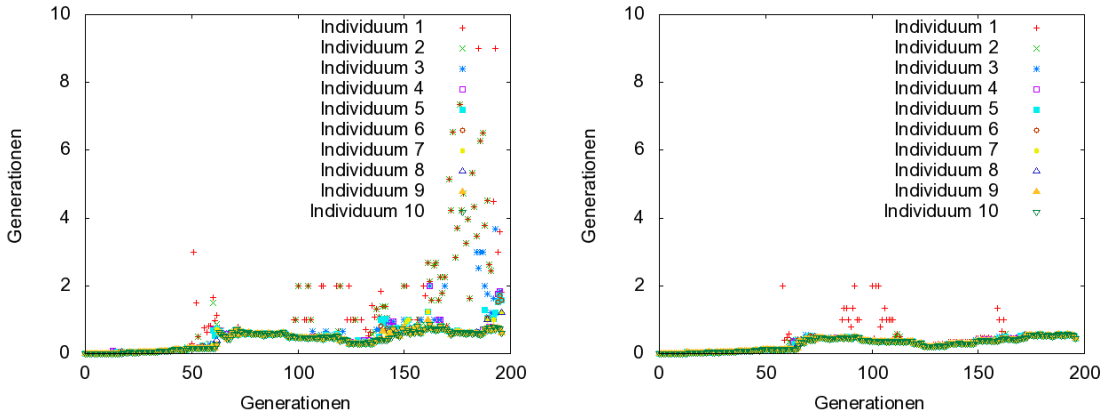

Abbildung 6.14: Fitnesswerte der 10 besten Individuen einer Generation für $\sigma_{share} = 1.0$ (links) und $\sigma_{share} = 2.0$ (rechts)

$\sigma_{share} = 2.0$ noch stärker beobachten. Hier sind es oftmals nur noch einzelne Individuen, die sofort zu einem Wechsel führen.

Aufgrund der hohen Anzahl nahezu gleicher Individuen liegen die Fitnesswerte so niedrig, dass ein einzelnes Individuum immer alle anderen in kurzer Zeit ersetzen kann. Mit einer niedrigeren Anzahl von Individuen kann dieses Verhalten zwar etwas verbessert werden, allerdings ergeben sich dann die bereits vorher gezeigten Nachteile einer zu kleinen Generation (siehe Abbildung 6.15).

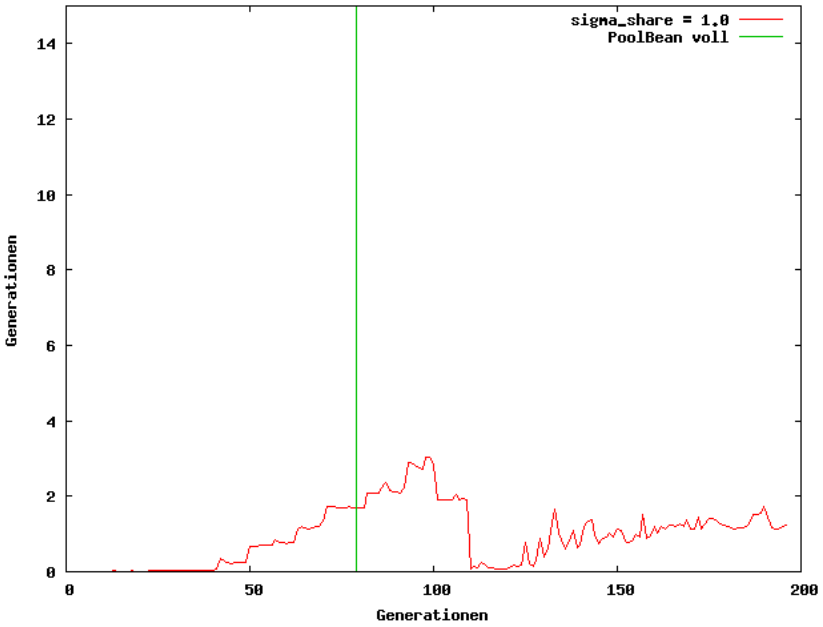

Abbildung 6.15: Verhalten bei $\sigma_{share} = 1.0$ und $\mu = 15$

Durch die kleinere Generation ist die Anzahl der ähnlichen Individuen entsprechend beschränkt. Dadurch fallen die Spitzen durch das Fitness-Sharing-Verfahren wesentlich kleiner aus. Allerings läßt sich auch ein starker Einbruch des Fitnessverlaufs feststellen, der nicht durch die Bestrafungsfunktion, sondern durch die mangelnde Größe der Generation verursacht wurde. Die beste Steigerung der Diversität des Suchergebnisses wurde für eine Generation von $\mu = 50$ bei einem Wert von $\sigma_{share} = 0.75$ gefunden. Für diesen Wert ergibt sich die beste Balance zwischen schnellen Themenwechseln und der gewünschten hohen Diversität. Durch die so erreichte bessere Streuung der Individuen im Suchraum wird der typbasierte Crossover-Operator für die Rekombination noch einmal interessant. Dadurch, dass ohne die Fitness-Sharing-Funktion alle Individuen sehr schnell gegen ein Optimum konvergieren, sind sich alle Individuen sehr schnell sehr ähnlich. Durch die typbasierte Rekombination konnten dann keine Individuen mehr entstehen, die sich in Richtung eines weitere Optimums bewegen.

Auch bei höherer Diversität kann mit Hilfe des typbasierten Crossover-Operator keine Verbesserung erreicht werden, die den zusätzlichen Aufwand durch das Ermitteln und Abgleichen der Typen rechtfertigen kann (siehe Abbildung 6.16). Im Wesentlichen entsteht aber auch kein Nachteil für die Leistungsfähigkeit des Systems. Die erreichten Fitnesswerte sowie die Reaktionsgeschwindigkeit auf Themenwechsel liegen in etwa auf einem Niveau.

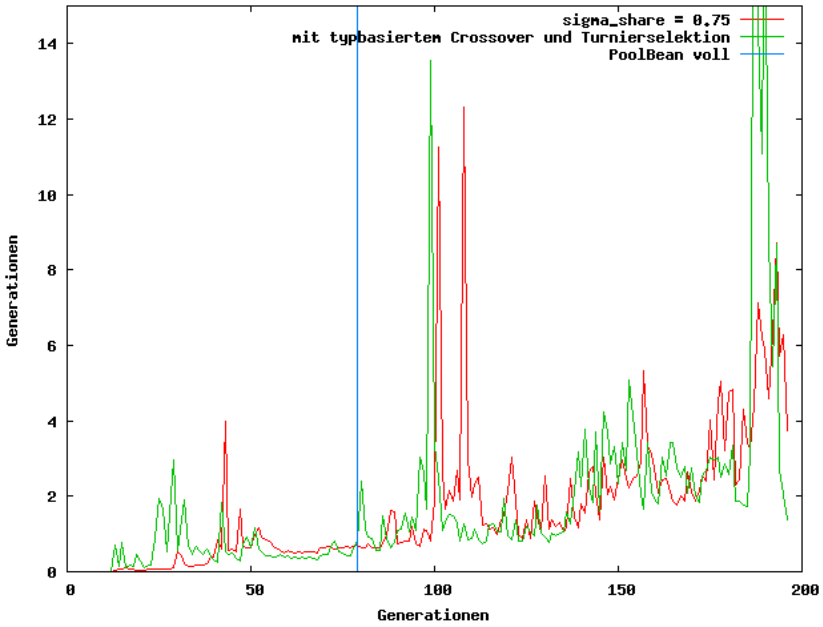

Abbildung 6.16: Fitness-Sharing in Kombination mit typbasiertem Crossover

Selektionsverfahren

Bei den Selektionsverfahren wird zwischen den diskriminierenden und nicht diskriminierenden Verfahren unterschieden. Im Rahmen einer Evolutionsstrategie wird in der Regel die diskriminierende Selektion eingesetzt. Diese sortiert die Individuen nach ihrem Fitnesswert und übernimmt nur die μ besten in die nächste Generation. Um aber eine höhere Diversität und eine bessere Anpassungsfähigkeit an die dynamische Fitnesslandschaft zu erreichen kann es durchaus sinnvoll sein auch nicht diskriminierende Verfahren zu betrachten. Drei mögliche nicht diskriminierende Varianten wurden dafür implementiert, von denen die Selektion mit fitnessbasierter Wahrscheinlichkeit (Monte-Carlo-Verfahren) aber aufgrund vieler bekannter Schwächen nicht weiter betrachtet werden soll.

Selektion auf Basis eines Rankings Bei dieser Variante der Selektion wird jedes Individuum mit einer festen Anzahl anderer Individuen nacheinander verglichen, die zufällig aus der Generation gewählt werden.

Jedes Mal wenn das betrachtete Individuum besser ist wird ein Zähler hochgesetzt, der hinterher der Summe der Siege dieses Individuums entspricht. Anhand der Anzahl dieser Siege werden am Ende alle Individuen sortiert und die besten μ in die nächste Generation übernommen.

Über die Anzahl der Wettkämpfe, die für jedes Individuum durchgeführt werden, kann der Selektionsdruck entsprechend konfiguriert werden. Der angegebene Wert entspricht dann jeweils der Anzahl der Wettkämpfe, die der Algorithmus pro Individuum ausführt.

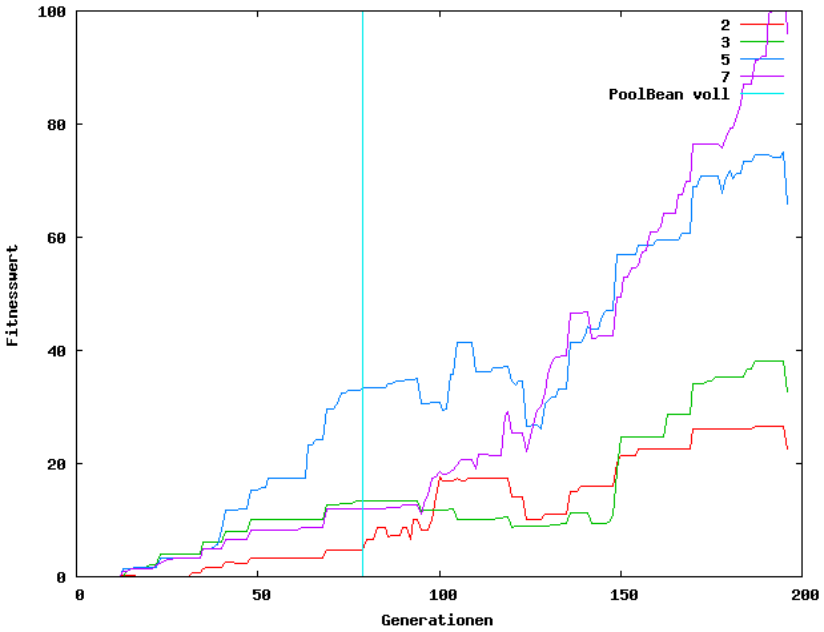

Abbildung 6.17: Selektion auf Basis eines Rankings mit variiertem Selektionsdruck

Abhängig von diesem Selektionsdruck ändert sich auch das Verhalten des Algorithmus (siehe Abbildung 6.17). Die Leistungsfähigkeit steigt dabei stetig mit der Anzahl der Wettkämpfe. Eine geringere Neigung zu Einbrüchen aufgrund der dynamischen Fitnesslandschaft ist erst ab einem Wert von sieben erkennbar. Hier steigt die Fitnesskurve jedoch weniger schnell.

Turnierselektion Bei der Turnierselektion werden eine bestimmte Anzahl Individuen aus der Generation zufällig ausgewählt, von diesen wird das beste Individuum direkt in die nächste Generation übernommen. Der Selektionsdruck kann hier durch die Anzahl der in einer Runde gezogenen Individuen eingestellt werden. Selbst bei hohen Werten für die Anzahl der gezogenen Individuen (siehe Abbildung 6.18) ist der Selektionsdruck nicht hoch genug, um an die Leistung bei Verwendung anderer Selektionsverfahren anzuschließen.

Im Vergleich zu der diskriminierenden Selektion ergeben sich hier keine wesentlichen Vorteile (siehe Abbildung 6.19). Die Turnierselektion bleibt in ihrer Leistungsfähigkeit zu weit hinter anderen Verfahren zurück. Mit Hilfe des Rankings lassen sich zwar gute Ergebnisse erzielen, allerdings leiden diese unter einer langsamen Startphase. Ist diese überwunden, lassen sich sowohl hohe Fitnesswerte erzielen als auch die Sprünge der dynamischen Fitnesslandschaft gut ausgleichen.

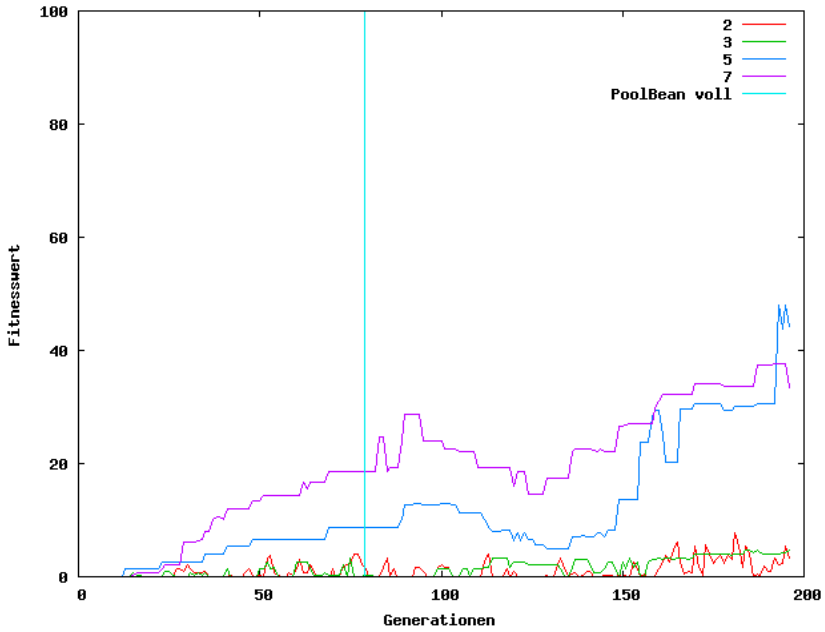

Abbildung 6.18: Turnierselektion mit variiertem Selektionsdruck

6.3.5 Konfiguration des Niching-Verfahrens

Um die Diversität der Suchergebnisse steigern zu können wurde die Evolutionsstrategie um einen Cluster-Based-Niching-Ansatz erweitert. Dieser folgt dem Vorbild aus der Natur und setzt nur Individuen in Konkurrenz zueinander, die sich eine Nische teilen. Eine Nische entspricht hierbei einem lokalen Optimum. Das Ziel ist es dabei möglichst viele lokale Optima gleichzeitig zu finden. Im Falle der semantischen Navigation soll dadurch das Bild der Links verbessert werden. Mit einer einfachen Evolutionsstrategie konvergieren alle Individuen schnell zu einem einzelnen Optimum. Ein Benutzer verfolgt aber bei seiner täglichen Arbeit mehrere verschiedene Interessen und Themengebiete, die in der Ausgabe der semantischen Navigation berücksichtigt werden sollen. Der hier verwendete Test-Benutzer durchläuft insgesamt fünf verschiedene Interessensgebiete jeweils zweimal. Für das hier angewendete Verfahren ist in diesem Fall also eine Anzahl von fünf gefundenen Nischen wünschenswert. Sollen dem Benutzer mehr Links angezeigt werden als Nischen gefunden wurden, so müssen aus einigen Nischen mehrere Individuen gewählt werden.

Wie bereits bei der Verwendung von Fitness-Sharing wird auch hier die Ähnlichkeit zweier Individuen als Abstandsmaß genutzt. In einem Cluster befinden sich also nur Individuen, deren Verhältnis von ungleichen und gleichen fitnessrelevanten Typen kleiner ist als ein vorgegebener Wert σ_{share}. Für das Fitness-Sharing brachte dieser Wert bei $\sigma_{share} = 0.75$ optimale Ergebnisse. Für die Nutzung im Rahmen des Niching-Verfahrens wurde ein ähnlicher Wertebereich getestet (siehe Abbildung 6.20).

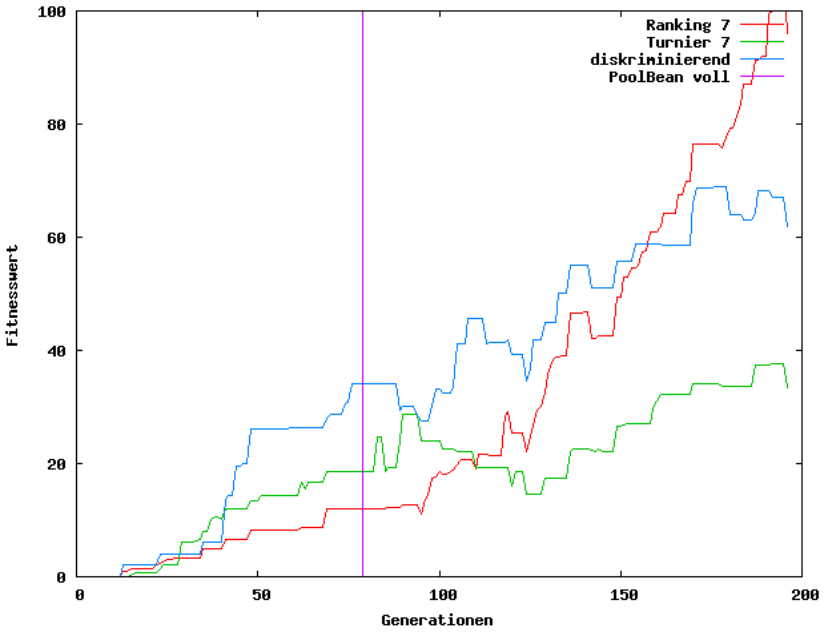

Abbildung 6.19: Die Selektionsverfahren im direkten Vergleich

Dabei täuscht die Leistungsfähigkeit für den Wert $\sigma_{share} = 0.5$. Bei diesem oder kleineren Werten können zwei Individuen nur in einer Nische sein, wenn ihre fitnessrelevanten Typen nahezu identisch sind. Für die Individuen gilt daher, dass sie trotz bereits hoher Ähnlichkeit immernoch voneinander unterschieden werden. In der ersten Phase des Algorithmus werden alle bereits vorhandenen Gruppen nach neuen Gruppen durchsucht, die sich während der letzten Iteration gebildet haben könnten. Dabei ergeben sich extrem viele unterscheidbare Gruppen der Größe eins. Sobald eine Gruppe aber nicht aus mindestens zwei Individuen besteht wird sie in einer abschließenden Phase wieder zur ersten Gruppe hinzugefügt. Erst, wenn sich nach einigen Iterationen die Individuen aneinander angleichen entstehen größere Gruppen aus identischen Individuen.

Diese bleiben dann als eine Art Konstanten erhalten. Änderungen der Individuen in diesen Gruppen sind nicht mehr möglich, da, sobald ein Individuum z.B. durch Mutation von den anderen abweicht, es sofort als nicht mehr der Nische zugehörig identifiziert und somit der ersten Gruppe hinzugefügt wird (für Details siehe Algorithmus 4.1).

Ab einem Wert von $\sigma_{share} = 0.75$ tritt dieses Verhalten nicht mehr auf. In den einzelnen gefundenen Nischen ist weitere Optimierung möglich, wodurch diese nach einigen Iterationsschritten auch wieder wegfallen können. Erst bei Werten von $\sigma_{share} > 1.0$ wird deutlich, dass nicht mehr alle Nischen gefunden werden, da die einzelnen Pfade mit dieser Einstellung nicht mehr gut genug zu unterscheiden sind.

Eine deutliche Steigerung der gefundenen Nischen ist durch den zusätzlichen Einsatz von Fitness-Sharing zu erwarten. Durch die Steigerung der Diversität in den einzelnen Nischen sollen so schneller neue Nischen gefunden und die Gesamtzahl der Nischen erhöht

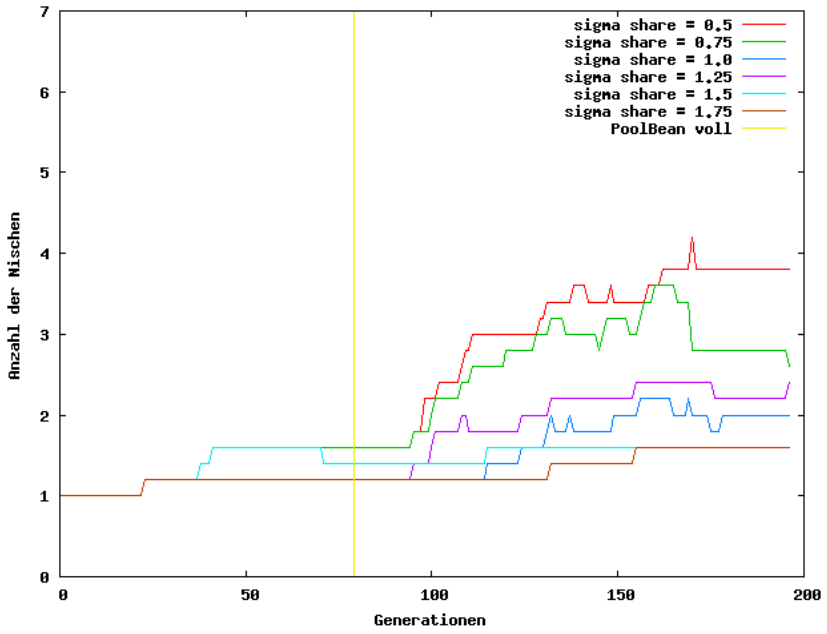

Abbildung 6.20: Anzahl der gefundene Nischen in Abhängigkeit von σ_{share}

werden. Die Pfade des verwendeten Testnutzers entsprechen mindestens fünf Themengebieten, die durch den Algorithmus erkannt werden sollten. Wesentlich mehr verschiedene Themen sind nicht zu erwarten, da dafür die Größe der PoolBean nicht ausreicht. Mehr als zehn Themen sind bereits für die Darstellung in der Benutzeroberfläche nicht notwendig, da hier entsprechend der Konfiguration der Oberfläche nur zehn Themen dargestellt werden. Für die Laufzeit des Algorithmus sind aber zehn Themengebiete bereits sehr kritisch. Daher sollte diese Anzahl möglichst nicht überschritten werden.

Wird der Algorithmus mit den hier ermittelten Werten für $\sigma_{share} = 0.75$ für das Fitness-Sharing und das Cluster-Based-Niching-Verfahren gestartet, ergibt sich nach kürzester Zeit ein eindeutiges Bild: Innerhalb weniger Generationen entdeckt der Algorithmus eine viel zu hohe Anzahl von Nischen, die die Antwortzeiten des Servers auf ein unzumutbares Maß von weit über zehn Sekunden erhöhen. Diese Konfiguration lässt sich für eine bessere Performance und eine geeignetere Anzahl an Nischen aber leicht anpassen, da sich auch für $\sigma_{share} = 1.0$ bei beiden Verfahren noch eine gute Nutzbarkeit gezeigt hat. Für diese Einstellung zeigen sich im Laufe der ersten 200 Generationen sehr gute Ergebnisse (siehe Abbildung 6.21). Die Nischen werden schneller gefunden und pendeln nach einiger Zeit zwischen Werten von sechs und acht. Die Auswahl, die der Benutzer durch die semantische Navigation geboten bekommt, enthält dadurch ausreichend viele unterschiedliche Links.

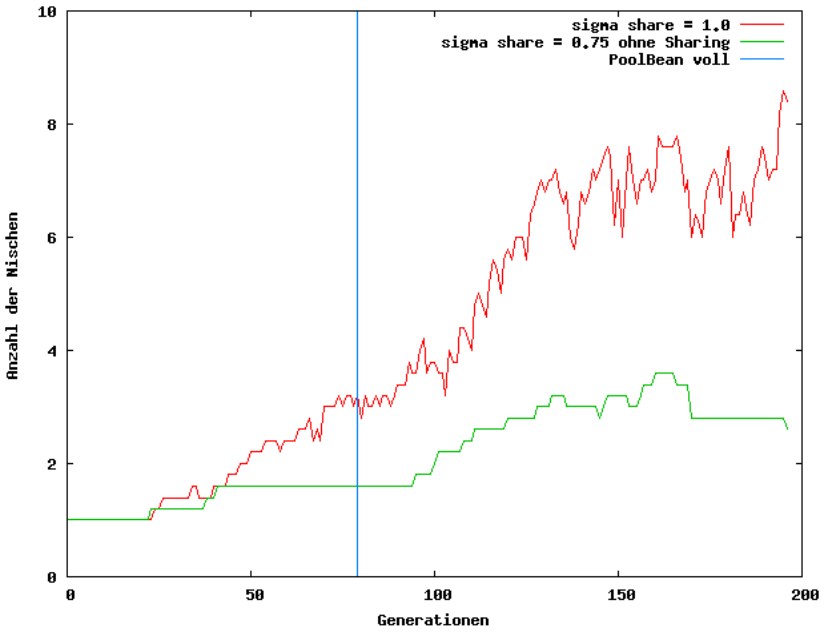

Abbildung 6.21: Anzahl der gefundene Nischen mit und ohne Fitness-Sharing

6.4 Welche Verfahren sind geeignet?

Für das Auffinden eines Links, der genau einem Thema des Benutzers entspricht, ist eine einfache Evolutionsstrategie durchaus geeignet. Diese konvergiert schnell gegen ein lokales Optimum. Themenwechsel bereiten dem Verfahren allerdings Probleme: Bei ausreichend großer Parent- sowie Offspring-Generation und genügend viele Individuen für die Rekombination können die Themenwechsel zwar schnell berücksichtigt werden, allerdings bleibt der Algorithmus immer bei einem einzelnen Optimum. Diese Werte wurden bei $\mu = 50$, $\rho = 5$ und $\lambda = 200$ festgelegt und liegen damit bereits am oberen Rande dessen, was für eine gute Laufzeit des Algorithmus noch verantwortbar ist. Dabei muss darauf Rücksicht genommen werden, dass es sich um eine Webapplikation handelt, in der von einem Benutzer eine gewisse Erwartungshaltung an die Reaktionszeiten auf einen Klick existiert. Diese Reaktionszeit darf zwei Sekunden nicht überschreiten, da die Benutzer gerade im Umfeld des Web 2.0 mit Ajax-basierten Oberflächen zunehmend eine ähnliche Performance forden, die sie von Desktopapplikationen gewohnt sind. Ein wesentlicher Punkt für die Laufzeit ist die Anzahl der Fitnessberechnungen pro Generation. Da davon ausgegangen wird, dass die Interessen des Benutzers einem stetigen Wandel unterliegen, muss für alle Individuen in jeder Iteration der Fitnesswert neu bestimmt werden.

Die schlechte Diversität der Suchergebnisse bei der einfachen Evolutionsstrategie liegt aber auch am Aufbau der Fitnesslandschaft, die sich durch die hier genutzte Bewertung der Individuen auf Basis von gleichen Pfaden in den Typen ergibt. Lange Pfade in den Typen, bei denen auch mehrere Typen pro Instanz gleich sind, können zu sehr hohen Fit-

nesswerten führen. Dieses Phänomen konnte bei der Initialisierung durch Benutzereingaben genauer beobachtet werden. Obwohl Individuen oft schon bei einem Fitnesswert von über 30 sinnvolle Ergebnisse liefern können, haben z.B. Individuen, deren Pfad genau einem Pfad aus der PoolBean entsprechen, oftmals Werte von weit über 500. Aus diesen extremen Optima kann der Algorithmus selbst nicht ohne weitere Hilfsmittel ausbrechen bis der entsprechende Pfad wieder aus der PoolBean entfernt wird.

Um den Algorithmus leistungsfähiger zu machen und die Diversität der Suchergebnisse zu erhöhen, wurden eine alternative Rekombination sowie verschiedene nicht diskriminierende Selektionsverfahren getestet. Diese hatten alleine aber keinen positiven Einfluss auf die Ergebnisse.

Die Diversität ließ sich auch mit Hilfe von Fitness-Sharing nur unwesentlich erhöhen. Im Wesentlichen wurde durch den Einsatz von Fitness-Sharing die Reaktionsgeschwindigkeit auf Themenwechsel erheblich verbessert. Dafür wurde eine Definition für den Abstand zweier Individuen auf der Basis des Verhältnis zwischen ungleichen und gleichen fitnessrelevanten Typen entwickelt. Auch wenn durch dieses System die Fitnesswerte selbst stark reduziert werden, bleibt ihr Verhältnis und damit die Funktionalität des grundlegenden Algorithmus erhalten. Für das Verfahren wurde ein Wert von $\sigma_{share} = 0.75$ als geeignet ermittelt. Dies entspricht einem Verhältnis von drei ungleichen zu vier gleichen fitnessrelevanten Typen. Bei diesem Wert werden ausreichend viele Individuen als " ähnlich" erkannt. Sobald einzelne neue Individuen entstehen, können sich diese schnell in der Generation verbreiten und so den Wechsel zu einem neuen Thema einleiten.

Einen deutlichen Fortschritt bei der Verbesserung der Suchergebnisse brachte die Nutzung des Cluster-Based-Niching-Verfahrens. Durch das Aufteilen der Individuen auf Nischen, die einzelnen Interessen des Benutzers entsprechen, ist es möglich den Algorithmus nach mehreren lokalen Optima gleichzeitig suchen zu lassen. Der Benutzer bekommt so für jede gefundene Nische Links in seiner Navigation angeboten, die sich deutlich voneinander unterscheiden. Auch hier wurde für die Unterscheidung der einzelnen Individuen das Abstandsmaß genutzt, das schon für das Fitness-Sharing zum Einsatz gekommen ist. Um eine möglichst der Anzahl der Interessen des Benutzers entsprechende Anzahl von Nischen zu finden, wurde hier der Wert $\sigma_{share} = 0.75$ ermittelt.

Mit Hilfe des zusätzlich angewandten Fitness-Sharing-Verfahrens konnte die Anzahl der gefundenen Nischen und die Geschwindigkeit mit der diese gefunden wurden noch deutlich verbessert werden. Konfiguriert man beide Systeme mit den hier ermittelten Werten von $\sigma_{share} = 0.75$ so führt dies zu einer explosionsartigen Erhöhung der Anzahl der gefundenen Nischen. Die Laufzeit des Algorithmus leidet unter dieser hohen Anzahl zu stark, so dass die Antwortzeiten des Servers auf ein unzumutbares Maß ansteigen. Außerdem macht diese enorm hohe Anzahl von Nischen nur Sinn, wenn auch entsprechend viele Links angeboten werden sollen. Um die Übersichtlichkeit zu gewährleisten wurde das System hier aber nur für zehn Links in der semantischen Navigation konfiguriert. Da für die PoolBean eine Größe

von maximal 15 Pfaden vorausgesetzt wurde, ist es auch unwahrscheinlich, dass diese 15 Pfade wesentlich mehr als zehn verschiedenen Interessen des Benutzers entsprechen. Man kann also das System zu Gunsten der Laufzeit etwas drosseln, indem man für beide Verfahren einen Wert von $\sigma_{share} = 1.0$ nutzt. Hier stellt sich ein sehr gutes Verhalten ein, bei dem der Algorithmus zwischen fünf und zehn verschiedene Themen aus dem Benutzerverhalten ermittelt und entsprechende Links anbietet.

Kapitel 7

Zusammenfassung und Ausblick

Grundsätzlich ist diese Arbeit mit dem Problem konfrontiert, dass das in der Einleitung skizzierte große System für ein Intranet aus Blogs, Wikis, CMS und Foren in einer Portalumgebung auf Basis eines alles umschließenden semantischen Webs in der vorgestellten Form noch nicht existiert. Trotzdem ist diese Entwicklung absehbar, wenn man aktuelle Produkte großer Softwarehäuser für diesen Bereich betrachtet. Insofern ist die Problemstellung der mangelnden Übersichtlichkeit solcher Systeme noch nicht in dem Maß präsent. Ein semantisches Wiki stellt nur einen Ausschnitt dieses Gesamtsystems dar und kann dennoch bereits ein gutes Beispiel für die vorgestellte Problematik liefern, da sich gerade die Inhalte eines Wikis aufgrund ihrer Struktur nur mit starken Einschränkungen in einer klassischen Navigation abbilden lassen.

Die mangelnde Verbreitung solch offener Systeme ist dabei auf die hohe Komplexität der Bedienung zurückzuführen. Für den Benutzer wurden noch keine Assistenzsysteme entwickelt, die es erlauben nicht nur semantische Inhalte zu erschaffen, sondern auch Ontologien selbst zu bearbeiten, um sie den wachsenden Anforderungen in einem lebendigen System anzupassen. Um die Funktionsweise und Leistungsfähigkeit evolutionärer Algorithmen für eine semantische Navigation zu testen konnte somit kein System gefunden werden, auf dem reale Benutzereingaben für die Tests zur Verfügung gestanden hätten. Um trotzdem eine brauchbare Testumgebung zu entwickeln wurden die Inhalte im IkeWiki an den Aufbau eines schematisierten Firmenintranets angelehnt. Die darauf erfolgten Benutzereingaben wurden durch einen Webtest simuliert. Durch diese gut reproduzierbaren Eingaben wurde die Interpretation bestimmter Ereignisse bei der Ausgabe erleichtert. Trotzdem musste an dieser Stelle auf eine Aussage eines realen Benutzers über die tatsächliche Nützlichkeit der angebotenen Links in einer Umgebung mit echten Inhalten verzichtet werden.

Die angewendete Evolutionsstrategie wurde auf Basis der zentralen Ideen der Darstellung eines Individuums als Pfad im semantischen Web und der Fitnessberechnung durch einen typbasierten Vergleich mit den Inhalten einer PoolBean entwickelt. Diese grundlegenden Ideen führten schnell zu einem funktionstüchtigen System, das in der Lage ist ein

einzelnes Thema des Benutzers zu erkennen und für dieses alternative Links anzubieten. Die Schnelligkeit, mit der dieses Thema gefunden wurde, sowie die Reaktionszeit bei einem Themenwechsel und die Qualität der Ergebnisse, die sich in den Fitnesswerten wiederspiegelt, wurden durch die Einstellung der exogenen Parameter optimiert. Diese wurden für ein leitungsstarkes System bei bei $\mu = 50$, $\rho = 5$ und $\lambda = 200$ festgelegt.

Die Module für die Initialisierung, Rekombination, Mutation, Selektion und Fitnessberechnung mussten einzeln auf die Anforderungen in einem semantischen Web angepasst werden. Dabei konnten oft nur sehr einfache Verfahren überhaupt so angepasst werden, dass sie auf der genutzten Darstellung eines Individuums funktionieren können. Für die getesteten Varianten der Initialisierung, Rekombination und nicht diskriminierenden Selektion konnte hier nachgewiesen werden, dass sie zu keiner Verbesserung der Ergebnisse führen. Allerdings besteht hier weiteres Verbesserungspotential, wodurch diese Module die erwarteten Leistungen eventuell noch erreichen können. Für die jetzigen Implementierungen gilt, dass sie bei leicht erhöhten Laufzeiten jeweils zu geringfügig niedrigeren Fitnesswerten führten.

Die Optimierung der Suchergebnisse für ein einzelnes Thema des Benutzers reicht nicht aus, um in der semantische Navigation mehrere Links zu präsentieren. Da der Benutzer aber mit hoher Wahrscheinlichkeit mehrere Interessen verfolgt, sollen ihm möglichst zu jedem seiner Themen entsprechende Links angeboten werden. Durch die Erhöhung der Diversität der Suchergebnisse wurde dies erreicht. Dazu wurde eine Kombination des Cluster-Based-Niching-Verfahren mit Fitness-Sharing als Bestrafungsfunktion für sich stark ähnelnde Individuen eingesetzt. Beide Verfahren konnten mit einem Wert von $\sigma_{share} = 1.0$ so eingestellt werden, dass für den Testbenutzer nach einer gewissen Startphase zwischen fünf und zehn Nischen gefunden wurden, wobei eine Nische einem Interesse des Benutzers entspricht. Jedes Interesse stellt hier also ein lokales Optimum dar.

Dabei pendelt die Anzahl der gefundenen Nischen hier in einem sehr guten Intervall, da immer mindestens die fünf Interessen berücksichtigt werden, die für den Testbenutzer beabsichtigt waren, aber die Anzahl der Nischen auch nie zu hoch wird, wodurch die Anforderungen an die Hardware des Servers deutlich steigen würde. Die durch diese Konfiguration erreichten Ergebnisse habe einen Stand erreicht, der für den ausstehenden Test an realen Benutzern eine hohe Erfolgswahrscheinlichkeit erwarten lässt. Um einen Test mit realen Benutzern gut durchführen zu können, sollte das Verhalten der Benutzer in einem ersten Schritt analysiert werden. Dabei sollten Daten über die durchschnittliche Länge eines Pfads, die Anzahl der Pfade, die pro Tag geklickt werden, sowie die durchschnittliche Anzahl der Themen, die einen Benutzer gleichzeitig beschäftigen, erhoben werden. Anhand dieser lässt sich dann feststellen, ob der hier angenommenen Wert von 15 Pfaden in der PoolBean ausreichend ist, um genügend Interessen des Benutzers aufzunehmen.

Die aktuelle Implementierung der semantischen Navigation hat alleine durch die Fittnesswertberechnung bereits eine hohe Laufzeit, die gerade bei vielen gleichzeitig arbeiten-

den Benutzern und häufigen Klicks hohe Ansprüche an die Hardware des Servers stellt. Für diese Arbeit wurden die PostgreSQL Datenbank und der Java-Applikationserver auf einem mit VMware (http://www.vmware.com/de/) simulierten Rechner installiert. Der simulierte Rechner wurde mit 2048 MB Hauptspeicher und 2 CPUs mit jeweils 2,5 GHz Taktfrequenz ausgestattet. Weil das System gerade für ein entsprechend großes Intranet mit vielen simultan arbeitenden Benutzern gedacht ist, da erst unter diesen Bedingungen überhaupt die Problematik der schlechten Übersichlichkeit entsteht, muss davon ausgegangen werden, dass leistungsstärkere Hardware auf jeden Fall notwendig ist. Allerings steckt gerade in Bezug auf die Speicherauslastung noch ein hohes Optimierungspotential in der Auslagerung von Inhalten des Speichers in die Datenbank. Auch die Berechnung der Fitness eines Individuums kann für eine bessere Laufzeit noch optimiert werden. Desweiteren ergibt sich bei der Fitnessfunktion das Problem, dass durch den minimalen Fitnesswert von null ein Plateau entsteht, welches vereinzelt dazu führen kann, dass der Algorithmus über lange Zeiträume nur schlechte Ergebnisse liefert. Durch eine Änderung der Fitnessfunktion, so dass auch Pfade der Länge null noch unterschiedliche Fitnesswerte erreichen können, könnte dieses Verhalten wahrscheinlich verbessert werden.

Um die semantische Navigation in der hier vorgestellten Konfiguration sinnvoll nutzen zu können, sollte das zugrundeliegende semantische Web dem Benutzer ausreichend Freiheiten in der Gestaltung seiner semantischen Verknüpfungen gewähren. Viele automatisch vergebene Typen und Verknüpfungen können schnell dazu führen, dass die Ergebnisse der evolutionären Suche alleine durch ihr extrem häufiges Vorkommen in den Pfaden stark beeinflusst werden. Diese Verknüpfungen müssten dann durch die konfigurierbaren Filter im LinkPathFilter aussortiert werden, um die Funktionalität der semantischen Navigation zu gewährleisten.

Eine Mindestanzahl von Benutzern setzt die semantische Navigation nicht voraus. Allerdings ergibt sich die Möglichkeit, durch die semantische Navigation Wissen im Unternehmen zu verbreiten, natürlich erst, wenn das Unternehmen eine Größe erreicht, in der nicht jeder Mitarbeiter die Arbeit jedes anderen unmittelbar kennt.

Um den Wissenstransport im Unternehmen weiter zu verbessern besteht die Möglichkeit das System um einen weiteren Benutzer zu erweitern. Dieser sollte nicht im System vorhanden sein, sondern eine Art Sammelstelle für alle Benutzer darstellen. Sämtliche Eingaben von allen Benutzern würden also nicht nur bei dem spezifischen Benutzer sondern zusätzlich bei einem Dummy-Benutzer in der PoolBean gespeichert. Die jeweils aktuellen besten Ergebnisse für diesen Dummy-Benutzer können dann in der semantischen Navigation unter dem Punkt "Das interessiert alle" präsentiert werden.

Anhang A

Weitere Informationen

Auf der beigelegten CD befinden sich sowohl die Quelltexte der hier genutzten Software IkeWiki mit den entsprechenden Anpassungen und Erweiterungen, die für diese Arbeit notwendig waren, als auch die Quelltexte der selbst implementierten Komponenten inklusive aller dafür verwendeten Bibliotheken in den jeweiligen Versionen. Zusätzlich befindet sich die API im HTML-Format und eine PDF-Version dieses Dokuments auf der CD. Die Lizenzen, unter denen die genutzte Software veröffentlicht wurde, liegen den entsprechenden Sourcen bei.

Abbildungsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] D. Goldberg, J. Richardson: *Genetic Algorithms with sharing for multimodal function optimization.* In: Grefenstette, J. (Herausgeber): *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, Seiten 41–49, Cambridge, Massachusetts, United States, 1987. L. Erlbaum Associates Inc.

[2] Droste, Stefan: *Zu Analyse und Entwurf evolutionärer Algorithmen.* Dissertation, Fachbereich Informatik, Universität Dortmund, 2000.

[3] Droste, Stefan und Dirk Wiesmann: *On representation and genetic operators in evolutionary algorithms.* Technical Report CI-41/98, Universität Dortmund, 1998.

[4] Felix Streichert, Gunnar Stein, Holger Ulmer und Andreas Zell: *A Clustering Based Niching EA for Multimodal Search Spaces.* In: P. Liardet et al. (Herausgeber): *Lecture Notes in Computer Science*, Seiten 293–305. Springer, Berlin, 2004.

[5] M. Bellmore, G. L. Nemhauser: *The Traveling Salesman Problem: A Survey.* Operations Research, 16(3):538–558, 1968.

[6] Schaffert, Sebastian: *IkeWiki: A Semantic Wiki for Collaborative Knowledge Management.* In: Tolksdorf, Simperl, Schild (Herausgeber): *Proceedings of the 1st International Workshop on Semantic Technologies in Collaborative Applications*, Seiten 388–394, Manchester, UK, 2006. IEEE Computer Society.

[7] Schwefel, Hans-Paul: *Evolutionsstrategie und numerische Optimierung.* Dissertation, Technische Universität Berlin, 1975.

[8] Shir, Ofer Michael: *Niching in Derandomized Evolution Strategies and its Applications in Quantum Designs.* Dissertation, Universiteit Leiden, 2008.

[9] Tim Berners-Lee, James Hendler, Ora Lassila: *The Semantic Web: a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities.* Scientific American, 284(5):34–43, 2001.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 24. Dezember 2008

Kay Thielmann