Multi-objective Routing mit Ameisenalgorithmen

Eine Semesterarbeit von
Peter Keller

Betreut durch
Placi Flury

Co-Betreuer:
Kostas Katrinis
Professor:
Bernhard Plattner

18. Februar 2004

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Gute Routingalgorithmen sind ein wichtiger Bestandteil von modernen Kommunikationsnetzen. Die Anforderungen an das Routing sind zusammen mit der Grösse der Netzwerke gewachsen. Die Daten müssen nicht nur ihr Ziel finden, sie sollen dabei auch den optimalen Pfad benutzen. Optimal kann hier aber verschiedenes bedeuten. Der Benutzer, der gerade per Telnet auf einer anderen Maschine eingeloggt ist, möchte eine möglichst kurze Verzögerung, um vernünftig arbeiten zu können. Jemand der grosse Dateien übertragen möchte, für den ist die Verzögerung unwichtig – das Einzige was zählt ist die verfügbare Bandbreite. Dabei haben wir aber noch die Interessen der Netzbetreiber vergessen, die den Verkehr mit möglichst geringen Kosten weiterleiten möchten.

Die immer grösser werdenden Netze und die verschiedenen Anforderungen haben dazu geführt, dass die verwendeten Algorithmen immer ausgefeilter, aber leider auch immer komplexer wurden. Einen Gegenpol dazu finden wir in der Natur. Das wohl prominenteste Beispiel hierzu sind die Ameisen. Sie sind nicht durch eine zentrale Intelligenz gesteuert, haben keine komplexen Algorithmen, führen keine aufwendigen Berechnungen durch, und doch lösen sie das Problem, Futter zu finden und es auf effiziente Weise zum Nest zu schaffen. Zwei wichtige Eigenschaften sind dabei Verteiltheit und Einfachheit, die auch für diese Arbeit zentral sind.

1.2 Ziel

In dieser Arbeit soll untersucht werden, ob es möglich ist, mit Ameisen multiobjective Routing zu machen. Wir haben uns auf zwei Ziele beim Finden eines Pfades festgelegt: Geringe Anzahl Hops und geringe Kosten[1]. Ein Algorithmus sollte idealerweise in der Lage sein, bei entsprechenden Einstellungen einen Pfad mit geringen Kosten gegenüber einem anderen Pfad mit weniger Hops aber mehr Kosten vorzuziehen. So sollten Pareto-Punkte von Hops und Kosten gefunden werden.

[1] Anstatt Hops und Kosten wären natürlich auch beliebige andere Metriken möglich. Wichtig ist bei dieser Arbeit lediglich die Begrenzung auf zwei.

Kapitel 2

Stand der Technik

Bevor man daran gehen kann, das Verhalten von Ameisen und anderen sozialen Insekten zu kopieren, muss deren Verhalten und die Mechanismen, die das Funktionieren des Staates garantieren, zuerst verstanden werden. *Swarm Intelligence[1]* beinhaltet eine Vielzahl Experimente mit Ameisen und Bienen und versucht diese Grundlegenden Mechanismen aufzuzeigen.

Algorithmen die von diesen Mechanismen Gebrauch machen, sind in verschiedenen Gebieten zu finden. Insbesondere gibt es auch vielversprechende Arbeiten über NP-komplexe Probleme, wie z.B. zum *traveling salesman problem[2].*

Ein grosses Forschungsgebiet ist aber das Routing in Kommunikationsnetzwerken, wie in *AntNet[3]* oder in [6], wo die Möglichkeiten zur Lastverteilung untersucht wurden. Die dabei benutzten Algorithmen funktionieren im Prinzip meist sehr ähnlich, unterscheiden sich jedoch im Detail. Fast in jeder Arbeit werden Problemspezifische Änderungen und Erweiterungen am Algorithmus vorgenommen. Das Gegenstück dazu in der Natur sind z.B. Ameisenstämme die sich an verschiedene Umgebungen angepasst haben.

Kapitel 3

Material und Methoden

3.1 Zur Arbeit benutzte Programme

Die Simulation wird in OMNeT++[4] durchgeführt, für das man eigene Module in C++ schreiben kann. Ein solches Modul beschreibt die Funktion der Knoten. Mit dem in OMNeT++ mitgelieferten *gned* kann man eigene Topologien erstellen, jedoch ist dies schon für wenige Knoten recht zeitaufwendig, insbesondere weil Links in OMNeT++ immer unidirektional sind. Das heisst man muss für jede Verbindung zwischen zwei Knoten zwei Links erstellen. Hinzu kommt die Bedingung meiner Implementierung, dass der Ein- und Ausgehende Link den gleichen Index haben müssen, in[2] und out[2] muss also mit dem gleichen Nachbar verbunden sein.

Mit BRITE[5] lassen sich zufällige Topologien generieren, die diese Bedingung erfüllen. Mit einem Zusatz zu BRITE lassen sich diese Topologien in das OMNeT++ eigene Format exportieren.

Für die automatisierte Ausführung und teilweise Auswertung benutzte ich Skripte in sh, python und zur Visualisierung gnuplot.

3.2 Designentscheidungen

Die im Folgenden beschriebenen Verfahren habe ich in meiner Implementation verwendet, die den Namen *Primant* trägt.

3.2.1 Modell

Die Ameisen bewegen sich in einem Netzwerk, in welchem die Knoten N die Orte sind, an dem sich die Ameisen entscheiden müssen, zu welchem Knoten sie weitergehen möchten. Jeder Knoten hat Verbindungen zu anderen Knoten, auch Links genannt. Ein Knoten hat eine beliebige Anzahl solcher Links $l_{1..k}$ zu seinen k Nachbarknoten. Jedem Link l_i sind Kosten c_i zugeordnet. Die Kosten eines Weges vom Quell- zum Zielknoten berechnen sich aus der Summe über die Kosten der einzelnen Links.

Wenn eine Ameise den Zielknoten findet, dann kehrt sie auf dem gleichen Weg[1] zurück zum Quellknoten und legt dabei eine Pheromonspur. Die Pheromonstärke P_i zu den entsprechenden Links i wird in den Knoten gespeichert.

3.2.2 Entscheidungsbaum und Heuristik

Eine Ameise kann sich an jedem Knoten entscheiden, ob sie exploiten oder exploren will. *Exploiten* bedeutet, dass die Ameise das vorhandene Wissen im Netzwerk, also die Pheromone,

[1] Dies ist eine Vereinfachung. Siehe dazu 3.2.5

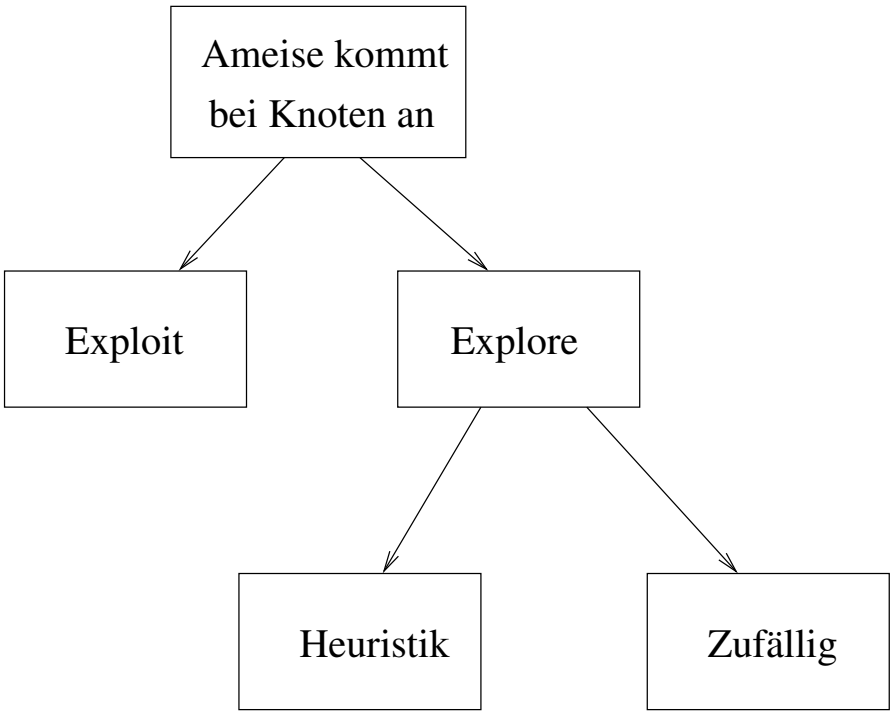

Abbildung 3.1: Entscheidungsbaum einer Ameise im Programm *Primant*

für ihre Entscheidung nutzt. *Exploren* bedeutet hingegen, dass die Ameise die Pheromone nicht für ihre Entscheidung nutzt, sondern neue Wege erforscht. Die Wahrscheinlichkeit dass die Ameise sich für explore entscheidet, wird mit dem Parameter *curiosity* eingestellt. Eine Exploiter-Ameise wird den besten bekannten Pfad nehmen, aber die schon besuchten Knoten scheiden aus. Eine Explorer-Ameise hingegen kann sich entweder zufällig mit gleicher Wahrscheinlichkeit für einen Link entscheiden, oder sie kann die Heuristik anwenden (siehe Abb. 3.1). Wie häufig die Heuristik angewandt wird, entscheidet der Parameter *useHeuristic*.

Die Heuristik wählt ebenfalls nur Links zu unbesuchten Nachbarn N_u. Die Wahrscheinlichkeit, dass die Heuristik den Link l_i, $i \in N_u$ wählt, ist

$$P(l_i) = \frac{1/c_i}{\sum_{j \in N_u} 1/c_j}$$

Das Bedeutet z.B. dass bei den Kosten 5 für Link 1, 10 für Link 2, der erste Link doppelt so häufig durch die Heuristik ausgewählt wird.

Man könnte die Heuristik auch so machen, dass sie immer den Link mit den kleinsten Kosten wählt, aber das ist unnatürlich. Zwei Links deren Kosten beinahe gleich sind, sollten von der Heuristik auch etwa gleich oft ausgewählt werden, was nicht der Fall wäre wenn man immer nur den günstigsten Link wählt.

3.2.3 Pheromonstärke und Alter

Bei kleinen Netzwerken mit z.B. 4 Knoten arbeitete mein Algorithmus auch ohne einen Zusatz, wie z.B. das Alter, sehr zuverlässig. Als ich dann allerdings Experimente mit grösseren Netzen machte, musste ich feststellen, dass die Konvergenz ungenügend ist. Um dem entgegenzuwirken habe ich jeder Ameise ein Alter gegeben (Implementiert ist dies indem bei jeder Ameise die Geburtszeit abgespeichert wird). Das Alter einer Ameise berechnet sich somit aus der vergangenen Simulationszeit seit der Geburt.

Das Alter kommt zur Wirkung, wenn die Ameise Pheromone ablegt. Dabei habe ich mich stark an die Idee in [6] angelehnt. Die Summe aller Pheromone bei einem Knoten ist 1. Wenn eine Ameise Pheromone auf den Link r ablegt, verändern sich die Pheromonstärken auf den Links i mit

$$P_i = \frac{P_i+a}{1+a} \qquad \forall i \neq r$$
$$P_i = \frac{P_i}{1+a} \qquad sonst$$

Dabei bezeichnet a die Stärke der Veränderung und nimmt mit zunehmendem Alter der Ameise ab: $a = deltaP/age$. Über $deltaP$ kann man einstellen, wie startk die Veränderung der Pheromone maximal sein soll. In meinen Simulationen war $deltaP = 0.5$.

Diese Art das Pheromon zu verändern hat die nützliche Eigenschaft, dass Ameisen, die einen Link finden, der bis jetzt wenig Pheromone hatte, stärkeren Einfluss haben als Ameisen auf einem ohnehin schon mit viel Pheromon belegten Link. Das Ziel ist hier also, das "umkippen" auf einen besseren Pfad zu begünstigen und dadurch das "einbrennen" zu verhindern.

3.2.4 Evaporation

In Primant habe ich auf Evaporation verzichtet, da sie keinen grossen Einfluss haben würde. Die Art wie die Ameisen die Pheromone verändern (siehe 3.2.3) hat schon die Eigenschaft, das "Umkippen" zu erleichtern. Evaporation hätte den selben Effekt. Der Einzige Fall wo sich die Evaporation nützlich wäre, ist, wenn auf einmal viele Ameisen an einen Knoten kommen, der noch veraltete Informationen gespeichert hat, die inzwischen durch Veränderungen im Netzwerk ungültig geworden sind. Hier würde das gesammelte Wissen durch Evaporation mit der Zeit vergehen und die Ameisen nicht auf einen falschen, weil auf alten Informationen beruhenden, Pfad führen. Ein solcher Fall kommt aber in unserer Simulation nicht vor, deshalb ist die Evaporation unnötig.

3.2.5 Loops entfernen und verhindern

Es gilt unbedingt zu verhindern, dass irgendwann ein Pfad gefunden wird, der im Kreis herum führt. Allerdings kann man nicht verhindern, dass eine Ameise beim Erforschen des Netzes irgendwann gezwungen ist, auf einen schon besuchten Knoten zu gehen. Man muss also dafür sorgen, dass diese Schlaufe wieder entfernt wird. Man kann dies entweder im Zielknoten machen, oder, wie ich es mache, bei jedem Knoten.

Es ist auch wünschenswert, dass die Ameise möglichst selten eine Schlaufe macht. Man kann dies nicht ganz verhindern[2], aber zumindest die Wahrscheinlichkeit dafür verkleinern. Die Idee ist, dass eine Ameise wenn möglich immer zu einem Knoten weitergeht, den sie noch nicht besucht hat. Wenn dies nicht möglich ist, da schon alle Nachbarknoten besucht sind, wird in *Primant* wieder zufällig entschieden und die Schlaufe danach entfernt.

Dies hat Vor- und Nachteile. Der Vorteil ist, dass Ameisen sich schneller im Netzwerk verteilen, da sie versuchen, nie dahin zurückzukehren wo sie schon einmal waren. Der Nachteil ist aber, dass es passieren kann, dass eine Ameise "gefangen" wird. Damit ist gemeint, dass sie an einem Ort ist, wo die Knoten um sie herum bereits besucht sind. Sie wird dann wie beschrieben zufällig entscheiden, wohin sie gehen will. Im Ungünstigsten Fall kann es dann passieren, dass die Ameise danach nochmals die gleiche Schlaufe läuft, da alle anderen Knoten schon besucht sind.

3.3 Implementation

Ich werde hier nur soweit auf das Programm eingehen wie es nötig ist um zu verstehen, wo man eigene Änderungen am Besten anbringen sollte. An vielen Stellen befinden sich auch

[2] Ausser natürlich man löscht die Ameise, wenn sie im Kreis gelaufen ist.

hilfreiche Kommentare im Code selber.

3.3.1 Initialisation

Die Initialisation findet in der Methode *activity()* statt. Zuerst wird der Zufallsgenerator mit Hilfe der aktuellen Zeit initialisiert. Darauf wird geprüft, ob die Anzahl Netzwerkknoten nicht zu gross ist, also nicht grösser als *maxnodes* ist. Falls man an diese Grenze stossen sollte, kann man *maxnodes* ohne weiteres am Anfang der Datei vergrössern. Die Pheromontabellen der einzelnen Knoten werden auf `1 / gatesize` initialisiert, es sind also zu Beginn alle Routen gleich gut.

Im Abschnitt `Get neighbour nodes` wird die Datenstruktur *remoteModuleId[]* gefüllt. Sie enthält zu jedem lokalen Gate die ID des verbundenen Netzwerkknoten. Dabei wird auch geprüft, ob das jeweilige in- und outputgate zum selben Knoten verbunden ist, und gegebenenfalls wird eine Fehlermeldung erzeugt.

Um die eigentliche Simulation zu beginnen, wird im im nächsten Abschnitt (`Send first birth-message`) eine Message vom Typ *KIND_BIRTH* verschickt, was zu Geburten von Ameisen in regelmässigen Abständen führt (Methode *birth()*). Es wird auch eine Ameise vom Typ *KIND_ROUTEBIRTH* verschickt, die zur Geburt von Ameisen führt, welche regelmässig die momentan beste Route erkunden.

3.3.2 Simulation

In der grossen `for(;;)` Schlaufe werden die Simulationsereignisse behandelt. Zudem ist ein '*Special event*' möglich, der gedacht ist, um während der Simulation das Netzwerk zu verändern, also z.B. einen Knoten ausfallen zu lassen.

Die eigentliche Behandlung der Events beginnt mit dem *switch(kind)*, der fünf Verschiedene Typen unterscheidet: Forward-Ants, Backward-Ants, Geburts-Messages für normale und *KIND_BESTROUTE*-Ameisen, und die *KIND_BESTROUTE*-Ameisen selbst. Im Wesentlichen wird bei Forward-Ants die Ameise mit *sendexplore()* oder *sendexploit()* weitergeschickt, oder mit *returnAnt()* auf ihren Heimweg geschickt. Bei Backward-Ants wird *sendbackward()* benutzt, falls die Ameise ihr Ziel noch nicht erreicht hat. Bei Geburts-Messages wird *birth()* aufgerufen, und wieder eine Geburts-Message generiert. Dies führt zu Ameisengeburten im Abstand von *birthrate*, das in der ini-Datei eingestellt werden kann. Die *KIND_BESTROUTE*-Ameisen werden der Methode *sendBestRoute(rant)*[3] übergeben, falls sie ihr Ziel noch nicht erreicht haben. Falls sie schon am Ziel sind, so wird ein Eintrag ins Logfile geschrieben.

3.3.3 Methode *sendexplore*

Diese Methode sendet die Ameise als Explorerameise weiter. Das heisst sie soll wenn möglich zu einem beliebigen, noch nicht besuchten Nachbar weitergehen.

Zunächst wird unser Knoten zu der Route der Ameise hinzugefügt, da sie uns ja gerade besucht. Danach wird untersucht, welche Nachbarknoten die Ameise schon besucht hat. Nun haben wir also in *visited[]*, welche Nachbarn schon besucht wurden. `visited[2] = 1` bedeutet also, dass der Nachbarknoten an Gate 2 schon besucht wurde. In *nrVisited* wird gezählt, wieviele Nachbarn schon besucht wurden.

Als Nächstes wird ein beliebiger der noch nicht besuchten Nachbarn ausgesucht, oder, falls schon alle besucht wurden, ein beliebiger Nachbar. Am Ende wird nun die Ameise noch gesendet.

[3] die Funktion *sendBestRoute(rant)* tut im Wesentlichen nichts anderes als immer Exploiten, so dass die Ameise auf dem momentan besten bekannten Pfad zum Ziel kommt

3.3.4 Methode *sendexploit*

Diese Methode sendet die Ameise als Exploiterameise weiter. Das heisst sie soll den Pfad mit der stärksten Pheromonspur wählen, aber zu einem Knoten den sie noch nicht besucht hat.

Zuerst tut diese Methode dasselbe wie *sendexplore()*, unser Knoten wird zur Route hinzugefügt und wir suchen unbesuchte Nachbarn. Wenn schon alle Nachbarknoten besucht wurden, dann wird die Ameise als Explorer weitergeschickt, d.h. *sendexplore()* wird aufgerufen. Dadurch wird die Gefahr, dass die Ameise nochmals die selbe Schlaufe läuft, geringer. Der Weg mit der stärksten Pheromonspur ist mit grosser Wahrscheinlichkeit der Weg, den sie schon vorher gegangen ist, und der war ja offensichtlich nicht der Beste, wenn sie nun einmal im Kreis gelaufen ist.

Wenn es noch unbesuchte Nachbarn gibt, so wird derjenige mit der stärksten Pheromonspur gewählt, und die Ameise wird verschickt.

3.3.5 Methode *sendbackward*

Diese Methode wird aufgerufen, wenn eine Ameise auf dem Rückweg ist.

Das letzte Element der Route wird entfernt, da dies den Knoten beschreibt, in dem sich die Ameise gerade befindet. Dann wird das Gate gesucht, an dem sich der Knoten befindet der nun als letztes in der Route steht, und die Ameise wird verschickt.

3.3.6 Veränderung der Pheromone mit *pheroUpdate()*

Diese Methode wird aus *main()* aufgerufen, wenn eine Backward-Ant ankommt. Zu Beginn meiner Arbeit wurde hier die Pheromonstärke einfach um 1 erhöht. In der Endversion wird hier aber ein Pheromonupdate in Abhängigkeit des Alters durchgeführt (siehe S.6).

3.3.7 Weitere Methoden der Klasse *node*

Die Methode ***birth()*** erzeugt eine neue Ameise und schickt sie an den eigenen Knoten (ohne Zeitverzögerung). Damit wird erreicht, dass die neue Ameise gleich behandelt wird wie wenn sie schon unterwegs wäre. Die Ameise ist dann zwar im ersten Schritt schon im Kreis gelaufen (Vom Startknoten wieder zum Startknoten), aber diese Schlaufe wird durch *removeLoop* gleich wieder entfernt.

returnAnt() wird von *main* aufgerufen, wenn eine Ameise ihr Ziel erreicht hat. Quelle und Ziel der Ameise werden nun vertauscht und der Typ ändert sich zu *KIND_BACKWARD*. Dann wird sie dorthin zurückgeschickt, wo sie hergekommen ist.

Die Methode ***isNeighbour()*** wird an vielen Orten gebraucht. Sie ergibt -1 wenn ihr eine Module-ID (also die ID eines Knotens) übergeben wird, zu der kein direkter Link existiert. Wenn ein Link existiert, dann gibt die Methode die Nummer des Gates zurück, das zu dem Modul führt.

Kapitel 4

Resultate und Diskussion

4.1 Unerwünschte Effekte und Schwierigkeiten

Eine naheliegende Messung eines Ameisennetzwerks besteht darin, zu messen, ab wann die Ameisen den schnellsten Pfad gefunden haben und dabei bleiben. Doch hier ist Vorsicht geboten. Man stelle sich vor, dass zu Beginn der Simulation eine grosse Menge Ameisen auf die Suche nach dem Zielknoten geht. Die Ameisen, die den kürzesten Pfad finden, werden als erstes zurückkommen. Zumindest während einer gewissen Zeit (bis die Ameisen, die einen längeren Weg gegangen sind, zurückkehren) wird nun der kürzeste Pfad am meisten Pheromone haben. Wenn die Exploitation nun genug hoch ist, wird dieser Weg verstärkt und die Ameisen werden diesen Pfad sehr schnell mit viel Pheromon kennzeichnen.

Das Unschöne an dieser schnellen Konvergenz ist offensichtlich dass sie so nur beim initialisieren des Netzwerks funktioniert. Der Effekt beruht darauf dass die zuerst zurückkehrenden Ameisen schon soviel Pheromone legen dass quasi schon alles entschieden ist. Startet man weniger Ameisen, so wird es möglich (d.h. wahrscheinlicher) dass die ersten paar Ameisen nicht den besten Weg gehen und trotzdem früher zurück sind als die erste Ameise die den besten Weg wählt. Der längere Pfad wird dann verstärkt und die Ameisen können sich auf einen falschen, d.h. nicht den kürzesten Weg einstellen. Man nennt dies auch oft "einbrennen", da die Ameisen von diesem Weg nicht mehr wegkommen.

Der Effekt kann auch so verstanden werden, dass die Ameisen im Prinzip ein Flooding machen, da innert kurzer Zeit sehr viele Ameisen gestartet werden. Die Erste, die zurückkehrt, hat, vereinfacht gesagt, den kürzesten Pfad gefunden.

Dieser Effekt ist unabhängig von Metriken wie z.B. Kosten, es kommt einzig darauf an, welche Ameise zuerst zurückkehrt, also welcher Pfad den kleinsten Delay hat. In Primant bedeutet dies, dass der Weg mit der kleinsten Anzahl Hops gefunden wird, da alle Links den gleichen Delay aufweisen.

4.2 Aussagekräftige Evaluation

Die Evaluation stellt ein nicht zu unterschätzendes Problem dar. Es gibt viele Parameter die man einstellen kann und deren Auswirkungen nicht ohne weiteres einleuchten. Möchte man zudem das Zusammenwirken von Parametern verstehen, also was passiert wenn mehrere Parameter gleichzeitig verändert werden, so ist ein vielfaches mehr an Aufwand nötig. Ich habe mich deshalb auf die Analyse der Auswirkungen bei der Veränderung von einzelnen Parametern konzentriert.

Ausserdem muss man wenn möglich immer mehrere Versuche mit den selben Einstellungen machen, da sich der Algorithmus von Mal zu Mal anders verhalten kann. Eine Automation der Experimente ist daher von grossem Vorteil. Desweiteren müssten die Versuche auch auf

verschiedenen Netzwerken durchgeführt werden, da auch Topologien einen Einfluss auf die Resultate haben. Ich habe meine Versuche aber meist nur auf einem Netzwerk durchgeführt, da dieser Vorgang schwierig zu automatisieren ist. Ich habe dem zumindest teilweise Rechnung getragen indem ich die Start- und Zielknoten variiert habe.

Die Daten die man aus einer Simulation erhält sind erstens die Routen die jede Einzelne Ameise gegangen ist, und zweitens die beste Route die zu jedem Zeitpunkt im Netzwerk existierte.

Zum Schluss muss man noch einen Weg finden, die gewonnenen Daten so darzustellen, dass man eine Vermutung bestätigen oder widerlegen kann. Dies führte zu vielen verschiedenen Darstellungsformen, viele davon hatten leider nicht die erwartete Aussagekraft.

4.3 Sofortiges Entfernen der Loops

Wie in Kapitel 3.2.5 beschrieben, kann das sofortige Entfernen von Loops dazu führen, dass Ameisen "gefangen" werden. Um zu überprüfen, ob dies tatsächlich geschieht, habe ich die Simulation so modifiziert[1], dass nach 1000 Zeitschritten keine Ameisen mehr geboren werden. Ich habe das Experiment auf einem Netzwerk mit 100 Knoten gemacht[2], die Einstellungen waren useHeuristic=0.2; curiosity=0.8; deltaP[3]=0.5.

Die Simulationszeit hatte ich auf 1500 beschränkt, jedoch waren alle Simulationen vorher fertig, weil keine Ameisen mehr vorhanden waren. Das bedeutet also dass jede Ameise irgendwie zum Ziel kam, keine blieb "gefangen". Die Ameisen kamen meistens schon alle vor der Simulationszeit 1200 zum Ziel.

Die Routen der letzten Ameisen, die nach der Simulationszeit 1100 ankommen, haben etwa 30 bis 40 Hops, sie benötigten dafür aber etwa 100 bis 200 Zeitschritte. Das bedeutet dass diese Ameisen etwa 60 bis 170 Zeitschritte damit verbracht haben in Schlaufen zu laufen. Das sieht auf den ersten Augenblick nach recht viel aus, aber wenn man bedenkt dass nach der Simulationszeit 1100 nur noch etwa 5 Ameisen ankommen, dann scheint es doch kein grosses Problem zu sein. Der einzige Effekt ist ein etwas erhöhter Verkehr durch die Ameisen die "hängen bleiben". Doch auch die darf man in Kauf nehmen, wenn der Grossteil der Ameisen ihr Ziel dafür schneller findet. Ob dies aber tatsächlich der Fall ist, kann ich nicht beweisen, da hierfür Vergleichsdaten fehlen.

4.4 Einfluss der Heuristik

Der Wunsch am Anfang der Arbeit war, über die Häufigkeit der Anwendung der Heuristik einstellen zu können, wie wichtig die Kosten gegenüber den Hops auf einem Pfad sind. Wird die Heuristik häufig angewandt, so sollten die Kosten sinken und dafür mehr Hops in Kauf genommen werden. In Abb. 4.1 sind alle Routen aufgetragen, die während zwei Experimenten gefunden wurden. Jeder Punkt entspricht der Route einer Ameise. In der Darstellung kann man erkennen, dass der obere linke Rand der Punktewolke vor allem Routen enthält, die aus der Simulation mit wenig Heuristik stammen, wobei die Punkte am Rand unten rechts vor allem aus der Simulation mit starker Heuristik stammen. Das bedeutet also, dass durch die Anwendung der Heuristik im Allgemeinen tatsächlich Routen mit kleineren Kosten gefunden werden.

Deutlich wird dies vor allem bei den Routen im Bereich 30 bis 40 Hops, wo sehr viele verschiedene Routen möglich sind. Bei den kürzeren Routen wird der Effekt nicht sichtbar.

[1] Durch ein einfaches `if(simTime()>1000) continue;` bei der Behandlung der Ameisen vom Typ *KIND_BIRTH* und *KIND_BESTROUTE*

[2] Auf dem Netzwerk *100lowconn*

[3] deltaP bestimmt die Stärke der Veränderung der Pheromone, siehe 3.2.3

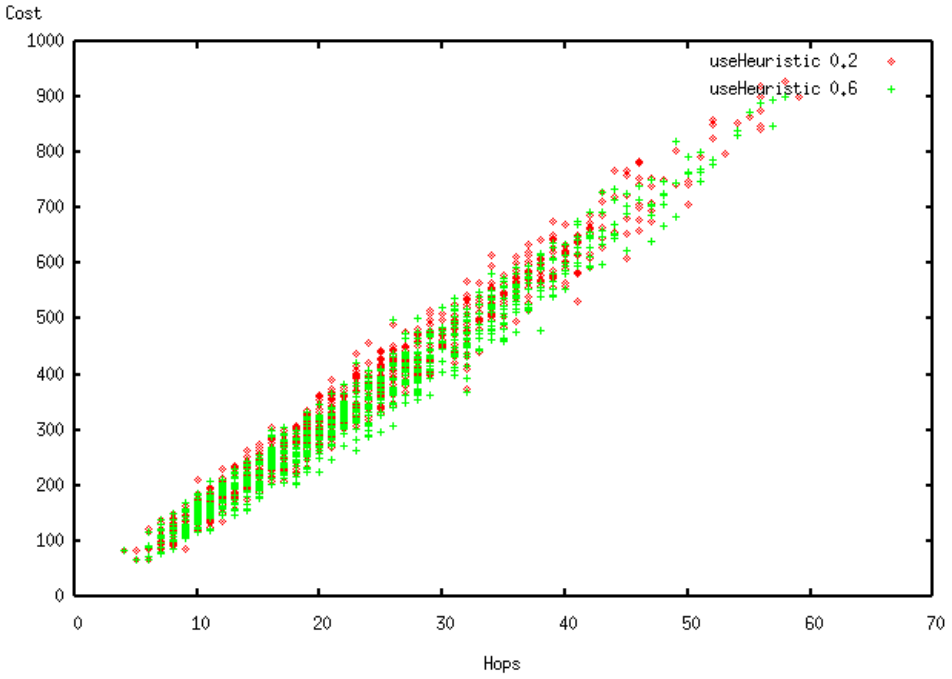

Abbildung 4.1: Resultat aus zwei Versuchen, bei denen die Heuristik verschieden Oft angewandt wurde. Wenn die Heuristik häufig angewandt wird, liegen die gefundenen Routen tendenziell bei tieferen Kosten.

Die für uns wichtigen Punkte liegen aber ganz unten links, bei tiefen Kosten bzw. wenigen Hops, wo Abb. 4.1 wenig aussagt. In diesem Bereich ist vor allem interessant, welche Route von den Ameisen als die Beste gefunden wird. Die Abb. 4.2 und 4.3 zeigen zwei Experimente, an denen man den Einfluss der Heuristik sehen kann[4]. Wird die Heuristik stark gewichtet, so ergeben sich häufig Instabilitäten. Interessant ist in Abb. 4.3 vor allem dass das System nicht wie erwartet einen Pfad mit weniger Kosten und mehr Hops findet, sondern dass sich beides verschlechtert; das System kippt bei der 500sten Ameise zu einem schlechteren Pfad um.

Bei der 900sten Ameise werden die Kosten nochmals höher, wobei die Anzahl Hops gleich bleibt. Die Heuristik scheint hier offensichtlich nicht das zu bewirken was wir erwarten. Der Grund dafür liegt meiner Meinung darin, dass die Ameisen nur lokal optimieren, das heisst sie bevorzugen einen Link aufgrund seiner Kosten, auch wenn er aus topologischer Sicht sehr schlecht ist. Wenn also eine Ameise einen Link mit kleinen Kosten begeht, dann kann diese Entscheidung im Nachhinein noch grosse Kosten verursachen. Dazu kommt noch, dass die Kosten sich nicht verändern, das heisst ein gewisser Teil der Ameisen wird immer auf diesen verlockenden Link gehen, auch wenn es darauf sehr wenige Pheromone hat.

[4]Ich habe zwei Experimente ausgesucht deren Verlauf einigermassen typisch ist für diese Einstellungen. Allerdings ist es vor allem bei den instabilen Versuchen wie in Abb. 4.3 schwierig zu sagen, was typisch ist. Auch bei diesen Einstellungen konvergiert das System manchmal schnell und bleibt stabil. Die zwei Beispiele dienen nur als Illustration für meine Erfahrungen die ich bei vielen Experimenten gemacht habe.

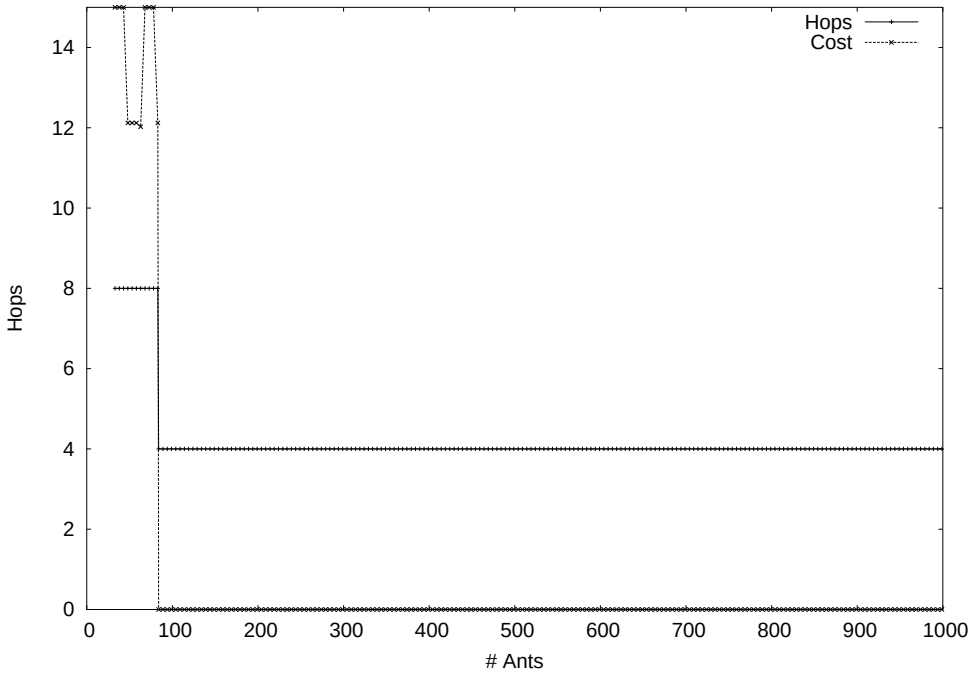

Abbildung 4.2: Die Entwicklung von Hops und Kosten der besten Route in Abhängigkeit der Zeit. Hier für *useHeuristic = 0.2*

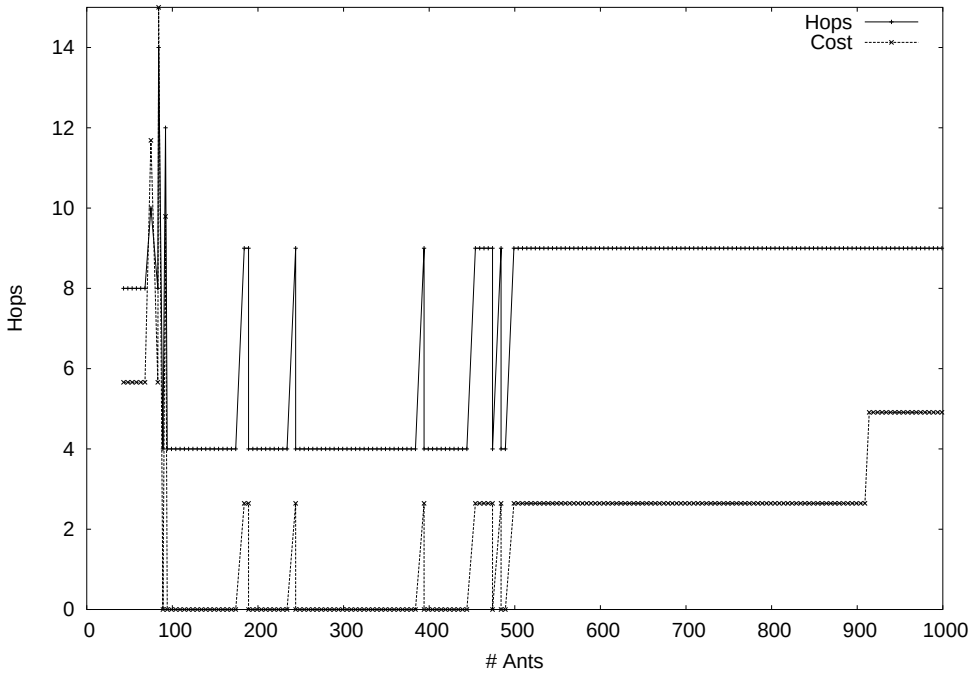

Abbildung 4.3: Die Entwicklung von Hops und Kosten der besten Route in Abhängigkeit der Zeit. Hier für *useHeuristic = 0.6*

Kapitel 5

Ausblick

Ameisenbasierte Algorithmen sind ein interessantes Forschungsgebiet und haben auf den ersten Blick vielversprechende Eigenschaften, die wir auch auf Routingalgorithmen für Computernetzwerke übertragen möchten.

Ein grosser Unterschied zu konventionellen Algorithmen besteht darin, dass Ameisenalgorithmen nicht von Menschen erfunden worden sind, sondern der Natur abgeschaut. In der Natur sind diese Algorithmen durch Evolution lange optimiert worden, das heisst die Algorithmen an sich sind gut (das ist schon dadurch bewiesen dass es noch Ameisen gibt). Wenn wir dies kopieren wollen, dann entstehen Schwierigkeiten, weil wir in der Natur den Algorithmus, der für unser Problem optimiert ist, nicht finden. Auch die Ameisenstämme funktionieren nicht alle gleich. So gibt es auch Ameisen, die sich tatsächlich auf einen Weg "einbrennen" und einen später angebotenen besseren Weg nicht annehmen [7].

Die Herausforderung bei der Arbeit mit Ameisenalgorithmen ist, alle Parameter richtig zu wählen, die das Verhalten der Kolonie am Ende bestimmen. Leider kann man die Auswirkungen von vielen Parametern nur schwer abschätzen, man muss also Simulieren, um zu sehen, ob man den gewünschten Effekt erzielt, so wie ich es in dieser Arbeit getan habe. Die Evolution hat die Ameisen schon lange optimiert und passt sie ständig an. Wenn wir diese Algorithmen benützen wollen, müssen wir aber wieder fast bei Null anfangen.

Wir müssen unseren Algorithmus genau auf unsere Problemstellung anpassen, so wie auch die Ameisen ihrer Umgebung angepasst sind. Ich denke deshalb nicht dass es die "Ameise für Alles" gibt, sondern dass man jede Problemstellung einzeln betrachten und sorgfältig einen Algorithmus mit den gewünschten Eigenschaften entwickeln muss. Auch bei meiner Arbeit waren die Bedingungen vielleicht zu wenig genau festgelegt. Es würde helfen, wenn man die ungefähre Grösse des Netzwerkes wüsste, und auch wie stark die Knoten untereinander verbunden sind[1]. Die Ameisenalgorithmen haben also durchaus ein grosses Potential, müssen aber sorgfältig auf ein Problem angepasst werden.

Im Punkt Multiobjective Routeing wurden die Erwartungen durch meinen Algorithmus leider nicht Erfüllt. Trotzdem kann man einen Einfluss auf die Routen sehen, der in die richtige Richtung geht (Abb. 4.1). Es wird jetzt also darum gehen, die Instabilitäten zu vermeiden, die bei mir stets durch die Heuristik entstanden.

[1] Natürlich ging es bei dieser Arbeit eher darum, Erfahrungen auf diesem Gebiet zu sammeln, nicht um eine konkrete Anwendung. In diesem Sinne hatte ich auch Erfolg.

Anhang A

Aufgabenstellung

A.1 Introduction

Complex systems often exhibit patterns, structures and behavior that can not be explained satisfactorily by referring to the individual components the system is composed of, or to the interactions between these individual components. The global perception of the system clearly exceeds the understanding of the local interactions and characteristics of the system's components. We may state it with Aristoteles: *the whole is greater than the sum of its parts.* Complex systems of particular interest for us are such revealing macro–behavior that is unpredictable from the behavior of the individual components (micro–behavior) and vice versa. Examples of such systems are gases (2^{nd} law of thermodynamics), lasers and some populations of social insects like ants, bees and termites. Common to all is, that at a certain point a transition from uncorrelated behavior to self-organization and order is taking place. Though the reasons and processes for the respective transitions are not yet fully understood, reinforcement, positive and negative feedback loops, cooperation, competition, and in particular the way information flows within the system, are the ingredients that have so far been identified as responsible for the transitions.

Our motivation to investigate complex system behavior emerges from a new situation in computing networks. There we face a growth of bandwidth and computer performance by factors of $1.5 - 2$ every year. In parallel to performance and bandwidth growth, systems become more and more complex. Complexity has been recognized as the challenge the IT world has to cope with, if it wants to continue to be as successful as it was the last decades.
This semester thesis investigates complex system behavior by studying the concepts ants use for finding the shortest paths between their nest and food locations. We embed the study in a technical context by using artificial ants for routing in an Internet network. Our main interest is set on the information flow generated by the system, on self–organization and self–adaption mechanisms that provide the robustness of the system, and on the evaluation of different designs and implementations of artificial ants.

Keywords self–organization, self–adaption, emergence, feedback mechanisms, information flow, evaluation

A.2 How Ants find the Shortest Path

Populations of social insects (like ants, bees, termites), are little sophisticated. Nevertheless they manage to solve complex problems like, e.g. the construction of nests, the regulation of temperature with deviations of less than 1 C^o, and the coordination of activities. They do this

without being guided by some *global* entity like a queen, without help of any proactive pacemakers (e.g. some intelligent individuals triggering and coordinating actions), and without following any internal biological plan. The complex behavior simply *emerges* from the *local* interactions of a large amount of individuals. In many cases the *emergence* is complemented by concepts of *stigmergy*. Stigmergy is an indirect communication over the environment. Individuals interpret and modify the environment according to specific stimuli. This again triggers the behavior of other individuals, which respond with additional modifications of the environment.
Two kinds of stigmergy are distinguished, such that changes the physical characteristics of the environment, as found by termites (sematectonic stigmergy) and sign–based stigmergy. The later is used by ants and bases on pheromones, a kind of volatile hormones. Pheromones enable ants to modify the environment in a way that is again understood by them. Stigmergy joined by some rules of behavior permits them to find the shortest paths between food resources and their nest.
Figure A.1 sketches how two ants proceed in finding food. While they are foraging (looking

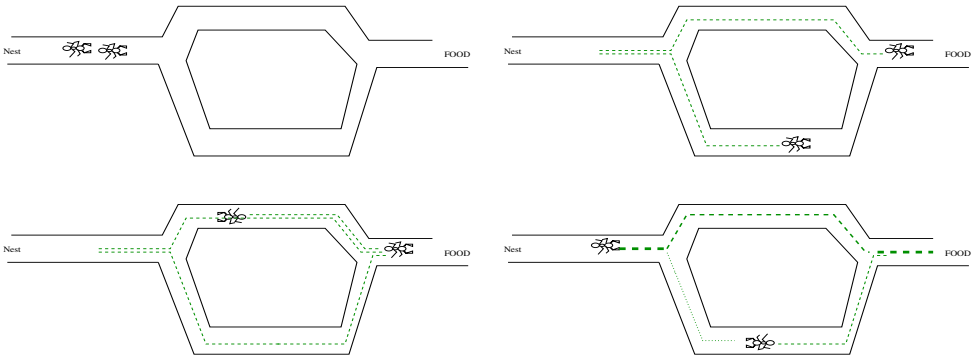

Abbildung A.1: Foraging ant finding the shortest path between nest and a food resource.

for food), they drop a constant amount of pheromone. In the first picture both ants arrive at the junction. Since there is no pheromone there yet, the decision on what direction to take is not biased, i.e. each direction has the same probability to be selected. Assuming each ant selects a different way, the one taking the shortest discovers the food resource first. We assume furthermore, that our ants take the same way as they arrived to go back to the nest. The ant that took the shortest path will therefore arrive first to the nest. Ants that start now foraging will, upon arrival at the first junction, be biased to take the way which has more pheromone, i.e. the probability to take the shortest path is higher. Two main concepts apply here: one is that decisions are taken probabilistically with a ratio of exploiting the currently best solution and exploring new solutions (adding some noise to the system). The second concept is based on feedback and is termed *reinforcement learning*. Reinforcement learning states that the shortest path is getting stronger and stronger, since more ants chose it,and doing so, continue to make it more and more attractive. In order to make this concept more effective, old information (e.g. the longer path), needs to become obsolete. Pheromone has a characteristic that amplifies this effect: it evaporates.
The information kept in the system flows towards a state of equilibrium. The information flow is supported by the evaporation of pheromone, by the feedback mechanisms of reinforcement learning, and by the ratio of exploration of new solutions and exploitation of currently best solutions.

A.3 Task Description

The semester thesis consists of three main tasks:

- design and development of several ant algorithms,
- implementation of the ant algorithms in an extensible simulation tool that allows comparison among the different ant algorithms and with other algorithms like, e.g. flooding, OSPF etc. (the simulation tool will be OMNeT++),
- evaluation of algorithms and derivation of conclusions.

A.3.1 Design and Development of Ant Algorithms

The student is expected to design and develop several algorithms for locating resources in a network. Note the term resource is used in a generic way; it may specify a host, a network, a service etc.. Although, at a first sight, the algorithms tackle down similar problems as do routing protocols like OSPF and Distance Vector Routing, their approach and objectives follow a different paradigm: they focus on simplicity as it is our approach of opposing complexity. Issues of particular interest are the transitions from uncoordinated system behavior to organized system behavior, the effects of reinforcement learning and how they influence the ability of the system to adapt to changing environmental conditions. We expect that our ant algorithms, since they follow an integral approach that affects the whole system, will be applicable in different contexts, be it for routing, for system management, for resource discovery, for load balancing etc..
The design and development of ant algorithms starts with an intense study of currently available literature on that topic. A thesis on a similar subject and some selective papers will be provided by the supervisors. Nevertheless it is expected that the student is looking for additional literature himself (preferably on the Internet).
The requirements on the ant algorithms are that they should be *simple*, decentralized, distributed, that there is no global entity involved, i.e. that no global view on the network, nor global information is kept somewhere and that there is therefore no single point of failure. The algorithms need neither to be efficient what concerns the performance consumption (CPU), nor bandwidth consumption, since we assume that there is enough of both and since our primary goal is to cope with complexity.
Since the algorithms are expected to give insights on emergence, self–adaption (robustness), on self–organization, and the flow of information in complex systems, the design and development of the ant algorithms should provide means to extract these insights.

A.3.2 Implementation in OMNet++

The implementation of the previously designed ant algorithms will be taken out with the OMNet++ network simulator. The ant algorithms will be implemented as C++ modules. The student is allowed/encouraged to complement his implementation with work already provided by other research groups under the condition, however, that this work is freely available, that it is marked as such, and that tribute is given to the respective persons.
A particular challenge for the simulation is the stochastic nature of the ant algorithms. The simulations should therefore be designed to be fast so that the inferences on the evaluation are statistically expressive.

A.3.3 Evaluation

The evaluation of the ant algorithms is a central part of the thesis and should therefore be taken out carefully. The purpose of the evaluation is to compare the ant algorithms against

themselves and with other algorithms. A task of the student consists in finding applications and scenarios suitable for these comparisons. Metrics of interest will be simplicity (the term will need to be put in a context), speed of system transition (convergence), adaptability (changing situations like topology changes and node failures), robustness, information flow etc.
From the evaluation results conclusions should be drawn and potential application fields be identified.

A.4 Organization of Work (kept in German)

- Mit den Betreuern sind wöchentliche Sitzungen zu vereinbaren. In diesen Sitzungen soll der Student mündlich über den Fortgang der Arbeit und die Einhaltung des Zeitplans berichten und anstehende Probleme diskutieren. Es ist demnach wichtig, dass sich der Student auf die Sitzungen vorbereitet.

- Am Ende der zweiten Woche ist ein Zeitplan für den Ablauf der Arbeit vorzulegen und mit dem Betreuer abzustimmen. Der Zeitplan soll wichtige Meilensteine der Arbeit identifizieren. Bei Meilensteine handelt es sich um Teilziele, die zugleich Entscheidungen über das weitere Vorgehen mit sich ziehen.

- Am Ende des ersten Monates muss eine Vorabversion des Inhaltsverzeichnisses zur Dokumentation den Betreuern abgegeben und mit diesem besprochen werden.

- Die Dokumentation soll parallel zur Arbeit geführt werden. Die Dokumentation kann mit beliebigen Textverarbeitungsprogramm erstellt werden. Empfohlen sei jedoch LaTeX.

- Nach der Hälfte der Arbeitsdauer soll ein kurzer mündlicher Zwischenbericht abgegeben werden, der über den Stand der Arbeit Auskunft gibt. Dieser Zwischenbericht besteht aus einer viertelstündigen, mündlichen Darlegung der bisherigen Schritte und des weiteren Vorgehens gegenüber Professor Plattner. Dieser Bericht soll vorher bereits mit den Betreuern besprochen werden.

- Am Schluss der Arbeit muss eine Präsentation von 15 Minuten im Fachgruppen- oder Institutsrahmen gegeben werden. Anschliessend an die Schlusspräsentation soll die Arbeit Interessierten praktisch vorgeführt werden.

- Bereits vorhandene Software kann übernommen werden und gegebenenfalls angepasst werden.

- Es ist ein Schlussbericht über die geleisteten Arbeiten abzuliefern (4 Exemplare). Der Bericht ist in Deutsch oder Englisch zu halten. Der Abstract und die Zusammenfassung (sog. Executive Summary), müssen sowohl auf deutsch und englisch geschrieben sein. Im Appendix soll die Aufgabenstellung als auch der Zeitplan gegeben sein.

- Die Arbeit muss auf CDROM archiviert abgegeben werden (inkl. Unterlagen der Präsentation). Stellen Sie sicher, dass alle Programme sowie Dokumentation sowohl in der laufähigen, resp. druckbaren Version als auch im Quellenformat vorhanden, lesbar und verwendbar sind.

Anhang B

Zeitplan

Woche	Arbeit
43	Zieldefinition
44	Literaturrecherche, Zeitplan aufstellen
45	
46	Einarbeiten in OMNeT++, Installation
47	
48	Design neuer Algorithmen
49	
50	Implementation, Zwischenbesprechung
51	
52	Evaluation, Dokumentation
1	
2	
3	
4	Reserve
5	Abgabe, Präsentation

Literaturverzeichnis

[1] Swarm Intelligence. From Natural to Artificial Systems; Eric Bonabeau, Marco Doriga, Guy Theraulaz; Oxford Univerity Press; 1999

[2] Ant colonies for the traveling salesman problem; Marco Dorigo, Luca Maria Gambardella; Université Libre de Bruxelles; 1996.

[3] AntNet: A Mobile Agents Approach to Adaptive Routing; Technical Report 97-12, IRIDIA, Université Libre de Bruxelles; 1997

[4] OMNeT++, a discrete event simulation environment;
URL: http://www.omnetpp.org (18.2.2004)

[5] BRITE – Boston university Representative Internet Topology Generator;
URL: http://www.cs.bu.edu/brite/ (18.2.2004)

[6] Ant-like agents for load balancing in telecommunication networks; Schoonderwoerd, O. Holland and J. Bruten; ACM Press, 1997

[7] Staatsbildende Insekten als Vorbilder für Software-Agenten; Beckert, A.; 1997.
URL: http://fsinfo.cs.uni-sb.de/~abe/w5/Bionik/ACO-Seminar-Vortrag.pdf (18.2.2004)