

# **Ameisenalgorithmen und ihre Anwendung auf das RCPSP**

**Seminarausarbeitung**

Sven Albrecht

8. Februar 2008

### Vorbemerkung

Die vorliegende Seminararbeit entstand im Rahmen des Seminars „Complex Scheduling“ im Wintersemester 2007 / 2008 an der Universität Osnabrück. Es werden zunächst Ameisenalgorithmen im Allgemeinen vorgestellt und anschließend wird gezeigt, wie sich die Idee der Ameisenalgorithmen auf die Problemstellung des RCPSP übertragen läßt. Grundkenntnisse aus dem Bereich des „Complex Scheduling“ werden vorausgesetzt. Insbesondere sollte der Leser mit dem RCPSP und seiner Terminologie vertraut sein. Um gegebenenfalls Kenntnisse aufzufrischen wird an dieser Stelle auf Brucker und Knust [2] verwiesen.

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>4</b>
1.1	Biologischer Hintergrund . . . . .	4
<b>2</b>	<b>Ant Colony Metaheuristik</b>	<b>6</b>
2.1	Grundidee . . . . .	6
2.2	Formalisierung des Problems . . . . .	7
2.3	Generelle Eigenschaften von ACO . . . . .	8
<b>3</b>	<b>Anwendung auf das RCPSP</b>	<b>10</b>
3.1	Entscheidungen für Ameisen . . . . .	11
3.2	Pheromonupdate . . . . .	12
3.3	Heuristiken . . . . .	13
3.4	Optionale Modifikationen . . . . .	14
3.4.1	Kombination von direkter und summierter Auswertung . . . . .	15
3.4.2	Parameteränderung während der Laufzeit . . . . .	15
3.4.3	Entkommen aus lokalen Minima . . . . .	16
3.4.4	Lokale Suche . . . . .	16
3.4.5	Bidirektionale Planung . . . . .	16
<b>4</b>	<b>Ergebnisse und Diskussion</b>	<b>17</b>
4.1	Ergebnisse . . . . .	17
4.2	Diskussion . . . . .	19
	<b>Literatur</b>	<b>20</b>

# 1 Einleitung

Unter dem Begriff „Ameisenalgorithmen“ wurden Metaheuristiken zur Lösung komplexer Probleme zusammengefasst. Die Grundidee, die zur Lösungsgenerierung ist vom Verhalten natürlicher Ameisen inspiriert, daher rührt auch der Name der Metaheuristiken. Die ersten Ameisenalgorithmen wurden von Marco Dorigo 1992 in seiner Ph.D. Thesis „Optimization, Learning and Natural Algorithms“ formuliert, in der er Strategien von einigen Ameisenarten bei der Futterbeschaffung als Algorithmus formalisierte und auf andere Probleme übertrug.

Die Ausarbeitung ist in insgesamt vier Abschnitte aufgeteilt. Der folgende Abschnitt erläutert die biologische Motivation, die hinter den Prinzipien von Ameisenalgorithmen steht. In Kapitel 2 wird eine allgemeine Formulierung von Ameisenalgorithmen vorgestellt. Kapitel 3 zeigt eine Möglichkeit, wie man die in Kapitel 2 gezeigten Formulierungen auf das RCPSP übertragen kann und an welchen Stellen sich Unterschiede ergeben. Abschließend wird in Kapitel 4 eine Überblick zu den Ergebnissen und eine Diskussion präsentiert.

## 1.1 Biologischer Hintergrund

Bestimmte Ameisenarten orientieren sich bei der Futtersuche anhand von chemischen Botenstoffen, den sogenannten *Pheromonen*. Pheromone werden von einzelnen Ameisen während ihrer Fortbewegung auf dem Boden abgesondert. Andere Ameisen können die hinterlegten Pheromone wahrnehmen und präferieren Wege, auf denen viele Pheromone abgelagert sind, gegenüber Wegen, die über weniger oder keine Pheromoninformationen verfügen. In einigen Experimenten [7] an echten Ameisen wurde der Nutzen dieser Art von Kommunikation demonstriert: Eine Ameisenkolonie wurde mit einer Futterquelle über verschiedene, unterschiedlich lange Wege verbunden. Werden zunächst alle Wege, die Kolonie und Futterquelle verbinden, von Ameisen frequentiert, lässt sich nach einer gewissen Explorationszeit eine deutliche Präferenz des kürzesten Weges erkennen. Dieses Verhalten wird durch die Pheromone gesteuert. Brechen Ameisen zeitgleich auf unterschiedlich langen Wegen zur Futterquelle auf und sind noch keine Pheromonspuren vorhanden, so kommt die Ameise, die den kürzeren Weg zur Futterquelle zurücklegen musste, zuerst dort an. Die Strecke, auf der die Ameise zur Futterquelle gekommen ist, wird gegenüber anderen Wegen präferiert, da sich auf diesem Weg schon Pheromone befinden. Andere Ameisen, die später bei der Futterquelle eintreffen, werden auch tendenziell diesen Rückweg bevorzugen, da die Pheromonablagerungen auf diesem Weg stärker sind, als die Ablagerungen auf den anderen Wegen. Zusätzlich können auf dem kürzesten Pfad innerhalb des gleichen Zeitraums mehr Ameisen zur Futterquelle und zum Nest zurück laufen, als auf den längeren Pfaden. Somit können auf dem kürzeren Weg mehr Pheromone abgelagert werden, als auf den längeren Wegen. Die stärkeren Pheromonablagerungen sorgen schließlich dafür, dass weitere, von der Kolonie aufbrechende, Ameisen mit einer höheren Wahrscheinlichkeit diesen günstigsten Pfad wählen. Verstärkt wird dieser Effekt zusätzlich durch die Evaporation der Pheromone im Laufe der Zeit. Wird ein (ungünstiger) Pfad längere Zeit nicht mehr von Ameisen frequentiert,

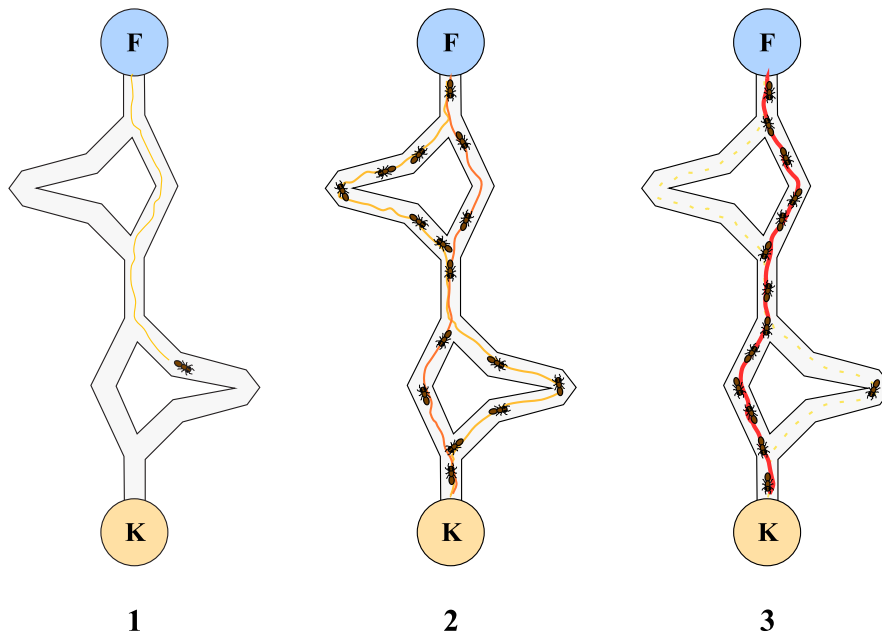


Abbildung 1: Veranschaulichung der Futtersuche von Ameisen. Die Futterquelle ist mit F markiert, die Kolonie mit K. Pheromone werden farblich auf den Wegen eingezeichnet. Je rötlicher und dicker die Pheromone dargestellt sind, desto stärker sind sie.

Phase 1 zeigt eine einzelne Ameise, die die Futterquelle entdeckt hat.

Phase 2 veranschaulicht die Explorationsphase, auf der alle möglichen Pfade frequentiert werden.

Phase 3 zeigt die Wege der Ameisen nach Konvergenz.

so weist er schließlich keine Pheromonablagerungen mehr auf. Dadurch hat er für die Ameisen auch keine zusätzliche Anziehungskraft mehr. In Abbildung 1, Seite 5 wird der Ablauf dieses Verhaltens veranschaulicht.

Zu beachten ist außerdem, dass echte Ameisen robust auf Störungen reagieren. Falls ein gefundener kürzester Pfad blockiert wird, gelingt es den Ameisen, nach kurzer Zeit, sich an die neue Situation anzupassen und eine möglichst gute Alternativroute zu finden. Diese Eigenschaft wird auch von Ameisenalgorithmen übernommen. Sowohl die Algorithmen, als auch das biologische Vorbild können keine optimale Lösung garantieren. In beiden Fällen wird jedoch meist ein gutes lokales Minimum erreicht.

Im nun folgenden Kapitel 2 wird die Übertragung des natürlichen Vorbilds auf eine Metaheuristik für Optimierungsprobleme beschrieben.

## 2 Ant Colony Metaheuristik

Die sogenannte *Ant Colony Metaheuristik*, im Folgenden als ACO (kurz für *Ant Colony Optimization*) bezeichnet, geht auf Marco Dorigo zurück. Die hier verwendeten Notationen sind an seine Notationen aus [5] angelehnt. Dort lässt sich auch eine ausführlichere Beschreibung der ACO finden, in der unter anderem eine Lösung des *Travelling Salesman Problem* (TSP) als Beispiel präsentiert wird. In Abschnitt 2.1 wird zunächst die Grundidee der Ameisenalgorithmen vorgestellt. Abschnitt 2.2 beschreibt, wie diese Idee formalisiert wird und in Abschnitt 2.3 werden noch einige weitere Eigenschaften von Ameisenalgorithmen erläutert.

### 2.1 Grundidee

Ameisenalgorithmen versuchen in ihrer Implementierung und formalisierung möglichst nahe an ihrem Vorbild aus der Natur zu bleiben. Wie bei einer echten Ameisenkolonie gibt es keine zentrale Instanz, die für einzelne Ameisen plant, was als nächstes zu tun sei. Noch sind die künstlichen Ameisen, mit deren Hilfe Lösungen generiert werden, besonders komplex und können ohne Zusammenspiel mit anderen Ameisen gute Lösungen finden. Auch der Informationsaustausch zwischen Ameisen findet nicht direkt statt, sondern über Pheromone.

Um ein Problem mittels Ameisenalgorithmen lösen zu können, muss dieses Problem zunächst auf einen Graphen abgebildet werden. Abhängig von der Problemstellung kann den Knoten verschiedene Bedeutung zufallen. Für das TSP beispielsweise bietet es sich an, jede Stadt, die besucht werden soll, durch einen Knoten zu repräsentieren. Ebenso ist die Bedeutung der Kanten zwischen den einzelnen Knoten abhängig von der Problemstellung. Beim TSP existiert genau dann eine Kante zwischen zwei Knoten existieren, wenn es eine Verbindung zwischen diesen beiden Städten gibt. Mit den jeder Kante wird eine bestimmte Kostenfunktion assoziiert. Im Falle des TSP sind die Kosten einer Kante einfach durch die Wegstrecke zwischen den beiden verbundenen Städten bestimmt. Lösungen werden von einer künstlichen Ameise generiert, indem sie im mit Hilfe der Kanten zwischen verschiedenen Knoten traviersiert, bis eine gültige Lösung gefunden wurde. Wann eine Lösung von einer künstlichen Ameise gefunden wurde ist natürlich ebenfalls problemabhängig und würde beim TSP darin bestehen, dass alle Knoten genau einmal besucht wurden. Es wird stets in jeder *Generation* des Algorithmus eine gleichbleibende Anzahl von künstlichen Ameisen ausgesickt. Von diesen Ameisen werden künstliche Pheromone auf den Kanten des Graphen plziert, um anzuzeigen, ob eine Kante zu einer vielversprechenden Lösung beitragen kann. Pheromone können entweder direkt dann abgelegt werden, wenn eine Ameise eine Kante benutzt oder nachträglich, nachdem eine Bewertung der gefundenen Lösung stattgefunden hat. Ist eine Lösung generiert und wurden eventuell nachträglich Pheromone für diese Lösung verteilt, so hat die künstliche Ameise ihren Zweck erfüllt und alle von ihr verbrauchten Ressourcen können freigegeben werden. Sobald alle Ameisen einer Generation Lösungen generiert haben, beginnt die nächste Generation mit identischen Ameisen. Diese werden jedoch in ihren Entscheidung von den verteilten Pheromonwerten früherer Generationen beein-

flußt. Terminieren kann der Algorithmus entweder nach einer vorgegebenen Anzahl von Generationen oder wenn eine Lösung bestimmter Güte erreicht wurde.

Nachdem jetzt die Grundidee des Algorithmus erläutert wurde, folgt ein Abschnitt, in dem gezeigt wird, wie sich diese Ideen formalisieren lassen.

### 2.2 Formalisierung des Problems

Um die in Abschnitt 2.1 beschriebenen Ideen auf ein konkretes Probleme anwenden zu können, muss das Problem entsprechend formuliert werden. Im Folgenden werden einige grundsätzliche Notationen für Ameisenalgorithmen vereinbart. Einige der aufgeführten Begriffe wurden schon im vorherigen Abschnitt erklärt und werden jetzt hier formal definiert. Die eingeführten Definitionen werden so, oder in ähnlicher Form, in den meisten Implementierungen von ACO auftauchen. Die von Dorigo verwendeten englischen Bezeichnungen werden jeweils in Klammern aufgeführt:

- Eine endliche Knotenmenge (oft auch bezeichnet als *components*)  
 $C = \{c_1, c_2, \dots, c_{N_C}\}$ , wobei  $N_C$  die Anzahl der Knoten bezeichnet.
- Eine endliche Kantenmenge  $L$  (*transitions / connections*), auf den Elementen aus  $\tilde{C}$ .  $\tilde{C}$  sei als Teilmenge des Kartesischen Produkts  $C \times C$  definiert.  
 $L = \{l_{c_i c_j} \mid (c_i, c_j) \in \tilde{C}\}$  Eine einzelne Kante zwischen zwei Knoten  $c_i$  und  $c_j$  wird als  $l_{c_i c_j}$  bezeichnet.
- Eine (lokale) Kostenfunktion  $J_{c_i c_j}$  (*connection cost*) für jede Kante  $l_{c_i c_j}$ . Je nach Problemstellung kann die Zeit ein weiterer Parameter in der Kostenfunktion sein.  
 $J_{c_i c_j} \equiv J(l_{c_i c_j}, t)$
- Eine endliche Menge an Nebenbedingungen  $\Omega$  (*constraints*), geknüpft an die Elemente aus  $C$  und  $L$ .  
 $\Omega \equiv \Omega(C, L, t)$
- Ein Zustand  $s$  (*state*) ist definiert, als eine Sequenz von Knoten aus  $C$ .  
 $s = \langle c_i, c_j, \dots, c_k, \dots \rangle$   
Sei  $S$  die Menge aller Zustände, so wird die Menge der gültigen Zustände (*feasible states*), unter Berücksichtigung von  $\Omega(C, L, t)$ , als  $\tilde{S}$  bezeichnet. Offensichtlich gilt  $\tilde{S} \subseteq S$ . Die Länge der Abfolge eines Zustandes wird durch  $|s|$  angegeben.
- Als Nachbarschaft (*neighborhood structure*) zwischen zwei Zuständen  $s_1 = \langle c_i, \dots, c_{s1} \rangle$  und  $s_2 = \langle c_{s1}, c_{s2} \rangle$ ,  $s_1, s_2 \in S$  bezeichnet man den Fall, wenn  $s_2$  eine Erweiterung von  $s_1$  um einen Knoten  $c_{s2} \in C$  ist und eine Kante  $l_{c_{s1} c_{s2}} \in L$  zwischen dem letzten Knoten in  $s_1$  und dem hinzugefügten Knoten besteht.
- Eine Lösung  $\Psi$  (*solution*) besteht aus einem gültigen Zustand  $s \in \tilde{S}$  der alle Anforderungen des Problems erfüllt. Eine Lösung bezeichnet man als *mehrdimensional*, wenn sie über mehrere unterschiedliche Zustände realisiert werden kann.

- Es existieren Pfadkosten (*cost*)  $J_\Psi(L, t)$  für jede Lösung  $\Psi$ . Hierbei ist  $J_\Psi(L, t)$  eine Funktion über alle lokalen Kosten  $J_{c_i c_j}$ , der Kanten, die in der Lösung  $\Psi$  enthalten sind.

Ist ein Optimierungsproblem auf einen Graph  $G = (C, L)$  abgebildet, so versucht ACO Pfade in diesem Graph zu finden, die Lösungen  $\Psi_1 \dots \Psi_n$  entsprechen. Wann ein Pfad eine gültige Lösung darstellt, ist problemabhängig. Beispielsweise wäre eine gültige Lösung für das TSP ein Hamiltonkreis im Graph, also ein Pfad, bei dem jeder Knoten des Graphen genau einmal besucht wird. Bei der Pfadgenerierung wird versucht die Kosten  $J_\Psi(L, t)$  zu minimieren, unter Berücksichtigung aller Nebenbedingungen  $\Omega$ .

Pfade werden von künstlichen Ameisen iterativ generiert. Beginnend von einem Startknoten wird der Graph so lange traversiert, bis eine gültige Lösung erstellt ist. Einzelne Ameisen sind in der Lage gültige Lösungen zu erstellen und eine Lösung mittels der Pfadkosten zu bewerten. Um gezielt möglichst günstige Lösungen zu finden ist jedoch ein Informationsaustausch zwischen den Ameisen nötig. Der Informationsaustausch findet, ähnlich wie im natürlichen Vorbild, indirekt über eine Veränderung der Umgebung statt: Für jede Kante  $l_{c_i c_j}$ , im Folgenden abgekürzt durch  $l_{ij}$ , werden künstliche *Pheromone*  $\tau_{ij}$  hinterlegt. Je höher der Pheromonwert  $\tau_{ij}$  für eine Kante  $l_{ij}$ , desto attraktiver wird es für eine Ameise diese Kante als nächstes zu benutzen. Neben den Pheromonwerten, die sich im Laufe des Algorithmus ändern kann die Auswahl der Kanten auch von vorgegebenen *Heuristiken*  $\eta_{ij}$  beeinflusst werden. Zu Beginn des Algorithmus können heuristische Werte Ameisen dazu bewegen vielversprechend erscheinende Kanten zu bevorzugen. Beim TSP könnte mittels einer Heuristik beispielsweise aus der Menge der auswählbaren Kanten die Kante mit den geringsten Kosten etwas bevorzugt werden.

### 2.3 Generelle Eigenschaften von ACO

ACO und davon abgeleitete Algorithmen zeichnen sich durch einige generelle Eigenschaften aus. Die Wichtigsten sind hier kurz skizziert:

- Einzelne Ameisen können inkrementell Lösungen generieren. Gute Lösungen entstehen im Allgemeinen jedoch erst durch Informationsaustausch zwischen Ameisen. Informationsaustausch wird über Pheromone realisiert.
- Ameisen nutzen neben den in ihnen gespeicherten Informationen lediglich lokale Informationen. Zu den lokalen Informationen zählen, in einem Knoten  $c_i$ , die Menge der Kanten  $l_{ij}$ , die diesen Knoten mit weiteren Knoten verbinden, und die zugehörigen Pheromonwerte  $\tau_{ij}$  und heuristischen Werte  $\eta_{ij}$ .
- Ameisen selbst sind nicht adaptiv. Das bedeutet, die Ameisen selbst verändern sich nicht im Laufe des Algorithmus. Unterschiedliches Verhalten von Ameisen verschiedener Generationen wird lediglich durch lokale Modifikationen der Pheromonwerte der einzelnen Kanten erreicht.

Einzelne Ameisen müssen ebenfalls über bestimmte Eigenschaften verfügen, damit die ACO verlässliche Ergebnisse liefert. Auch diese sind hier kurz aufgeführt:

- Ameisen suchen nach Lösungen mit minimalen Kosten  $\hat{J}_\Psi = \min_\Psi J_\Psi(L, t)$ .
- Jede Ameise  $k$  besitzt ein Gedächtnis  $M^k$ . Dies wird benutzt um den aktuellen Zustand der Ameise festzuhalten. Mit Hilfe des Zustandes lässt sich der bisher zurückgelegte Weg der Ameise ermitteln. So ist es stets möglich auf Zulässigkeit zu prüfen werden, eine gefundene Lösung zu bewerten und, falls nötig, den Pfad zurückzuverfolgen.
- Eine Ameise  $k$  kann einen bestimmten vorgegebenen Startzustand  $s_s^k$  besitzen und eine oder mehrere Endbedingungen. Für einige konkrete Probleme ist dieser Punkt jedoch optional. Um beispielsweise gültige Lösungen für das TSP zu finden, kann der Startzustand für jede Ameise beliebig gewählt werden.
- Eine Ameise  $k$  in Zustand  $s_r = \langle s_{r-1}, i \rangle$  kann jeden Knoten in ihrer zulässigen Nachbarschaft  $N_i^k$  besuchen. Es gilt:  $N_i^k = \{j \mid (j \in N_i) \wedge (\langle s_r, j \rangle \in \tilde{S})\}$  mit  $N_i = \{c_j \mid l_{ij} \in L\}$
- Eine Ameise  $k$  in Knoten  $i$  kann sich zu jedem Knoten  $j \in N_i^k$  bewegen. Die Wahl des nächsten Knotens wird probabilistisch getroffen.
- Die probabilistische Auswahl wird aufgrund einer lokal in Knoten  $i$  abgelegten Datenstruktur  $\mathcal{A}_i = [a_{ij}]$  getroffen. In  $\mathcal{A}_i$  sind Pheromonwerte und heuristische Werte für jede ausgehende Kante  $l_{ij}$  abgelegt. Zusätzlich fließen das Gedächtnis  $M^k$  und die Nebenbedingungen  $\Omega$  in den Auswahlprozess ein.
- Wird von Knoten  $i$  als nächstes ein Nachbarknoten  $j$  besucht, so kann eine Ameise den Pheromonwert  $\tau_{ij}$  aktualisieren.
- Ist eine Lösung erstellt worden, kann eine Ameise mit ihrem Gedächtnis  $M^k$  ihren zurückgelegten Pfad zurückverfolgen und dabei die Pheromonwerte aktualisieren.
- Von den zwei gerade vorgestellten Arten des Pheromonupdates sollte mindestens eine Art implementiert werden. Es können aber auch durchaus beide Methoden in einer Implementation zusammen verwendet werden.
- Sobald eine Ameise eine Lösung erstellt hat und evtl. ihren Pfad zurückverfolgt hat, stirbt sie und alle von ihr benutzten Ressourcen können freigegeben werden.

### 3 Anwendung auf das RCPSP

Im vorangegangenen Kapitel 2 wurden die allgemeinen Prinzipien und formalen Notationen von Ameisenalgorithmen eingeführt. In diesem Kapitel wird nun gezeigt, wie sich diese Prinzipien abwandeln lassen, um gute Lösungen für Instanzen des RCPSP zu generieren.

Um die im vorangegangenen Kapitel 2 vorgestellten Methoden auf eine Instanz des RCPSP anwenden zu können, muss das RCPSP zunächst in einer geeigneten Weise repräsentiert werden. Der von Merkle u.a. in [13] vorgestellte Ansatz nutzt die ACO Metaheuristik um Listen von Aktivitäten zu generieren, aus denen anschließend durch Plangenerierungsschemata konkrete Pläne erstellt werden. Die in [13] verwendeten Schemata waren das PSGS und das SSGS. Informationen zu Plangenerierungsschemata sind in [2], Kapitel 3, Abschnitt 3.2 zu finden. Um eine Liste von Aktivitäten für eine gegebene Instanz des RCPSP mittels ACO zu erstellen, muss die Instanz in einem für ACO geeigneten Graph abgebildet werden. Merkle u.a. [13] bilden dazu die Aktivitäten auf die Knotenmenge  $C$  ab. Innerhalb der Knotenmenge existiert ein zusätzlich generierter Startknoten  $c_0$  und ein Endknoten  $c_{n+1}$  bei  $n$  gegebenen Aktivitäten. Eine zulässige Reihenfolge von Knoten entspricht einer topologischen Sortierung im Vorrangnetzwerk. Das heißt in einem gegebenen Zustand ist ein Knoten  $c_j$  genau dann auswählbar, wenn alle Knoten  $c_i$ , mit einer Vorrangsbeziehung der Art  $i \rightarrow j$  bereits besucht wurden. Über den aktuellen Zustand, also bisherige Aktivitätenliste werden alle Knoten die bereits besucht wurden ausgeschlossen, so dass jeder Knoten nur genau einmal in die Liste eingefügt wird. Im Gegensatz zur Notation im vorangegangenen Kapitel 2 gibt es im Ansatz von Merkle u.a. [13] nicht für jede einzelne Kante  $l_{ij}$  zwischen zwei Knoten  $c_i$  und  $c_j$  Pheromonwerte, sondern die Pheromone werden in einer zentralen Matrix gespeichert, der sogenannten *Pheromonmatrix*. Die Pheromonmatrix ist eine  $(n+2) \times (n+2)$  Matrix, falls es  $n$  Aktivitäten gibt. Pheromonwerte  $\tau_{ij}$  geben in diesem Ansatz an, wie günstig es zu sein scheint, Aktivität  $j$  an Stelle  $i$  in der Aktivitätenliste einzufügen. Als Abbruchkriterium des Algorithmus wird entweder eine feste Anzahl von Generationen vorgeschlagen oder wenn die durchschnittlichen Lösungen sich über eine bestimmte Anzahl von Generationen nicht mehr verändert haben.

Alle Ameisen starten bei dem Startknoten  $c_0$  und durchlaufen den Graphen so lange von Knoten zu Knoten, bis sie zum Endknoten  $c_{n+1}$  gelangen. Unter den oben beschriebenen Voraussetzungen entsteht so eine vorangzulässige Liste von Aktivitäten. Aus einer solchen Liste wird mit Hilfe des SSGS oder des PSGS ein zulässiger Plan generiert, wobei die Plangenerierungsschemata für die Ressourcenzulässigkeit sorgen. Hierbei ist zu beachten, dass nach der obigen Notation der Knoten  $c_{n+1}$  erst erreicht werden kann, wenn alle seine Vorangsbeziehungen erfüllt sind, das heißt, wenn alle anderen Aktivitäten bereits in die Liste eingefügt wurden.

Die Entscheidung welcher Knoten aus der Menge der zulässigen Knoten von einer Ameise als nächstes besucht wird, wird probabilistisch getroffen. Die Wahrscheinlichkeiten werden hier einerseits von den Pheromonwerten  $\tau_{ij}$  der Pheromonmatrix als auch von heuristischen Werten  $\eta_{ij}$  beeinflusst.

Nachdem die generelle Idee der Anwendung von ACO zur Lösung einer RCPSP Instanz

beschrieben wurden folgen nun einige algorithmische Details. Abschnitt 3.1 beschreibt die Probabilistische Auswahl des nächsten Knotens. In Abschnitt 3.2 wird erläutert, wie das Pheromonupdate vorgenommen wird, während in 3.3 auf die Adaption bekannter Heuristiken für ACO eingegangen wird. In Abschnitt 3.4 werden schließlich noch einige optionale Modifikationen vorgestellt.

### 3.1 Entscheidungen für Ameisen

Eine Ameise, die sich in einem Knoten  $c_i$  ungleich dem Endknoten befindet, muss entscheiden welcher Knoten aus der Menge der zulässigen Knoten als nächster besucht werden soll. Diese Entscheidung wird probabilistisch getroffen und von zwei verschiedenen Größen beeinflusst. Die Menge an Pheromonen  $\tau_{ij}$  in der Pheromonmatrix, gibt Auskunft darüber, wie häufig Aktivität  $j$  an Stelle  $i$  der Aktivitätenliste von vorangegangenen Ameisen platziert wurde. für eine Kante  $l_{ij}$  von Knoten  $c_i$  zu einem Knoten  $c_j$  geben Auskunft darüber, wie häufig diese Kante schon von vorangegangenen Ameisen benutzt wurde. Je höher der Pheromonwerte  $\tau_{ij}$ , desto wahrscheinlicher scheint Aktivität  $j$  an Position  $i$  zu einer vorteilhaften Lösung zu gehören. Details zur Verteilung von Pheromonwerten für die Kantenmenge  $L$  des Graphen werden in Abschnitt 3.2 behandelt. Die zweite Größe, welche die Auswahl des nächsten Knotens beeinflusst ist der heuristische Wert  $\eta_{ij}$ . In den heuristischen Werten wird versucht Vorwissen über die Problemstellung zu nutzen, um einen möglichst gute Position zu finden. Im Gegensatz zu den Pheromonwerten ändern sich die heuristischen Werte  $\eta_{ij}$  nicht während der Iterationen des Algorithmus. Verschiedene Heuristiken werden in Abschnitt 3.3 diskutiert. Merkle u.a. stellen in [13] zwei verschiedene Arten vor, um die Wahrscheinlichkeiten für den nächsten in die Aktivitätenliste einzufügenden Knoten zu ermitteln. Die Menge der momentan zulässigen Aktivitäten (und damit auch Knoten) sei mit  $\mathcal{E}$  bezeichnet. Die sogenannte direkte Auswertung setzt die Wahrscheinlichkeit  $p_{ij}$  den Knoten  $j$  an Stelle  $i$  einzufügen folgendermaßen fest:

$$p_{ij} = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in \mathcal{E}} [\tau_{ih}]^\alpha [\eta_{ih}]^\beta} \quad (1)$$

Über die Parameter  $\alpha, \beta \in \mathbb{R}, \alpha, \beta \geq 0$  wird der relative Einfluss von von Pheromonwerten und heuristischen Werten bestimmt. In Formel (1) werden ausschließlich der Eintrag  $\tau_{ij}$  aus der Pheromonmatrix und der heuristische Werte  $\eta_{ij}$  benutzt um die Wahrscheinlichkeit für Aktivität  $i$  an Position  $j$  in der Aktivitätenliste zu bestimmen. Von den auswählbaren Knoten bekommt der Knoten die größte Wahrscheinlichkeit, bei dem das Produkt aus relativem Pheromonwert und relativem heuristischem Wert am größten ist.

Eine alternative Möglichkeit zur Bestimmung der Wahrscheinlichkeiten ist die summierte Auswertung. Hier werden die Pheromonwerte für das Einfügen eines Knotens  $j$  zu einem früheren Zeitpunkt miteinbezogen. So können Aktivitäten bevorzugt werden, bei denen die Pheromonwerte bereits für einen früheren Platz der Aktivitätenliste groß waren.

$$p_{ij} = \frac{(\sum_{k=1}^i [\gamma^{i-k} \cdot \tau_{ij}])^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{h \in \mathcal{E}} (\sum_{k=1}^i [\gamma^{i-k} \cdot \tau_{ih}])^\alpha \cdot [\eta_{ih}]^\beta} \quad (2)$$

Die Parameter  $\alpha$  und  $\beta$  entsprechen den Parametern aus Formel (1). Der Parameter  $\gamma \in \mathbb{R}, \gamma > 0$  bestimmt, wie Pheromonwerte früherer Entscheidungen den relativen Wert der Pheromone beeinflussen. Werte  $\gamma < 1$  verleihen den Pheromonen weiter zurückliegender Entscheidungen weniger Gewicht,  $\gamma > 1$  stärken den Einfluss der Pheromone früherer Entscheidungen. Für  $\gamma = 1$  werden alle Pheromonwerte  $\tau_{kj}, k \in \{1, 2, \dots, i\}$  gleich stark gewertet.

### 3.2 Pheromonupdate

Da die Pheromonwerte  $\tau_{ij}$  eine große Rolle in der Auswahl des nächsten Knotens spielen (vgl. Formel (1) und (2)), beeinflusst die Wahl, wie Pheromone verteilt werden das Verhalten des Algorithmus entsprechend. Zu Beginn des Algorithmus wird jeder Pheromonwert  $\tau_{ij}$  mit dem gleichen Startwert  $\tau_0 > 0$  versehen. Diese Initialisierung ist notwendig, damit zu Beginn des Algorithmus, gemäß den gegebenen Auswertungsmethoden, die Wahrscheinlichkeiten für eine Entscheidung bestimmt werden kann. Da  $[\tau_{ij}]^\alpha$  als Faktor einfließt gilt sowohl nach (1) als auch nach (2)  $p_{ij} = 0$  für  $\tau_{ij} = 0$ . Sind die Pheromone initialisiert, so werden alle Pheromone nach einer Generation folgender Formel entsprechend aktualisiert:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} \quad (3)$$

Der Parameter  $\rho$  mit  $\rho \in \mathbb{R}, 0 \leq \rho < 1$  bestimmt, wie schnell abgelegte Pheromone evaporieren. Dies wird vorgenommen, damit alte Pheromone keinen zu starken Einfluss entwickeln und so das Auffinden anderer Lösungen behindern. Anschließend werden zusätzliche Pheromone für die beste innerhalb dieser Generation gefundene Lösung  $\Psi^*$  verteilt. Die Lösung  $\Psi^*$  entspricht einem Zustand  $\langle c_0, c_i, \dots, c_k, c_{n+1} \rangle$ , in dem die Knoten genau in der Reihenfolge vorkommen, in der sie in der Aktivitätenliste aufgeführt sind. Die Pheromone, die zu dieser Lösung gehören sind die Pheromone  $\tau_{ij}$ , bei denen der Index  $i$  der Position des Knoten  $j$  in der Aktivitätenliste entspricht. Alle  $\tau_{ij}$ , die dieses Kriterium erfüllen, werden folgendermaßen modifiziert:

$$\tau_{ij} = \tau_{ij} + \rho \cdot \frac{1}{2T^*} \quad (4)$$

Der Parameter  $T^*$  entspricht dem Makespan für die Lösung  $\Psi^*$ . Je besser also die Lösung ist, desto mehr werden die Pheromonwerte, die an der Lösung beteiligt sind, erhöht. Es findet also eine nachträgliche Bewertung generierter Lösungen statt und kein iteratives Verteilen von Pheromonen, während der Lösungsgenerierung (vgl. 2.3) Auf gleiche Weise werden auch die Pheromonwerte der insgesamt besten gefundenen Lösung modifiziert. Dies führt dazu, dass künftige Ameisen mit höherer Wahrscheinlichkeit in der Nähe der bisher besten Lösung nach weiteren Lösungen suchen. Eine solche Strategie kann dazu führen, dass sich der Algorithmus schnell in ein lokales Minimum bewegt und aus diesem nicht wieder herausfindet, da stets Pheromone erhöht werden, die in Richtung des lokalen

Symbol	Bedeutung
$LF_i$	'latest finish time' von Aktivität $i$
$LS_i$	'latest start time' von Aktivität $i$
$ES_i$	'earliest start time' von Aktivität $i$
$S_i^*$	Menge aller Nachfolger von Aktivität $i$
$S_i$	Menge der direkten Nachfolger von Aktivität $i$
$\mathcal{E}$	Menge aller auswählbaren Aktivitäten
$\mathcal{Q}$	Menge der Ressourcen
$R_l$	Kapazität von Ressource $l$
$p_i$	Dauer von Aktivität $i$
$r_{il}$	Ressourcenanforderung von Aktivität $i$ an Ressource $l$

Tabelle 1: Notationen für Heuristiken

Minimums führen. In Abschnitt 3.4.3 wird eine Methode vorgestellt, mit der man dieses Risiko vermindern kann.

### 3.3 Heuristiken

Die hier vorgestellten Heuristiken fließen gemäß Formel (1) und (2) in die Wahrscheinlichkeiten ein, dass Aktivität  $j$  an Position  $i$  in der Aktivitätenliste auftaucht. Die heuristischen Werte  $\eta_{ij}$  verändern sich im Gegensatz zu den Pheromonen nicht. Ihre Aufgabe ist, die Ameisen in den ersten Iterationen in vielversprechend erscheinende Regionen zu führen. Problematisch können Heuristiken werden, wenn sie im Verhältnis zu den Pheromonen zu stark gewichtet sind und somit zu viel Einfluss auf den Weg einer Ameise ausüben. Bevor die einzelnen Heuristiken vorgestellt werden, sind einige Notationen in Tabelle 1 vereinbart.

Auf die einzelnen Heuristiken soll hier nur kurz eingegangen werden. Die Bedeutung der Heuristiken lässt sich schnell mit Hilfe der entsprechenden Formel und einer knappen Erläuterung erschließen. Relevant werden die Heuristiken im Hinblick auf die erzielten, in Abschnitt 4.1 diskutierten, Ergebnisse. Weitere Heuristiken, die sich auch prinzipiell für ACO adaptieren lassen, sind in [2], Kapitel 3, Abschnitt 3.3 zu finden. Alle hier vorgestellten Heuristiken haben gemeinsam, dass sie sich aus Eigenschaften der Aktivitäten im Vergleich mit anderen, zu diesem Zeitpunkt zulässigen, Aktivitäten ergeben. Zudem ergeben sich die Werte für die einzelnen Heuristiken stets dadurch, dass sie mit den Werten anderer verfügbarer Aktivitäten ins Verhältnis gesetzt werden. Dies spielt bei der Anwendung für ACO eine Rolle, da es hier nicht nur auf eine heuristische Reihenfolge unter den Aktivitäten ankommt, sondern die heuristischen Werte direkt über Formel (1) und (2) in den Algorithmus einfließen. Zum Beispiel wachsen meist die spätesten Endzeitpunkte für Aktivitäten die erst spät zulässig werden (die also viele Vorbedingungen haben). Dadurch, dass sie mit allen anderen zulässigen Aktivitäten ins Verhältnis gesetzt werden, lassen sich für diese Aktivitäten eher ähnliche heuristische Werte erzielen, wie für früh zulässige Aktivitäten. Mit  $\eta_{ij}$  wird stets der heuristische Wert bezeichnet, mit dem vorgeschlagen wird Aktivität  $j$  an Position  $i$  in der Aktivitätenliste zu platzieren.

Da die Position  $i$  nicht direkt in die Heuristik einspielt, sondern nur indirekt über die Menge der bereits zulässigen Aktivitäten  $\mathcal{E}$ , findet sich  $i$  folglich nicht auf der rechten Seite der angegebenen Formeln.

Die „Latest Finish Time“ (LFT) Heuristik bevorzugt Aktivitäten, deren spätester Endzeitpunkt im Verhältnis zu den anderen zulässigen Aktivitäten früh ist:

$$\eta_{ij} = \max_{k \in \mathcal{E}} LF_k - LF_j + 1$$

„Latest Start Time“ (LST) bevorzugt Aktivitäten deren spätester Startzeitpunkt im Verhältnis klein ist, die also früh gestartet werden müssen:

$$\eta_{ij} = \max_{k \in \mathcal{E}} LS_k - LS_j + 1$$

„Minimum Slack Time“ (MSL) versucht Aktivitäten, die einen verhältnismäßig kleinen Spielraum bezüglich ihrer Anfangszeit haben, möglichst schnell einzuplanen:

$$\eta_{ij} = \max_{k \in \mathcal{E}} (LS_k - ES_k) - (LF_j - ES_j) + 1$$

„Most Total Succesors“ (MST) berücksichtigt die Anzahl aller Nachfolger einer Aktivität:

$$\eta_{ij} = |S_j^*| - \min_{k \in \mathcal{E}} |S_k^*| + 1$$

„Greatest Rank Positional Weight All“ (GRPWA) sortiert Aktivitäten nach ihrer Dauer, sowie der Summe der Dauer aller ihrer Nachfolger:

$$\eta_{ij} = p_j + \sum_{i \in S_j^*} p_i - \min_{k \in \mathcal{E}} \left( p_k + \sum_{i \in S_k^*} p_i \right) + 1$$

„Weighted Resource Utilization and Precedence“ (WRUP) betrachtet die Ressourcenanforderungen durch eine Aktivität und die Anzahl ihrer direkten Nachfolger:

$$\eta_{ij} = \omega |S_j| + (1 - \omega) \sum_{l \in \mathcal{Q}} \frac{r_{jl}}{R_l} - \min_{k \in \mathcal{E}} \left( \omega |S_k| + (1 - \omega) \sum_{l \in \mathcal{Q}} \frac{r_{kl}}{R_l} \right) + 1 \quad \omega \in [0, 1]$$

### 3.4 Optionale Modifikationen

Nachdem in den vorangegangenen Abschnitten die Übertragung der generellen Methoden der ACO auf das RCPSP vorgestellt wurden, werden in diesem Abschnitt einige optionale Modifikationen vorgestellt. Mit Hilfe dieser Modifikationen lassen sich teilweise bessere Ergebnisse erzielen, jedoch sind sie keine Voraussetzungen für ein korrektes Arbeiten des Algorithmus.

### 3.4.1 Kombination von direkter und summierter Auswertung

Problembedingt kann sich sowohl die direkte Auswertung, als auch die summierte Auswertung (siehe 3.1) als günstig erweisen. Merkle u.a. schlagen daher in [13] eine Kombination von direkter und summierter Auswertung vor. Um dies zu erreichen, schlagen sie vor für die Berechnung der Wahrscheinlichkeiten Formel (1) zu benutzen und innerhalb der Formel den Pheromonwert  $\tau_{ij}$  durch den folgenden Ausdruck  $\tau'_{ij}$  zu ersetzen:

$$\tau'_{ij} = c \cdot x_i \cdot \tau_{ij} + (1 - c) \cdot y_i \cdot \sum_{k=1}^i \gamma^{i-k} \tau_{kj}$$

Mit  $x_i = \sum_{h \in \mathcal{E}} \sum_{k=1}^i \gamma^{i-k} \tau_{kh}$  und  $y_i = \sum_{h \in \mathcal{E}} \tau_{ih}$ . Der Parameter  $c \in \mathbb{R}, 0 \leq c \leq 1$  bestimmt dabei die Gewichtung von direkter zu summierter Auswertung. Die Faktoren  $x_i$  und  $y_i$  sind eingefügt, um den relativen Einfluss von direkter und summierter Auswertung anzupassen. Merkle u.a. behaupten in [13], dass für  $c = 1$  eine direkte Auswertung vorliegen würde und für  $c = 0$  eine summierte Auswertung. Dies wäre der Meinung des Autors nach nur korrekt, wenn man auf die Faktoren  $x_i$  und  $y_i$  in obiger Formel verzichten würde.

### 3.4.2 Parameteränderung während der Laufzeit

Während die in den vorigen Abschnitten erwähnten Parameter  $(\alpha, \beta, \rho)$  normalerweise einmal zu Beginn des Algorithmus fest gewählt werden, wird von Merkle u.a. vorgeschlagen, einige dieser Parameter während der Laufzeit zu verändern. Der Parameter  $\beta$  bestimmt den Einfluss der Heuristik auf den Entscheidungsprozess einer Ameise (vgl. Formel (1) und (2)). Zu Beginn des Algorithmus ist die Heuristik in der Regel hilfreich, um gute Lösungen zu generieren. Später können zu große heuristische Werte jedoch verhindern, dass Ameisen guten Pheromonen folgen und so die Erstellung guter Lösungen behindern. Deshalb scheint es sinnvoll, mit einem recht großen Startwert für  $\beta$  zu beginnen und  $\beta$  im Verlauf des Algorithmus bis auf 0 zu verkleinern, so dass später nur noch Pheromone die Wahrscheinlichkeiten für die Ameisen beeinflussen.

Mit  $\rho$  wird die Konvergenzgeschwindigkeit des Algorithmus gesteuert. Je höher  $\rho$ , desto schneller evaporieren alte Pheromonwerte (siehe Formel (3)) und desto mehr Pheromone werden auf die derzeit besten Lösungen verteilt (Formel (4)). Der Algorithmus wird also schneller konvergieren. Kleine Werte von  $\rho$  führen dazu, dass die Konvergenzgeschwindigkeit sinkt und der Algorithmus einen größeren Teil des Suchraums explorieren kann. Falls die Anzahl von Generationen für den Algorithmus beschränkt ist, schlagen Merkle u.a. deshalb zunächst einen kleinen Startwert für  $\rho$  vor, der während dem größten Teil der Laufzeit beibehalten werden sollte. Gegen Ende der Laufzeit sollte  $\rho$  dann erhöht werden, um das Konvergenzverhalten zu verbessern. Inwieweit man  $\rho$  verändern sollte, wenn die Anzahl der Generationen nicht begrenzt ist, wird nicht weiter beschrieben.

### 3.4.3 Entkommen aus lokalen Minima

Nach Abschnitt 3.2 wird stets die beste bisher gefundene Lösung mit weiteren Pheromonen versehen. Das kann dazu führen, dass die Suche zu stark auf die Nachbarschaft dieser bisher besten Lösung konzentriert wird. Besonders nachteilig ist dies, wenn in der Nähe dieser Lösung keine weiteren guten Lösungen existieren. Um einer zu frühen Konvergenz in einem lokalen Minimum vorzubeugen kann der Algorithmus so modifiziert werden, dass die bisher beste Lösung nur für eine maximale Anzahl  $g_{max}$  von Generationen mit weiteren Pheromonen versehen wird, solange keine bessere Lösung gefunden wurde. Nach  $g_{max}$  Generationen wird dann die beste Lösung innerhalb der Generation für das Update der Pheromonwerte zur besten bisher gefundenen Lösung.

### 3.4.4 Lokale Suche

Manchmal lassen sich die von ACO gefundenen Lösungen durch lokale Suchverfahren in der Nähe der Lösungen weiter verbessern. Als Nachbarschaften für die lokale Suche werden in Merkle u.a. [13] Rechtsshifts von Aktivitäten in der Liste der Aktivitäten oder der Tausch der Positionen von zwei Aktivitäten angegeben. Weitere Details zu diesen und anderen Nachbarschaften finden sich in [2], Kapitel 3, Abschnitt 3.4.

### 3.4.5 Bidirektionale Planung

Statt das Problem nur mit einer Kolonie von Ameisen zu bearbeiten, kann man teilweise bessere Lösungen erzielen, wenn man eine zweite Kolonie hinzunimmt, die das RCPSP von 'hinten nach vorne' plant. Dazu müssen alle Vorangbeziehungen umgedreht werden und die Rolle der beiden Dummy-Knoten  $c_0$  und  $c_{n+1}$  wird vertauscht. Anschließend wird auf den beiden Problemen eine unabhängige ACO gestartet. Nach einer vorher festgelegten Anzahl von Generationen werden die bisher erzielten Ergebnisse beider Kolonien verglichen. Die Kolonie, die im Durchschnitt in den letzten Generationen bessere Ergebnisse erzielt hat, wird weitergeführt. Der Optimierungsprozess der anderen Kolonie wird abgebrochen.

## 4 Ergebnisse und Diskussion

In diesem Kapitel soll, aus Platzgründen nur sehr knapp, auf die Ergebnisse von Merkle u.a. [13] eingegangen werden. Anschließend erfolgt in Abschnitt 4.2 eine kurze Bewertung, welche die persönliche Sicht des Autors zum Ansatz von Merkle u.a. widerspiegelt.

### 4.1 Ergebnisse

Um ihren Algorithmus mit anderen Verfahren zu vergleichen, greifen Merkle u.a. auf PSPLIB [11; 12] zurück. Von den dort zur Verfügung gestellten Instanzen werden die `j120.sm` Instanzen als Benchmarkset ausgewählt.

Um vergleichbare Ergebnisse zu erzielen werden von allen verwendeten Algorithmen maximal 5000 Pläne generiert. Die von Merkle u.a. implementierte ACO Variante zeichnet sich durch folgende Eigenschaften aus: Pro Generation werden 5 Ameisen verwendet. Es werden maximal 850 Generationen durchlaufen, während der ersten 100 Generationen wird bidirektional geplant (vgl. 3.4.5). Nach 850 Generationen oder wenn die durchschnittliche Lösung pro Generation sich 10 Generationen lang nicht verändert hat, geht der Algorithmus zu einer lokalen Suche über, bis die Maximalanzahl von 5000 Plänen erreicht ist.

Folgende Parameterkonfiguration wurde benutzt:  $\alpha = 1$ ,  $c = 0.5$ ,  $\gamma = 1.0$ . Parameter  $\beta$  bekommt als Startwert 2.0 und wird linear so verringert, dass er nach 425 Generationen auf 0 fällt.  $\rho$  wird zu Beginn auf 0.025 gesetzt und für die letzten 200 Generationen auf 0.075 erhöht. Die Anzahl von Generationen, bevor die beste Lösung nicht mehr mit zusätzlichen Pheromonen versehen wird ist  $g_{max} = 10$ . Es wurden die verschiedenen Heuristiken getestet, von denen LST die besten Ergebnisse lieferte. Tabelle 2 liefert eine Übersicht, über den Einfluss der Heuristiken auf das Verhalten des Algorithmus. Um die Güte ihres Verfahrens zu bewerten, vergleichen Merkle u.a. in [13] die gefundenen Lösungen mit einer Unteren Schranke ( $LB$ ), die auf dem kritischen Pfad beruht und nach [13] in Stinson et al. [10] beschrieben wird. Je kleiner die Abweichung der gefundenen Lösung (nach oben) von der unteren Schranke, desto besser ist die Lösung zu bewerten. Die in Tabelle 2 gezeigten Ergebnisse stellen die gemittelten Ergebnisse von jeweils 4 Durchläufen über alle 600 Instanzen der PSPLIB [11; 12] dar.

Im Vergleich zu den anderen Algorithmen lieferte die in Merkle u.a. [13] vorgestellte ACO bei 5000 generierten Plänen die besten Ergebnisse. Ein kleiner Ausschnitt der Ergebnisse des Vergleichs sind in Tabelle 3 hinterlegt. Wie für die verschiedenen Heuristiken ergibt sich auch hier wieder die Güte nach der Abweichung von  $LB$  nach [10]. Die dritte Spalte gibt die Quelle der beschriebenen Verfahren nach Merkle u.a. [13] an. Auch bei diesem Vergleich wurden jeweils 4 Durchläufe für jede Instanz der PSPLIB durchgeführt. Für 278 der 600 Benchmarkprobleme wurde die bis dato beste Lösung eingestellt. Ohne eine Beschränkung auf 5000 erstellte Pläne konnte der Algorithmus für 130 Instanzen neue beste Lösungen finden.

Desweiteren untersuchen Merkle u.a. noch ausführlich die Einflüsse verschiedener Parameterkonfigurationen von  $\gamma$  und  $c$  sowie die Auswirkungen von Bidirektionaler Planung im Vergleich zu reiner Vorwärts- oder Rückwärtsplanung auf das Verhalten des Algorith-

Heuristik	Abweichung von $LB$ in %	Standardabweichung
LST	35.43	0.046
LFT	35.64	0.043
GRPWA	35.79	0.049
MTS	36.38	0.004
WRUP ( $\omega = 0$ )	39.22	0.080
WRUP ( $\omega = 0.5$ )	38.32	0.082
WRUP ( $\omega = 1$ )	38.70	0.037

Tabelle 2: Auswirkung der verschiedenen Heuristiken auf die Performanz bei ansonsten gleichen Parametern. In der mittleren Spalte wird jeweils die durchschnittliche prozentuale Abweichung des Zielfunktionswertes der generierten Lösungen von einer unteren Schranke  $LB$  (basierend auf kritischem Pfad) angegeben. Die rechte Spalte beschreibt die Standardabweichung der Zielfunktion der generierten Lösungen.

Algorithmus	Abweichung von $LB$ in %	Quelle
ACO für RCPSP	35.43	[13]
Selbstadaptierender Genetischer Algorithmus	35.60	[9]
Genetischer Algorithmus	36.74	[8]
Zeitorientiertes Branch&Bound	37.10	[6]
Simulated Annealing	37.68	[1]

Tabelle 3: Vergleich mit einigen anderen Algorithmen. Je kleiner die prozentuale Abweichung in der rechten Spalte, desto besser. Diese Tabelle stellt nur einen kleinen Ausschnitt der Ergebnisse dar, für genauere Informationen siehe [13].

mus. Der interessierte Leser sei daher an dieser Stelle auf die bei weitem umfangreichere Beschreibung der Ergebnisse in [13] hingewiesen.

### 4.2 Diskussion

Die in Merkle u.a. [13] gezeigten Ergebnisse wirken zunächst einmal beeindruckend. Jedoch stellt sich bei der Einordnung des vorgestellten Algorithmus mit anderen Algorithmen die Frage der Vergleichbarkeit. Die Autoren von [13] sind augenscheinlich bemüht die Algorithmen unter gleichen Bedingungen (5000 generierte Pläne) antreten zu lassen. Hierbei müsste für eine wirklich gerechte Bewertung allerdings auch noch der Aufwand berücksichtigt werden, den die einzelnen Verfahren für die Erstellung der 5000 Pläne betreiben. Zumindest im Vergleich mit einigen der anderen Algorithmen betreibt die vorgestellte ACO einen deutlich höheren Aufwand. Verfahren, bei denen Pläne mit weniger Aufwand generiert werden können und dafür in der Regel mehr Pläne zum Erzielen guter Ergebnisse brauchen, sind bei einer solchen Bewertungsmethode im Nachteil. Allerdings ist es zugegebenermaßen extrem schwierig ist, wirklich geeignete und faire Kriterien zur Beurteilung der Performanz von Algorithmen zur Lösung von Optimierungsproblemen zu finden.

Ein weiterer Kritikpunkt schließt sich an den vorigen Punkt an: Bei dem vorgestellten Verfahren werden die eigentlichen Pläne durch das PSGS oder das SSGS generiert. Der Ameisenalgorithmus liefert lediglich eine Liste von Aktivitäten, welche von den Plangenerierungsschemata als Eingabe benutzt wird. Für die Generierung dieser Listen wurde mit den Ameisenalgorithmen ein sehr hoher Aufwand getrieben. Es bleibt zu hinterfragen, ob man die Erstellung von Aktivitätenlisten nicht auf andere Art und Weise effizienter lösen könnte.

Nach der persönlichen Meinung des Autors können bei einem statischen Problem, wie der Listenerstellung, Ameisenalgorithmen ihre große Stärke nicht ausspielen. Zwar bleibt der Lösungsansatz elegant, da er eine einfach nachvollziehbare und relativ leicht implementierbare Idee umsetzt, aber der eigentliche Vorteil von Ameisenalgorithmen besteht in ihrer Adaptivität im Bezug auf Änderungen in ihrer Umgebung. Beispielsweise beim Routing von Datenströmen in realen Netzwerken, bei denen schwankende Auslastung des Netzwerkes und Ausfall einzelner Netzknoten berücksichtigt werden müssen, können diese Stärken viel eher angewandt werden. Dem interessierten Leser sei deshalb die Lektüre von beispielsweise [3; 4] nahegelegt.

## Literatur

- [1] BOULEIMEN, K. ; LECOCQ, H. *A new efficient simulated annealing algorithm for the resource constrained project scheduling problem*. Tech. Rep., Service de Robotique et Automatisation, Université de Liège. 1998
- [2] BRUCKER, Peter ; KNUST, Sigrid: *Complex Scheduling (GOR-Publications)*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2006. – ISBN 3540295453
- [3] DI CARO, Gianni ; DORIGO, Marco: AntNet: Distributed Stigmergetic Control for Communications Networks. In: *Journal of Artificial Intelligence Research* 9 (1998), S. 317–365
- [4] DI CARO, Gianni ; DORIGO, Marco. *Two ant colony algorithms for best-effort routing in datagram networks*. 1998
- [5] DORIGO, Marco ; DI CARO, Gianni: The Ant Colony Optimization Meta-Heuristic. In: CORNE, David (Hrsg.) ; DORIGO, Marco (Hrsg.) ; GLOVER, Fred (Hrsg.): *New Ideas in Optimization*. London : McGraw-Hill, 1999, S. 11–32
- [6] DORNDORF, U. ; PESCH, E. ; PHAN-HUY, T.: A Branch-and-Bound Algorithm for the Resource-Constrained Project Scheduling Problem. In: *Mathematical Methods in Operations Research* 51 (2000), S. 565–594
- [7] GOSS, S. ; ARON, S. ; DENEUBOURG, J. ; PASTEELS, J.: Self-organized shortcuts in the Argentine ant. In: *Naturwissenschaften* 76 (1989), December, Nr. 12, S. 579–581
- [8] HARTMANN, S.: A competitive genetic algorithm for resource-constrained project scheduling. In: *Naval Research Logistics* 45 (1998), S. 733–750
- [9] HARTMANN, S. *Self-adapting genetic algorithms with an application to project scheduling*. Tech. Rep. 506, University of Kiel. 1999
- [10] J.P. STINSON, E.W. D. ; KHUMAWALA, B.M.: Multiple resource-constrained scheduling using branch and bound. In: *AIIE Transactions* 10 (1978), S. 252–259
- [11] KOLISCH, R. ; SCHWINDT, C. ; SPRECHER, A.: *Benchmark Instances for Project Scheduling Problems*. Kluwer, Amsterdam, 1998, S. 147–178
- [12] KOLISCH, R. ; SPRECHER, A.: PSPLIB — a project scheduling problem library. In: *European Journal of Operational Research* 96 (1997), S. 205–216
- [13] MERKLE, Daniel ; MIDDENDORF, Martin ; SCHMECK, Hartmut: Ant Colony Optimization for Resource-Constrained Project Scheduling. In: WHITLEY, Darrell (Hrsg.) ; GOLDBERG, David (Hrsg.) ; CANTU-PAZ, Erick (Hrsg.) ; SPECTOR, Lee (Hrsg.) ; PARMEE, Ian (Hrsg.) ; BEYER, Hans-Georg (Hrsg.): *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*. Las Vegas, Nevada, USA : Morgan Kaufmann, 10-12 2000. – ISBN 1-55860-708-0, S. 893–900