DOI 10.2195/LJ_Not_Ref_d_ Buechter_052005

Sequenzierung mit Ant-Colony-Systemen am Beispiel Querverteil-Wagen

CLARA MARIA NOVOA, M.E.
INDUSTRIAL AND SYSTEMS ENGINEERING DEPARTMENT (ISE), LEHIGH UNIVERSITY

DIPL.-ING. HUBERT BÜCHTER,
FRAUNHOFER INSTITUT MATERIALFLUSS UND LOGISTIK (IML), DORTMUND

Dieser Beitrag zeigt die Anwendung des Ant-Colony-System (ACS) Algorithmus auf die Sequenzierung von Querverteil-Wagen in einem Lager. Wir erweitern den Basisalgorithmus der Ant-Colony-Optimierung (ACO) für die Minimierung der Bearbeitungszeit einer Menge von Fahraufträgen für die Querverteil-Wagen. Im Vergleich zu dem Greedy-Algorithmus ist der ACO-Algorithmus wettbewerbsfähig und schnell. In vielen Lagerverwaltungssystemen werden die Fahraufträge nach dem FIFO-Prinzip (First-in-First-out) ausgeführt. In diesem Beitrag wird der ACO-Algorithmus genutzt, um eine optimale Sequenz der Fahraufträge zu bilden.

This paper presents an application of the ant-colony systems (ACS) algorithm for sequencing traversing-cars in a warehouse system. We extend the basic ant-colony optimization algorithm (ACO) for minimizing the time required to serve a set of incoming requests to the traversing-cars. We also develop a greedy algorithm. The comparison between ACS and the greedy algorithm indicates that the ACS algorithm is competitive and fast. In many warehouse management situa¬tions, the rule for sequencing requests for traversing cars is the FIFO rule. The results on this paper show that the ACS algorithm applies to warehouse traffic sequencing.

1. Überblick

Der Abschnitt 2 beschreibt das zu lösende Problem. Abschnitt 3 zeigt eine Formulierung des Problems durch „Linear-Integer-Programming“. Abschnitt 4 gibt einen kurzen Überblick über die Literatur zum Thema Ant-Colony-Systeme (ACS). In Abschnitt 5 werden Erweiterungen des ACS-Algorithmus zur Lösung des Sequenzierungsproblems mit nur *einem* Querverteil-Wagen vorgestellt. In Abschnitt 6 wird ein Greedy-Algorithmus beschrieben und die nummerischen Ergebnisse für ACS- und Greedy-Verfahren diskutiert. In Abschnitt 7 wird der ACS-Algorithmus für zwei Querverteil-Wagen, die auf einer gemeinsamen Schiene fahren, erweitert. Abschnitt 8 fasst die Ergebnisse zusammen.

2. Problembeschreibung

Betrachten wir ein Lagersystem nach Abbildung 1.

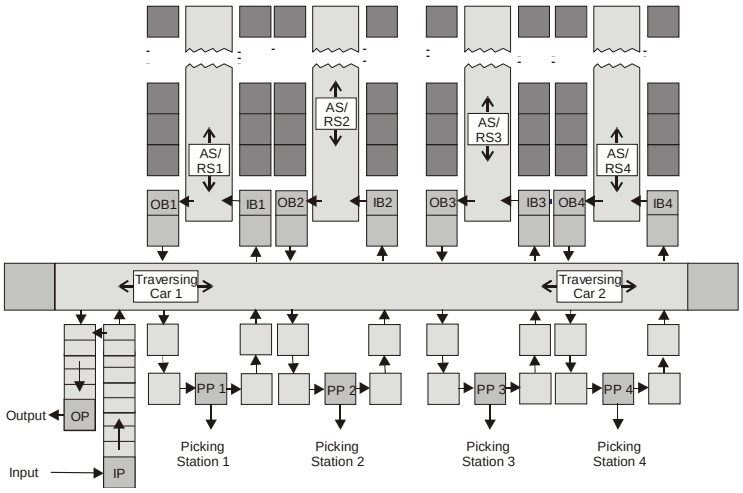

Abbildung 1: Hochregallager mit 4 Gassen und 4 Kommissionierplätzen

Das Lager verfügt über einen I-Punkt IP (Identifikation Point), vier Kommissionier-Stationen PP (Pick Place) mit jeweils einem Ein- und Ausgangspuffer beschränkter Kapazität. Die vier Lagergassen werden jeweils durch ein Regalbediengerät AS/RS (Automatic Storage and Retrieval System) bedient und verfügen ebenfalls über einen Eingangspuffer IB (Input Buffer) und einen Ausgangspuffer OB (Output Buffer) beschränkter Kapazität. Jedes AS/RS kann einerseits Paletten zwischen IB und den Lagerfächern und andererseits zwischen den Lagerfächern und OB transportieren. Das Lager verfügt über einen einzelnen Ausgabepunkt OP (Output Point). Jeder Fahrauftrag für den Querverteil-Wagen hat mit einer gewissen Wahrscheinlichkeit einen der Puffer IB oder den Ausgabepunkt KP zum Ziel. Ebenso existieren Fahraufträge zu den Eingangspuffern und von den Ausgangspuffern der Kommissionierstationen. Die Querverteil-Wagen bewegen sich horizontal auf einer gemeinsamen Schiene zwischen dem Lagerbereich und den Kommissionier-Stationen. Die Quellpunkte der Fahraufträge sind in Abbildung 1 mit einem Pfeil in Richtung der Gasse des Querverteil-Wagens und die Senken mit einem Pfeil in Gegenrichtung gekennzeichnet. Zurzeit werden die Fahraufträge nach dem FIFO-Prinzip abgearbeitet.

3. Formalisiertes Problem

Wir nennen das oben beschriebene Problem der Sequenzierung der Fahraufträge „sequencing traversing-cars problem“ (STCP). Die Lösung ist die optimale Sequenz, die die Zeit für die Abarbeitung einer vergegebenen Menge von Fahraufträgen minimiert.
Angenommen es existiert nur ein Querverteil-Wagen im System und die Anzahl der unterschiedlichen Quellen sei n. n_c sei die Anzahl der Elemente der Menge C der Fahraufträge, die sich aus der jeweils ersten Position einer Palette auf einer Quelle ergibt. C ist also die Menge der Fahraufträge, die zu einer gegebenen Zeit ausgeführt werden können. Zusätzlich enthält C den aktuellen Standort des Querverteil-Wagens, um die Kosten der ersten Fahrt zu einer Quelle berücksichtigen zu können. Wenn die Anforderung $j \in C$ gewählt wird, fährt der Wagen von der Startposition o_j zur Zielposition d_j. Wenn die aktuelle Position des Wagens d_i, $d_i \neq o_j$ ist, muss die Leerfahrt von d_i nach o_j berücksichtigt werden. Wird der Auftrag j unmittelbar nach dem Auftrag i ausgeführt, ergibt sich eine feste Zeit f_i, die der Wagen benötigt, um die Last von o_j nach d_j zu transportieren. Die Zeit, die zu minimieren ist, ist die Summe der Zeiten, in denen der Wagen unbeladen zum nächsten Auftragsziel fährt. Die Gleichung 1 beschreibt die Zielfunktion. Ohne Verlust der Allgemeingültigkeit sind die Zeiten der Lastfahrten ebenfalls berücksichtigt. In der Gleichung 1 ist x_{ij} eine 0/1-Entscheidungsvariable. Falls in der Lösung des STCP-Problems der Auftrag j unmittelbar auf den Auftrag i folgt, bekommt x_{ij} den Wert eins, sonst null.

$$\min \sum_{j \in C} \sum_{i \in C} (s_{ij} + f_j) x_{ij} \tag{1}$$

Nach jeder Bearbeitung eines Fahrauftrages werden die Menge C der ausführbaren Kandidaten sowie der Zustand der Quellen und Senken aktualisiert.

Die Einschränkungen des Traveling-Salesman-Problems (TSP) gelten ebenfalls für das hier zu lösende Problem der Sequenzierung (STCP). Jeder Auftrag hat genau einen Vorgänger und einen Nachfolger und eine Lösung darf keine Subtouren enthalten. Ein Auftrag *j* darf nur dann ausgeführt werden, wenn das Ziel d_j nicht vollständig belegt ist. Falls das Ziel für den Auftrag j nicht vollständig gefüllt ist, wird kd_j eins, sonst null. Die letzte Restriktion erfordert, dass die Bearbeitungszeiten an den Kommissionerstationen, den Übergabepuffern und dem Systemausgang mit in das Modell einzubeziehen sind. Aus diesem Grund werden die Zustände der Quellen und der Senken laufend aktualisiert. Die Gleichungen 2-5 stellen die Restriktionen des STCP dar.

$$\sum_{\substack{i \in C \\ i \neq j}} x_{ij} = 1 \qquad j \in C \tag{2}$$

$$\sum_{\substack{j \in C \\ j \neq i}} x_{ij} = 1 \qquad i \in C \tag{3}$$

$$\sum_{i \in U} \sum_{j \in U} x_{ij} \leq |U| - 1 \quad \forall U \subset C, \quad 2 \leq |U| \leq |C| - 2 \tag{4}$$

$$x_{ij} - kd_j \leq 0 \qquad \forall i, j \in C, i \neq j \tag{5}$$

$$x_{ij} = 0 \; or \; 1$$

4. Literatur Überblick

In [Gagné01], [Dorigo96], [Dorigo97], [deJong01] werden Ant-Colony-Algorithmen (ACO) beschrieben. In [Dorigo96] beschreiben die Autoren den Ant-System-Algorithmus (AS) und wenden ihn zur Lösung des Traveling-Salesman-Problems (TSP) an. Der Ant-System-Algorithmus arbeitet ähnlich dem Verhalten realer Ameisen. Ameisen kommunizieren über Pheromone. Das sind Stoffe, die sie in unterschiedlicher Intensität auf den verschiedenen Wegen zwischen Nest und Futterstelle hinterlassen. Wenn mehr Ameisen den gleichen Weg nutzen, steigt die Pheromon-Intensität. Wenn auf dem Weg ein Hindernis erscheint, werden die Ameisen unterschiedliche Wege zur Futterquelle finden. Die Ameisen, die den kürzesten Pfad nutzen, werden schneller auf die Pheromon-Spur zurückkehren. Schließlich wird, unabhängig vom ersten gefundenen Pfad, der bevorzugte Pfad vom Nest zur Futterquelle zum kürzesten Pfad tendieren [Dorigo96]. Die Autoren testen die drei folgenden Versionen von Ant-Systemen: Ant-Density, Ant-Quantity und Ant-Cycle. Die Experimente zeigen die Überlegenheit der Ant-Cycle-Version, da dieser globale Informationen für die Aktualisierung der Pheromon-Dichte nutzt, während die beiden anderen Versionen nur auf lokaler Information basieren. Die Experimente zeigen, dass der Algorithmus mit kommunizierenden Ameisen effektiver arbeitet als mit nicht kommunizierenden Ameisen. Der Synergieeffekt ist optimal, wenn die Anzahl der Ameisen etwa der Anzahl der Städte entspricht. Vergleiche von anderen spezialisierten TSP-Algorithmen mit Anwendung auf das 30-Städte-Problem zeigen, dass Ant-Cycle eine bessere Performance aufweist, auch wenn die anderen Algorithmen 2-Opt-Verfahren nutzen. Die erwähnten Algorithmen erreichen nur dann die Qualität von Ant-Cycle, wenn sie die Lin-Kernighan Heuristik nutzen. Auf das gleiche 30-Städte-Problem wurden Simulated Annealing, Ant-Cycle und Tabu Search mit einer Laufzeit von einer Stunde angewendet. Simulated Annealing berechnete ein nur geringfügig schlechteres Ergebnis als die anderen beiden.
In [Gagné01] setzen die Autoren ACO-Algorithmen zur Lösung des Ein-Maschinen-Scheduling-Problems ein. Es gibt sequenz-abhängige Vorbereitungszeiten; die Zielfunktion minimiert die Verspätungszeiten. Das Verfahren enthält eine vorausschauende Abschätzung um die Güte der Teillösungen abzuschätzen.
In [Dorigo97] beschreiben die Autoren einen verbesserten ACO-Algorithmus, den Ant-Colony-System (ACS) Algorithmus. In AS und ACS Algorithmen bilden die Ameisen jeweils unterschiedliche Touren. Jede Ameise wählt die nächste Stadt, um ihre eigene Tour, die von der Pheromon-Dichte der Kanten und von der Distanz der Städte abhängt, einzufügen. Ameisen bevorzugen Städte, die über kurze Wege mit einer hohen Pheromon-Dichte miteinander verbunden sind. Ein Zyklus endet, wenn alle Ameisen ihre Tour beendet haben. Ein Zyklus endet, wenn alle Ameisen ihre Tour beendet haben. ACS beinhaltet im Gegensatz zu AS eine lokale Aktualisierung der Pheromon-Dichte der Kanten. Diese Aktualisierung erfolgt immer, wenn eine Kante durchlaufen wurde. Durch eine temporäre Reduzierung der Spurdichte (Verdunstung von Pheromon) werden die nachfolgenden Ameisen ermutigt, auch die Umgebung der bisher besten Lösung zu erkunden. Die globale Aktualisierung erfolgt in ACS nur auf Kanten der kürzesten Tour eines Zyklus, der alle Ameisen umfasst. Verdunstung tritt nur auf allen Kanten auf, die nicht zur besten Tour gehören.

In [deJong01] setzt der Autor ein Multiple-Ant-Colony-System (MACS) zur Lösung eine Bus-Halteproblems (Bus-stop Allocation Problem, BAP) ein. Im BAP müssen alle Busse den Hauptbahnhof anfahren. Das Problem ist die Aufteilung der verbleibenden n-1 Haltestellen auf m Buslinien. Auf jeder Buslinie starten zwei Busse – beginnend an den beiden Endpunkten – in unterschiedlichen Richtungen. Das Ziel ist die Minimierung der mittleren Reisezeit über alle Passagiere, die an den unterschiedlichen Haltestellen warten.
Jede Buslinie wird durch ein eigenes Ant-Colony-System (ACS) repräsentiert. In diesen m Ant-Colony-Systemen werden die Pheromon-Spuren der einzelnen Kolonien getrennt aktualisiert. Jede Kolonie besteht aus r Ameisen. Alle Ameisen mit der gleichen Ordnungsnummer bilden kolonieübergreifend eine Gruppe und tragen gemeinsam zu einer Lösung bei. Die numerischen Beispiele zeigen, dass ein Multiple-Ant-Colony-System bessere Ergebnisse liefert, als Greedy- und Simulated-Annealing-Algorithmen. In dieser Veröffentlichung sind die Wege in beiden Richtungen befahrbar, so dass keine Konflikte zwischen den Bussen auftreten können.

5. Ant-Colony-System Algorithmus für einen Einzelwagen

Die Einzelschritte des ACO Algorithmus für die STCP-Lösung für einen Einzelwagen sind in Tabelle 1 zusammengefasst. Dieser Algorithmus weist gegenüber dem Ant-Colony-System nach [Gagné01], [Dorigo96], [Dorigo97], [deJong01] einige Änderungen auf. Für STCP sind die Aufträge analog zu den Wegen zwischen den Städten im TCP. Wenn kein ausführbarer Auftrag aus der Kandidatenmenge C mehr vorliegt, weil das Auftragsziel belegt ist, blockiert der Wagen. Diese Blockade dauert an, bis ein Auftragsziel bedient wird, so dass es durch den Wagen angefahren werden kann. Durch die Blockadezeit wird die Zyklusdauer größer als die Gesamtzeit, in der alle Ameisen ihre Tour vervollständigen können. Der Startpunkt für alle Ameisen wird nicht zufällig gewählt sondern er entspricht der initial gewählten Startposition. Aus diesem Grund ist es erforderlich, in der trail intensity matrix eine Zeile τ hinzuzufügen, die das Verkehrsaufkommen zwischen der initialen Position des Wagens und der Quellposition des ersten mögliche Auftrags einer Tour. Tabuk ist die aktuelle Tour für die Ameise k. Wenn Ameise k einen Auftrag bearbeitet, wird dieser Auftrag in die Tabuk –Liste eingetragen und aus der Warteschlange der Auftragsquelle entfernt. Die führenden Elemente der Warteschlangen der Quellpositionen sind ausführbare Aufträge und sie bilden über alle möglichen Quellpositionen die Menge C der Auftragskandidaten. Die Entscheidung zwischen der Erkundung neuer Lösungen (Exploration) und Nutzung bekannter Lösungen (Exploitation) wird durch den Wert des Parameters q0 bestimmt. Bis jetzt werden keine lokalen Optmierungsverfahren wie die 3-Opt-Methode in unserem Algorithmus genutzt. Die Variable ηij repräsentiert die Sichtbarkeit der Spur. Sie ist als Kehrwert der Zeit definiert, die die unmittelbare Bearbeitung des Auftrages j nach Auftrag i benötigt. Die Bearbeitungszeit eines Auftrages ist die Summe aus der Leerfahrt-, Lastfahrt-, Lastaufnahme- und Lastabgabezeit. Für den Wagen wird eine mittlere Geschwindigkeit von 1m/s angenommen. Die Wichtung der Intensitätsmatrix ist α, die Wichtung der Sichtbarbeit ist β. Die globale und die lokale Verdunstungsfaktoren werden mit ρg und ρl bezeichnet.

Tabelle 1: Ant-Colony-System-Algorithmus für einen Wagen

Schritt 1: Intialisierung
t=0 (t ist eine Laufvariable, die nach jedem globalen Update der Dichte-Matrix erhöht wird.) ; num_cycles=0;

Für jede Kante (i,j) initialisiere die Intensitäts-Matrix:
  wenn der Auftrag i vor dem Auftrag j bearbeitet werden kann $\tau_{ij}(0) = \tau_o = (nL)^{-1}$
  sonst $\tau_{ij}(0)=0$.
Dabei ist n die Anzahl der Aufträge und L ist die Länge der Sequenz, die sich aus dem Greedy-Algorithmus ergibt.

Für alle k Ameisen
  kopiere die initialen Quell- und Ziel-Stati,
  set current_time[k]=ant_step[k]= 0
  ant_finish[k] = false.

Schritt 2: Start
Für jede Ameise k:
  Setze die Ameise k auf die Startposition des Wagens

Schritt 3: Erzeuge eine Sequenz für jede Ameise
Solange nicht alle Ameisen beendet haben
  Für alle Ameisen k = 1 to num_ants
    Wenn not ant_finish[k]

ant_step[k] = ant_step[k] +1;
Wähle einen Auftrag j, $j \notin Tabu_k$ aus den l möglichen Kandidaten, der dem Auftrag i folgt, so dass:

$$j = \begin{cases} \arg\max_{l \notin tabu\ k} \left([\tau_{il}(t)]^{\alpha} [\eta_{il}]^{\beta} \right) & if \quad q \le q_0 \text{ (q ist eine Zufallszahl)} \\ J & if\ q > q_0 \text{ (Exploration)} \end{cases} \tag{6}$$

mit der Spurdichte $\tau_{il}(t)$ zwischen den Aufträgen i und l für die Iteration t . J ist eine Zufallszahl.
Die Wahrscheinlichkeit, das der Auftrag j gewählt wird ist

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} [\eta_{ij}]^{\beta}}{\sum_{l \in Tabu\ k} [\tau_{il}(t)]^{\alpha} [\eta_{il}]^{\beta}} \tag{7}$$

Prüfe, ob eine Blockierung auftritt und aktualisiere current_time[k], Quellen und Senken
Wenn keine Blockierung vorliegt, führe ein local update auf dem gewählten Teilweg (i,j) aus:

$$\tau_{ij}(t) = \rho_l \tau_{ij}(t) + (1 - \rho_l) \Delta \tau_{ij} \quad mit\ \Delta \tau_{ij} = \tau_0 \tag{8}$$

Wenn Ameise k die Tour beendet hat, setze ant_ finish[k] = true
Prüfe, ob alle Ameisen ihre Tour beendet haben.

Schritt 4: Globale Aktualisierung
Berechne die Länge der Tour L_k für jede Ameise k. Aktualisiere für jeden Teilweg (i,j) der optimalen Tour die Spur-Intensität:

$$\tau_{ij}(t+1) = \rho_g \tau_{ij}(t) + (1 - \rho_g) \Delta \tau_{ij} \quad mit\ \Delta \tau_{ij} = \frac{1}{L^*} \quad und\ L^* = Längeder optimalen Tour in Zyklus t \tag{9}$$

t = t+1;

Schritt 5: Abbruchbedingungen
Solange (num_cycles < max_cycles) und (keine Stagnation)
num_cycles = num_cycles +1
Für alle Ameisen k
current_time[k] =0, ant_step[k]=0, ant_finish[k]=false
Initialisiere alle Ameisen durch Clonen der Initialzustände der Quelle und Senken
Weiter mit Schritt 2

Zu Bewertung werden die gesamte Weglänge und – falls vorhanden – die Blockierungszeiten berücksichtigt. Das Resultat ist die tatsächlich benötigte Zeit für die Auftragsausführung. Jeder Lauf basiert auf folgenden Parametern, die zu Beginn vorgegeben werden;

- Anzahl der Anforderungen von den unterschiedlichen Quellen,
- Ziele für die einzelnen Aufträge,
- Zielbelegung,
- Startposition des Wagens und
- Bearbeitungszeiten an den Zielpositionen.

Es wird eine zufällig gewählte Zahl von Aufträgen mit jeweils unterschiedlichen Quell- und Zielpositionen im Intervall Null bis zur Maximalkapazität der Quellen und der Senken mit einer Gleichverteilung erzeugt. Das Ziel wird entsprechend den Werten einer Transportmatrix gewählt. Die Bearbeitungszeiten an den Kommissionierstationen und den Puffern wird einmal gleichverteilt im Intervall [30s, 60s] und einmal konstant mit 30sec angenommen.
Für den Algorithmus wurden folgende Parameter gesetzt. Die Anzahl der Ameisen wurde auf die jeweilige Anzahl der Aufträge gesetzt. Die Werte für α, β, ρ_{local}, and $\rho_{lglobal}$ wurden auf 1, 5, 0.9 und 0.9 gesetzt. Für q_0 wurden die Werte 0, 0.5 und 1.0 eingesetzt. Die Maimalzahl der Zyklen für den Algorithmus wurde auf 5000 gesetzt. Bereits nach 50 Zyklen wurde keine Verbesserung des Ergebnisses erreicht.

6. Nummerische Vergleiche: Greedy- und ACO-Algorithmus

Der ACO-Algorithmus wurde mit UML (Unified Markup Language) designed und in Java mit der Entwicklungsumgebung Forte 3.0 programmiert. Der Greedy-Algorithmus wurde ebenfalls in Java codiert. Zunächst wurde das Greedy-Verfahren implementiert, da es einfach ist und schnelle Laufzeiten erwarten lässt. Die Ergebnisse wurden mit dem bisher verwendeten FIFO-Verfahren verglichen. Nach der Implementierung des Greedy-Verfahrens wurde der ACO-Algorithmus in Bezug auf Ergebnis-Qualität und Laufzeit untersucht.

Der Greedy-Algorithmus nimmt in die Kandidatenmenge das erste Element jeder Quelle auf. Dann wird unter diesen Kandidaten derjenige ausgewählt, zu den der Wagen von seinem aktuellen Standort aus die kürzeste Leerfahrt hat. Die Gesamtzeit setzt sich aus der Zeit für die Leerfahrt zur Quelle und der Zeit für Lastfahrt zum Ziel zusammen. Das Verfahren terminiert, wenn alle Aufträge abgearbeitet sind. Abbildung 12 zeigt Mittelwerte für die Bearbeitungszeit aller Aufträge unter Einsatz des ACO-Algorithmus für 50% und 100% Exploration-Anteil und des Greedy-Algorithmus. Der Greedy-Algorithmus ist – insbesondere im Vergleich zur Ersparnis an Ausführungszeit – unwesentlich schneller als ACO. Das ACO-Verfahren mit 50% Exploration liefert das beste Ergebnis.

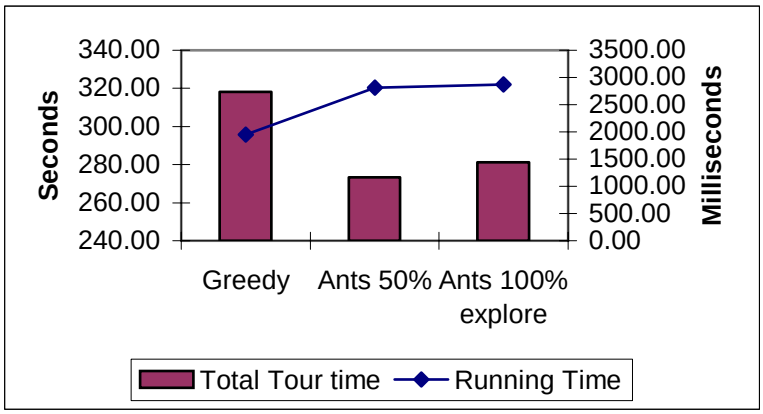

Abbildung 2: Mittlere Bearbeitungszeit und Rechenzeit für ACO- und Greedy-Algorithmus.

Tabelle 2 zeigt die Ergebnisse für die Läufe mit den beiden Verfahren im Detail. Die ACO-Ergebnisse gelten für 50% Exploration. Alle Quellen sind vollständig gefüllt, was 25 Aufträgen entspricht. Die Bearbeitungszeiten an den Kommissionierstationen und den Puffern werden als konstant angenommen.

7. Erweiterung des ACS-Algorithmus für zwei Wagen

Es werden zwei Kolonien mit jeweils k Ameisen eingesetzt. Je zwei Ameisen mit gleicher Ordnungszahl aus unterschiedlichen Kolonien bilden ein Paar, um das Problem gemeinsam zu lösen. Es werden zwei separate Matrizen für die Intensität der Pheromon-Spuren geführt. Diese Matrizen werden nach jeder Iteration t.aktualisiert. Wenn die Kolonie eins den Auftrag j als Nachfolger für den Auftrag i unter l Kandidaten wählt, muss (6) wie folgt angepasst werden:

$$j = \begin{cases} \underset{l \notin (tabuk^1 \cup tabuk^2)}{\arg\max} \left\{ \left[\tau_{il}^1(t)\right]^{\alpha} \left[\eta_{il}\right]^{\beta} \left[\psi_{il}^1\right]^{\chi} \right\} & if \quad q \le q_o \\ J \qquad if\ q > q_o \end{cases} \tag{10}$$

J wird zufällig gewählt. Die Wahrscheinlichkeit für die Auswahl des Auftrages j wird durch (11) beschrieben:

$$p_{ij}^{k^1}(t) = \frac{\left[\tau_{ij}^1(t)\right]^{\alpha} \left[\eta_{ij}\right]^{\beta} \left[\psi_{ij}^1\right]^{\chi}}{\sum_{l \notin (Tabu\,k^1 \cup Tabu\,k^2)} \left[\tau_{il}^1(t)\right]^{\alpha} \left[\eta_{il}\right]^{\beta} \left[\psi_{il}^1\right]^{\chi}} \tag{11}$$

In den Gleichungen (10) und (11) ist ψ^1_{ij} ein Maß für die Durchführbarkeit des Auftrages j nach dem Auftrag i durch eine Ameise k aus Kolonie eins wenn die entsprechende Ameise der Kolonie zwei mit der gleichen Ordnungsnummer einen Auftrag ausführt oder im Zustand Idle ist. Um mögliche Konflikte zu erkennen, werden die Pfadnummer und die Bearbeitungszeit der beiden Ameisen eines Paares verglichen. Zur Vermeidung von Konflikten auf Haltepositionen kann die Einführung von Idle-Zeiten erforderlich werden. In Gleichung (11) stellt χ das Gewicht für ψ^1_{ij} dar. Der Wert χ wurde mit 1.0 angenommen. Numerische Rechnungen zeigen, dass ACS bessere Ergebnisse liefert als ein Greedy-Verfahren. Die Rechenzeit erhöht sich jedoch auf bis zu 90 Sekunden.

Tabelle 2: ACO Ergebnisse für einen Querverteil-Wagen

	ACO Algorithm				Greedy Algorithm			
Run	Number of cycles	Tour length Seconds	Blocked time Seconds	Total time Seconds	Tour Length	Blocked time Seconds	Total time Seconds	% Decrease in total time
1	55	240.50	0.00	240.50	285.90	0.00	285.90	15.87
2	57	268.80	7.50	276.30	321.30	0.00	321.30	14.01
3	55	272.90	0.00	272.90	318.30	0.00	318.30	14.26
4	55	252.00	0.00	252.00	280.60	0.00	280.60	10.19
5	55	267.90	1.30	269.20	327.90	0.00	327.90	17.90
6	55	333.00	0.00	333.00	392.40	0.00	392.40	15.14
7	53	306.10	0.00	306.10	311.90	0.00	311.90	1.86
8	56	256.50	0.00	256.50	299.30	0.00	299.30	14.30
9	56	307.00	1.20	308.20	346.80	0.00	346.80	11.13
10	61	218.90	0.00	218.90	297.00	0.00	297.00	26.30
Average	55.80	272.36	1.00	273.36	318.14	0.00	318.14	14.08
Standard deviation	2.10	34.30	2.34	34.41	32.88	0.00	32.88	6.16
Coef. of variation	3.76%	12.59%	234.2%	12.59%	10.34%	-	10.34	43.69%

8. Fazit

Die Erweiterung des Ant-Colony-Algorithmus nach [Gagné01], [Dorigo96], [Dorigo97], [deJong01] wurde zur Lösung des STCP-Problems mit einem und zwei Wagen auf einer Schiene erweitert. Das Ergebnis zeigt, das der ACO-Algorithmus eine viel versprechende Technik zur Auftrags-Sequenzierung im Lagerbereich darstellt. Dieser Beitrag sollte die Anwendung und die Erweiterung des ACO-Algorithmus auf andere Anwendungsgebiete zeigen. Schließlich bestätigt diese Arbeit die Vorteile der positiven Rückkopplung und der Synergie, die das ACO-Verfahren charakterisieren. Weitere Arbeiten sollten andere Optimierungsverfahren für diese Anwendung implementieren und numerische Vergleiche durchführen.

Literatur

[Gagné01] Gagné, Caroline; Price, Wilson L.; Gravel, Marc: Scheduling a single machine with sequence dependent setup times using ant colony optimization," Faculté des Sciences de l'administration, Université Laval, Canada. S. 1-21. [http://wwwdim.uqac.ca/~c3gagne/DocumentRech/ANTS2000_SchedulingSingleMachine.pdf - Stand 04/2005]

[deJong01] Jong de, Jasper; Wiering, Marco: Multiple ant-colony systems for the Busstop Allocation Problem. University of Utrecht, IN: Cognitive Artificial Intelligence (2001), S. 1-8 [http://www.cs.uu.nl/groups/IS/studprojects/thesis_jasper.pdf – Stand 04/2005]

[Dorigo96] Dorigo, Marco; Maniezzo, Vittorio; Colorni, Alberto: The ant system: Optimization by a colony of cooperating agents. IN: IEEE Transactions on Systems, Man and Cybernetics– Part B, 26(1996)1, S. 1-26.

[Dorigo97] Dorigo, Marco; Gambardella, Luca M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IN: IEEE Trans. Evol. Comp., 1(1997)1, S. 53-66.