DOI 10.2195/LJ_proc_tenhompel_de_201108_02

Realtime Logistics – Ein Algorithmus zur parallelisierbaren Bestimmung von Transportkollisionen in Materialflusssystemen

Prof. Dr. Michael ten Hompel, Dipl.-Inform. Moritz Roidl, Jan Sören Emmerich
TU Dortmund
Lehrstuhl für Förder- und Lagerwesen

Abstract: In diesem Beitrag zur Komplexitätsanalyse von Materialflusssystemen unter Zeitrestriktionen wird ein Algorithmus vorgestellt, der die Bestimmung von Transportkollisionen als parallelisierbares Problem betrachtet und dessen Datenstrukturen auf die Analyse der Wechselbeziehungen von Lastobjekten ausgerichtet ist. Am Beispiel eines Deadlockszenarios wird die Funktionsweise des Algorithmus dargestellt und gezeigt, dass die explizite Betrachtung von zeitlichen und räumlichen Abhängigkeiten unter Lastobjekten eine Deadlockerkennung möglich macht. Der Algorithmus bildet die Grundlage für weitere Anwendungen in der Analyse der Echtzeitfähigkeit von Materialflusssystemen.

1 Einleitung

Die Analyse des Verhaltens von Materialflusssystemen unter Zeitrestriktionen bedarf neben einer Betrachtung des Zeitverhaltens der Förderelemente auch einer detaillierten Betrachtung der Wechselbeziehungen zwischen einzelnen Lastobjekten. In diesem Beitrag wird ein Algorithmus vorgestellt, der die Bestimmung von Transportkollisionen als parallelisierbares Problem betrachtet und dessen Datenmodell auf die Analyse der Wechselbeziehungen von Lastobjekten ausgerichtet ist. „Parallelisierbar“ bedeutet hier, dass die Lastobjektberechnungen in beliebiger Reihenfolge ohne globale zeitliche Synchronisation und daher auch verteilt durchgeführt werden können. Abhängigkeiten in Form von Transportkollisionen werden erst bei Bedarf berechnet und im Datenmodell explizit gespeichert. Dies ermöglicht die einfache Zurückverfolgung kausaler Zusammenhänge. So ist beispielsweise die Bestimmung möglich, welche Last den Stau oder Deadlock, etc. ausgelöst hat.

Der hier beschriebene Ansatz lässt sich in verschiedener Hinsicht mit existierenden Arbeiten vergleichen. Er basiert auf demselben Modellierungsgrad, den ereignisdiskrete Simulationssoftware in der Intralogistik verwendet. Insbesondere die genaue Abbildung der Reihenfolge von einzelnen Lastbewegungen kann als Unterscheidungsmerkmal gegenüber analytischen Ansätzen oder Ansätzen wie der mesoskopischen Simulation herangezogen werden (vgl. [Sch08a]). Im Gegensatz zur stochastischen Analyse von Systemen, wie der Warteschlangentheorie, erfordert der Algorithmus keine speziellen Voraussetzungen und Annahmen, sondern versucht

das Systemverhalten realistisch und bei einer festgelegten Last abzubilden (vgl. [Gud05], [Arn07], [Sch08b]).

Von Ansätzen, die auf ereignisorientierten Simulationswerkzeugen basieren (z.B. [Lie09]), unterscheidet sich der Algorithmus insofern, dass er nicht schrittweise mit einer globalen Zeitvariablen arbeitet. Er ist vergleichbar mit Ansätzen, die zunächst simulieren, ein Ablaufprotokoll erstellen und dieses dann anschließend analysieren (vgl. [Wus10]). Jedoch besteht hier der Unterschied, dass mit dem vorgestellten Algorithmus nachträglich und ohne vollständige Neuberechnung einzelne Ereignisse im Sinne einer höheren Experimentierfähigkeit verändert werden können.

Im Vergleich zu Arbeiten, die sich thematisch mit Deadlockvermeidung in der Praxis befassen, wie zum Beispiel bei [May09], steht bei dem hier gewählten Ansatz kein spezielles Fördersystem im Fokus.

Um einen Bezug zu realen Systemen herstellen zu können, werden Modelle und Lastdateien der Simulationssoftware AutoMod als Basis verwendet. Zusätzlich durchgeführte Simulationsexperimente können daher auch zu einer Validierung des Algorithmus dienen. Für die gewählte Anwendung wurde das Modell funktional sowie bei der Auswahl der möglichen Fördertechniken eingeschränkt. Es werden nur spurgebundene Fördersysteme betrachtet. Spezielle Regelungs- und Steuerungslogik wird ebenfalls nicht betrachtet.

2 Explizite Abbildung von Abhängigkeiten

Die Grundidee zu diesem Algorithmus besteht darin, die wechselseitigen Abhängigkeiten von Lastobjekten explizit als Datenstrukturen zu speichern und somit der einfachen Analyse zugänglich zu machen.

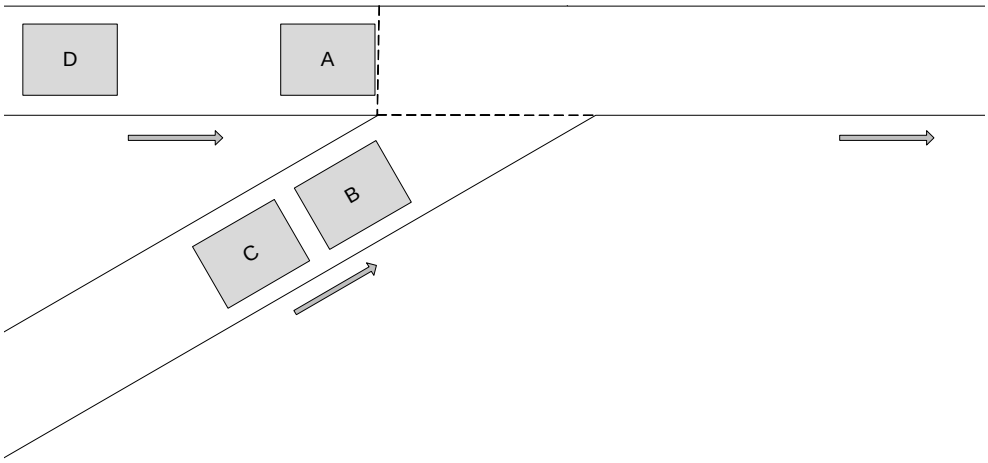

Abbildung 1: Lastsituation vor einer Zusammenführung. Bisher sind keine Abhängigkeiten zwischen den eintreffenden Lastobjekten vorhanden.

Das grundlegende Szenario wird in [Gud05] beschrieben: „Wenn die Gesamtbelastung einer Station die Belastungsgrenze erreicht oder überschreitet, kommt es vor den Einlaufpunkten zu Wartezeiten, Warteschlangen und Rückstaus,

die voranliegende Stationen blockieren können.“ Dazu kommen so genannte „stochastische“ Staus, die dann entstehen, wenn die Gesamtbelastung noch nicht erreicht wurde. Diese werden durch ungünstig eintreffende Lastkonfigurationen ausgelöst, die zu Staus und Deadlocks führen können.

Den grundsätzlichen Ursprung eines „stochastischen“ Staus beschreibt die Situation in Abbildung 1. Hier bewegen sich vier Lastobjekte auf eine Zusammenführung zu. Da jedoch die Ankünfte, wie aus der Abbildung ersichtlich, zu einer Kollision führen würden, muss Vorfahrt gewährt werden.

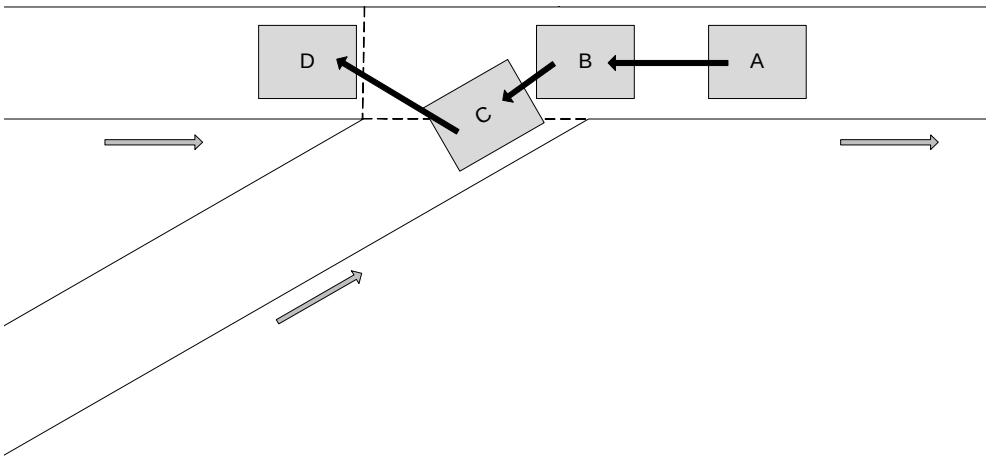

Abbildung 2: Lastsituation nach erfolgter Zusammenführung durch das FIFO-Prinzip. Das Lastobjekt, das Vorfahrt gewährt, gilt als abhängig gegenüber dem Lastobjekt, das weiterfährt.

Abbildung 2 zeigt die Situation, nachdem die Zusammenführung nach dem FIFO-Prinzip erfolgt ist. Zu beachten ist, dass einseitige Abhängigkeiten entstanden sind, die beschreiben welche Last direkt auf eine andere Last gewartet hat. Die Verkettung der Abhängigkeitsbeziehungen beschreibt indirekte Abhängigkeiten, wie sie in diesem Beispiel zwischen Last A und Last D bestehen.

3 Beispielszenario mit Deadlockerkennung

In diesem Beitrag wird die Arbeitsweise des Algorithmus anhand eines Beispiels erläutert und mit der ereignisorientierten Vorgehensweise, die in den meisten Materialflusssimulatoren eingesetzt wird, verglichen. Dabei steht nicht die Realitätsnähe im Vordergrund, sondern die einfache Nachvollziehbarkeit. Um dies zu erreichen, werden mehrere vereinfachende Annahmen getroffen, die eine Simulation auf dem Papier ermöglichen. Für diesen Vergleich wurde ein einfaches, einheitenloses Modell konstruiert, welches aus zwölf Belegungssegmenten aufgebaut ist, die gleichzeitig den Förderelementen entsprechen. Sechs Segmente stellen die Quellen und Senken des Modells dar, die sechs restlichen Segmente bilden den Sortierkreislauf (siehe Abbildung 3).

Die Geschwindigkeit ist so gewählt, dass ein Lastobjekt zwischen den Mittelpunkten der Segmente immer genau zwei Zeiteinheiten benötigt. Ein Lastobjekt hat dabei

genau die Länge eines Förderelements, belegt daher zur gleichen Zeit maximal zwei Belegungssegmente und minimal eines.

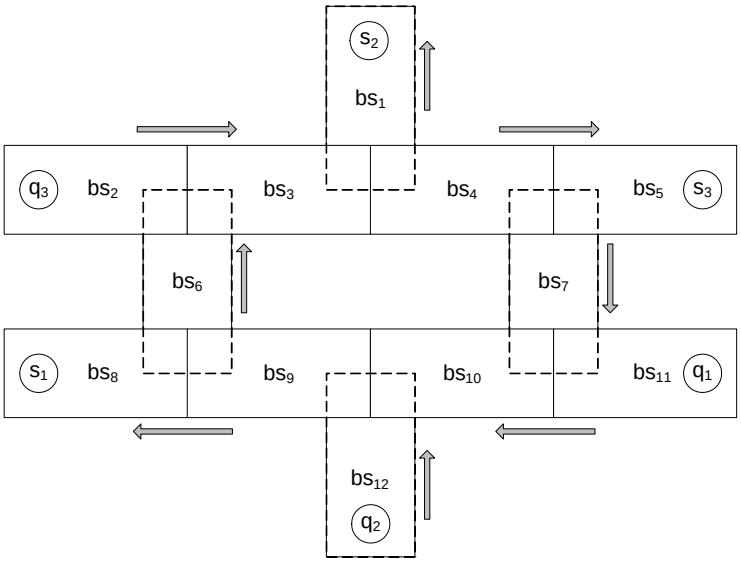

Abbildung 3: Schematisch dargestellter Sortierkreislauf mit drei Quellen und drei Senken zur Demonstration der Arbeitsweise des Algorithmus und zum Vergleich mit der ereignisorientierten Herangehensweise einer Simulation.

An den Zusammenführungen wird bei der Durchführung der ereignisorientierten Simulation sowie bei den Berechnungen des Algorithmus das FIFO-Prinzip angewandt.

Transport-auftrag	Eingangszeit-punkt	Quelle	Senke	Transportpfad
a	3	q_3	s_1	$(bs_2,bs_3,bs_4,bs_7,bs_{10},bs_9,bs_8)$
b	5	q_3	s_1	$(bs_2,bs_3,bs_4,bs_7,bs_{10},bs_9,bs_8)$
c	3	q_2	s_3	$(bs_{12},bs_9,bs_6,bs_3,bs_4,bs_5)$
d	1	q_1	s_3	$(bs_{11},bs_{10},bs_9,bs_6,bs_3,bs_4,bs_5)$
e	5	q_2	s_2	$(bs_{12},bs_9,bs_6,bs_3,bs_1)$
f	3	q_1	s_2	$(bs_{11},bs_{10},bs_9,bs_6,bs_1)$

Tabelle 1: Transportaufträge für das Beispielszenario

Die Transportaufträge setzen sich zusammen aus einem Lastobjekt, einer Eingangszeit, Quelle, Senke, einem Transportpfad und einem Entscheidungskennwert (siehe Tabelle 1). Der Entscheidungskennwert ist in diesem Beispiel die Identifikation des Transportauftrags. Es gilt die Ordnung: a < b < c < d < e < f. Die Versuchsanordnung ermöglicht eine schrittweise, synchrone Berechnung des Algorithmus und eine ereignisorientierte Simulation.

3.1 Ereignisorientierte Simulation

Bei der ereignisorientierten Simulation erfolgt der Fortschritt durch Abarbeiten von Ereignissen aus einer Ereignisliste. In einem ersten Schritt werden alle begonnenen Bewegungen vollendet. Dies erfordert im Vorfeld keine weitere Überprüfung, da das

Zielsegment im vorherigen Schritt schon frei sein musste. Ansonsten wäre die Bewegung nicht eingeleitet worden.

	bs_1	bs_2	bs_3	bs_4	bs_5	bs_6	bs_7	bs_8	bs_9	bs_{10}	bs_{11}	bs_{12}
1											c	
2										c_1	c_2	
3		a								c	e	b
4		a_2	a_1						b_1	c	e	b_2
5		f	a						b	c	e	d
6		f	a_2	a_1		b_1			b_2	c	e	d
7		f_2	f_1	a		b			c_1	c_2	e	d
8			f	a_2		b	a_1		c	e_1	e_2	d
9			f_2	f_1		b	a		c	e		d
10			b_1	f		b_2	a		c	e		d
11			b	f		c_1	a		c_2	e		d
12			b	f		c	a		d_1	e		d_2
13			b	f		c	a		d	e		
14			b	f		c	a		d	e		

Tabelle 2: Positionstabelle der ereignisorientierten Simulation mit Deadlockzustand ab Zeitschritt 13

In der Positionstabelle (siehe Tabelle 2) ist der noch nicht abgeschlossene Übergang eines Lastobjekts daran zu erkennen, dass zwei Segmente zur selben Zeit belegt werden. Der Kopf des Lastobjekts ist mit dem Index 1 versehen, das Ende mit dem Index 2. Die Ereignisroutine belegt das Segment, das im vorherigen Schritt mit dem Index 1 versehen war. Im nächsten Schritt markiert die Routine für jedes Lastobjekt, ausgehend vom derzeitigen Standpunkt im System, das nächste Belegungssegment im jeweiligen Transportpfad. Wenn dieser Teil der Ereignisroutine bei allen aktiven Transportaufträgen abgeschlossen ist, können die Reservierungswünsche, deren Zielsegment nur eine Markierung hat, ausgeführt werden.

Falls mehrere Markierungen für ein Segment vorliegen, muss entschieden werden welcher Transportauftrag Vorrang hat. Die Entscheidung fällt auf den Transportauftrag, der die größte Wartezeit auf ein freies Zeitintervall hat.

Die Information, wie lange ein Lastobjekt auf die Weiterfahrt warten musste, wird in diesem Fall in einer separaten Datenstruktur gespeichert. In einem ereignisorientierten Simulator müssen für die Aufnahme aller Informationen aus Zuständen, die nicht im aktuellen Berechnungszeitpunkt anfallen, Datenstrukturen zum Speichern vorhanden sein. Für den Fall, dass mehrere Aufträge ein Segment reservieren wollen, und diese die gleiche Wartezeit aufweisen, wird die Entscheidung aufgrund der Entscheidungskennzahl gefällt.

Die Ereignisliste wird abgearbeitet bis entweder alle Transportaufträge ausgeführt werden konnten und die Lastobjekte ihre Senken erreicht haben, oder zwei Zeitschritte hintereinander der gleiche Systemzustand vorliegt und damit ein Deadlockzustand besteht (siehe Abbildung 4).

In kleinen Systemen, wie in diesem Beispiel, lässt sich solch ein Zustand leicht erkennen. Anders verhält es sich in einem System, in dem mehrere tausend Lastobjekte gleichzeitig unterwegs sind und nur einige wenige sich in einem Deadlockzustand befinden. In solchen Fällen kann eine Erkennung praktisch nur durch Heuristiken erfolgen. Der schwerwiegendere Nachteil ist, dass wenig Informationen über das Zustandekommen des erreichten Zustands vorhanden sind, so dass bei einem erforderlichen Neustart der Simulation nicht klar ist, welche Änderungen an der Lastdatei nötig sind, um eine erneute Deadlocksituation zu verhindern.

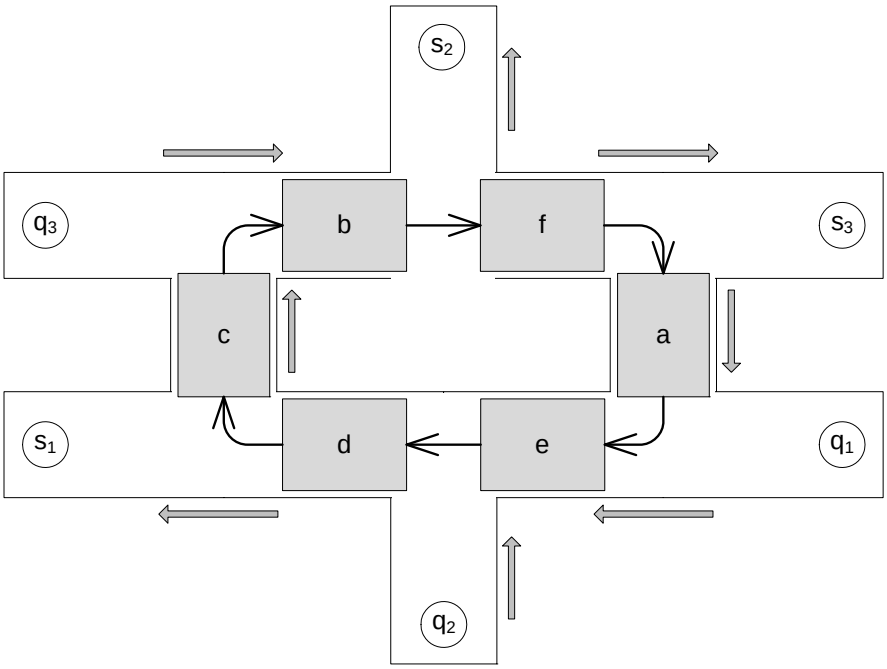

Abbildung 4: Der Sortierkreislauf hat einen Deadlockzustand erreicht.

3.2 Beschreibung des Algorithmus

Um die vorgestellte ereignisorientierte Vorgehensweise mit der des entwickelten Algorithmus vergleichen zu können, werden die gleichen Modelldaten, Parameter und Lastdaten verwendet (siehe Tabelle 1). Da die Transportaufträge, anders als beim vorherigen Beispiel, nicht parallel in das System eintreten, muss die Reihenfolge der noch ausstehenden Aufträge verwaltet werden. Dazu wird ein Kellerspeicher $A_{ausstehend}$ verwendet. Die Menge $A_{bearbeitet}$ enthält die aktuell bereits berechneten Aufträge.

Pro Transportauftrag werden zwei zusätzliche Datenstrukturen verwaltet. Alle Transportaufträge die im Laufe einer Berechnung von einem Auftrag ***a*** blockiert wurden, also mindestens einen Zeitschritt warten mussten, werden Teil der Menge $A_{nachfolger}(a)$. Diese Strukturen sind notwendig für den Fall, dass der blockierende Auftrag ***a*** gelöscht wird, da dann alle abhängigen Aufträge neu berechnet werden müssen, um wieder einen gültigen Systemzustand herzustellen.

Die zweite für jeden Auftrag *a* verwaltete Menge $A_{vorgänger(a)}$ enthält alle Transportaufträge, welche die eigene Bewegungsausführung bei der Berechnung verzögern. Die Menge $A_{vorgänger(a)}$ wird benötigt, um beim Entfernen eines Transportauftrages *a*, diesen aus allen Mengen $A_{nachfolger}$ zu entfernen, so dass alle Datenstrukturen nach dem Löschvorgang bereinigt sind.

Der Algorithmus beginnt mit der Festlegung einer beliebigen Berechnungsreihenfolge der Transportaufträge, da das Ergebnis der Berechnung davon unabhängig ist. Die Transportaufträge werden zuerst beispielhaft in umgekehrter Reihenfolge im Kellerspeicher $A_{ausstehend}$ abgelegt, so dass in diesem Fall $A_{ausstehend} = (c,a,b,e,f,d)$ ist.

Dementsprechend wird mit der Pfadreservierung für Auftrag *c* begonnen, dessen Lastobjekt auf Segment bs_{12} zum Zeitpunkt 1 in das System eintritt (siehe Tabelle 3).

	bs_1	bs_2	bs_3	bs_4	bs_5	bs_6	bs_7	bs_8	bs_9	bs_{10}	bs_{11}	bs_{12}
1											c	
2										c_1	c_2	
3										c		
4									c_1	c_2		
5									c			
6						c_1			c_2			
7						c						
8			c_1			c_2						
9			c									
10			c_2	c_1								
11				c								
12				c_2	c_1							
13					c							
14												

Tabelle 3: Positionstabelle nach der ersten Iteration mit den Reservierungen des Auftrags c im ansonsten leeren System. $A_{bearbeitet} = \{c\}$, $A_{ausstehend} = (a,b,e,f,d)$

Warten

Bei jedem Eintritt in ein neues Belegungssegment findet eine Überprüfung statt, ob zum Eintrittszeitpunkt oder davor das Segment als frei markiert ist. Ohne einen konkurrierenden Transportauftrag erreicht der Auftrag *c*, im Unterschied zur vorhergegangenen ereignisorientierten Simulation, sein Zielsegment und verlässt im nächsten Schritt an der Senke das System.

Zur genaueren Betrachtung der Datenstrukturen eines Belegungssegments eignet sich ein zum Teil gefülltes System wie es im fünften Iterationsschritt der Fall ist (siehe Tabelle 4). Drei der sechs Aufträge sind berechnet, Auftrag *c* muss aufgrund einer vorangegangenen Verdrängung durch Auftrag *b* ein weiteres Mal berechnet werden. Auftrag *c* versucht ab dem vierten Zeitschritt das Belegungssegment bs_9 zu reservieren. Dieser Schritt soll im Folgenden näher betrachtet werden.

Für jedes Segment existiert eine Reservierungstabelle *R* als Datenstruktur. Die Tabelle R_9 des Segments bs_9 hat zum Zeitpunkt 4 die folgenden Einträge:

Zeitschritt	$-\infty$ #	4	8	12	15
Belegung	*frei*	*b*	*frei*	*a*	*frei*

Dies bedeutet, dass das Segment von den Aufträgen *a* und *b* zu unterschiedlichen Zeitenintervallen belegt und ansonsten frei ist.

	bs_1	bs_2	bs_3	bs_4	bs_5	bs_6	bs_7	bs_8	bs_9	bs_{10}	bs_{11}	bs_{12}
1											*c*	
2										c_1	c_2	
3		*a*								*c*		*b*
4		a_2	a_1						b_1	*c*		b_2
5			*a*						*b*	*c*		
6			a_2	a_1		b_1			b_2	*c*		
7				*a*		*b*			c_1	c_2		
8			b_1	a_2		b_2	a_1		*c*			
9			*b*			c_1	*a*		c_2			
10			b_2	b_1		*c*	a_2			a_1		
11			c_1	*b*		c_2				*a*		
12			*c*	b_2	b_1				a_1	a_2		
13			c_2	c_1	*b*				*a*			
14				*c*				a_1	a_2			
15				c_2	c_1			*a*				
16					*c*							

Tabelle 4: Positionstabelle nach der vierten Iteration. $A_{bearbeitet}$ = {a,b,c}, $A_{ausstehend}$ = (e,f,d), $A_{nachfolger}(b)$ = {c}

Beim Versuch das Segment für Auftrag *c* zu reservieren, sucht der Algorithmus in der Reservierungstabelle R_9 den Zeitpunkt, der kleiner oder gleich dem gewünschten Eintrittszeitpunkt ist. In diesem Fall findet er den Schlüssel 4 mit dem Wert *b*. Das Segment ist also mit dem Auftrag *b* belegt.

Da Auftrag *c* seine Reservierung nicht ohne eine Kollision ausführen kann, ermittelt der Algorithmus die Position des Auftrags, die sich bei Anwendung des FIFO-Prinzips ergibt. Dazu wird Auftrag *c* in eine Tabelle der Ankunftszeiten AZ eingetragen, die zu jedem Segment als Datenstruktur existiert. Die Tabelle AZ_9 am Segment bs_9 hat zum gleichen Zeitpunkt folgenden Inhalt:

Auftrag	*a*	*b*	*c*
Ankunftszeit	*12*	*4*	*4*

Es wird die korrekte Position von *c* in Reservierungstabelle R_9 ermittelt. Die Aufträge *b* und *c* haben identische Ankunftszeiten. Daher wird anhand der Entscheidungskennwerte b und c die Reihenfolge entschieden. Die früheste mögliche Eintrittszeit für *c* ist der Zeitpunkt 7, nachdem *b* das Segment verlassen hat, da b < c.

Transportauftrag *c* reserviert das Vorgängersegment bs_{10} für die Wartezeit, bis die Bewegung nach bs_9 vollständig abgeschlossen ist. Auftrag *c* ist aufgrund der Kollision abhängig von *b*, es gilt: $A_{nachfolger}(b) = \{c\}$ und $A_{vorgänger}(c) = \{b\}$.

Die Reservierungstabelle R_9 des Segments bs_9 hat nach Abschluss der Berechnung von *c* die folgenden Einträge:

Zeitschritt	$-\infty$ #	4	7	10	12	15
Belegung	*frei*	*b*	*c*	*frei*	*a*	*frei*

Verdrängen

Neben der Funktion des Wartens auf ein freies Zeitintervall, besteht die Funktion der Verdrängung anderer Aufträge. Diese Funktion wird dann durchgeführt, wenn ein neu betrachteter Auftrag im Verlauf der Berechnung Vorrang vor einem bereits berechneten Auftrag bekommt. Dies ist möglich, da Aufträge in beliebiger Reihenfolge berechnet werden.

Zur Erläuterung der Verdrängung eignet sich die Berechnung des Transportauftrages *d* in der 7. Iteration des Algorithmus (siehe Tabelle 5)

	bs_1	bs_2	bs_3	bs_4	bs_5	bs_6	bs_7	bs_8	bs_9	bs_{10}	bs_{11}	bs_{12}
1											*c*	
2										c_1	c_2	
3		*a*								*c*	*e*	*b*
4		a_2	a_1						b_1	*c*	*e*	b_2
5			*a*						*b*	*c*	*e*	*d*
6			a_2	a_1		b_1			b_2	*c*	*e*	*d*
7				*a*		*b*			c_1	c_2	*e*	*d*
8			b_1	a_2		b_2	a_1		*c*	e_1	e_2	*d*
9			*b*			c_1	*a*		c_2	*e*		*d*
10			b_2	b_1		*c*	*a*		e_1	e_2		
11			c_1	*b*		c_2	a_2		*e*	a_1		
12			*c*	b_2	b_1	e_1			e_2	*a*		
13			c_2	c_1	*b*	*e*			a_1	a_2		
14			e_1	*c*		e_2			*a*			
15			*e*	c_2	c_1			a_1	a_2			
16	e_1		e_2		*c*			*a*				
17	*e*											

Tabelle 5: Positionstabelle während der 7. Iteration des Algorithmus. $A_{bearbeitet}$ = {b,c,e,a,d}, $A_{ausstehend}$ = (f), $A_{nachfolger}$(b) = {c}, $A_{nachfolger}$(c) = {e,d}, $A_{nachfolger}$(e) = {a}

Das Lastobjekt des Auftrags d tritt in das System auf Belegungssegment bs_{12} ein und reserviert seinem Pfad folgend das Segment bs_9. Auftrag *d* wird vor Auftrag *e* und nach dem Auftrag *c* einsortiert, wie in der Tabelle der Ankunftszeiten AZ_9 zu sehen ist:

Auftrag	*a*	*b*	*c*	*d*	*e*
Ankunftszeit	*12*	*4*	*4*	*6*	*10*

Die Reservierungstabelle R_9 des Segments bs_9 hat zu diesem Zeitpunkt folgende Einträge:

Zeitschritt	$-\infty$ #	4	7	10	12	15
Belegung	*frei*	*b*	*c*	*e*	*a*	*frei*

Da nach Auftrag *c* in R_9 kein freies Zeitintervall vorhanden ist, wird Auftrag *e* verdrängt. In der Menge der von e blockierten Aufträge $A_{nachfolger}(e)$ steht Auftrag a, für den nach dem Entfernen von *e* aus der Positionstabelle keine gültige Berechnung mehr existiert. Daher wird Auftrag *a* ebenfalls aus der Positionstabelle und den Reservierungstabellen der belegten Segmente entfernt.

Beim Wiedereinfügen der gelöschten Aufträge in den Kellerspeicher wird so vorgegangen, dass Auftrag *e* vor Auftrag *a* entnommen werden kann. Da bei einer Neuberechnung der beiden entfernten Aufträge *e* wieder Vorrang vor *a* hat, lässt sich durch eine solche Bearbeitungsreihenfolge eine überflüssige Neuberechnung verhindern.

Die Funktion des Verdrängens von bereits berechneten Aufträgen ermöglicht dem Algorithmus die anfangs beschriebene Parallelisierbarkeit in der Ausführung. Die Reihenfolge, in der Transportaufträge ausgeführt werden, beeinflusst nicht das Ergebnis des Algorithmus.

Erkennung von zyklische Abhängigkeiten

Die anfangs vorgestellte ereignisorientierte Simulation endet nach 13 Zeitschritten in einem Deadlock. Aus diesem Grund muss bei gleichem Model, Lastdaten, Regeln und Parametern auch die algorithmische Berechnung einen Zustand erreichen, der ein erfolgreiches Reservieren des letzten Transportauftrages verhindert. Wie sich eine Deadlocksituation bei der Berechnung äußert, wird anhand der letzten Positionstabelle (siehe Tabelle 6) erläutert.

Der letzte noch verbleibende Transportauftrag *f* muss seinem Pfad folgend die Segmente bs_2, bs_3, bs_4, bs_7, bs_{10}, bs_9, bs_8 reservieren können, damit dieses Lastszenario eine gültige Lösung besitzt. Beginnend auf Segment bs_2 wird *f* im Zeitschritt 6 von Auftrag *a* für eine Zeiteinheit blockiert (siehe Abschnitt „Warten“). Die Verzögerung erweitert die schon vorhanden Abhängigkeiten der Transportaufträge um $A_{nachfolger}(a) = \{f\}$.

Betrachtet man die Abhängigkeitsbeschreibung als Komposition von Funktionen wird die Verkettung aller 6 Transportaufträge $b \rightarrow c \rightarrow d \rightarrow e \rightarrow a \rightarrow f$ sichtbar. Eine Verkettung bezogen auf diesen Algorithmus bedeutet jedoch, dass alle Kettenglieder unterhalb eines Auftrages neu berechnet werden müssen, falls dieser verdrängt werden sollte. Eine zyklische Abhängigkeit entsteht in diesem Beispiel genau dann, wenn Auftrag *f* einen der anderen Aufträge in dieser Kette verdrängt.

Genau dies passiert, wenn *f* seinem Pfad folgend das Segment bs_3 reserviert, welches zum Zeitpunkt 7 frei ist. Im nächsten Schritt verdrängt *f* nun den auf bs_3 stehenden Auftrag *b* und es entsteht die zyklische Abhängigkeit. Dieser Vorgang lässt sich nicht ohne Veränderung von außen vermeiden und gleicht damit im Ergebnis dem Deadlock der ereignisorientierten Simulation.

	bs_1	bs_2	bs_3	bs_4	bs_5	bs_6	bs_7	bs_8	bs_9	bs_{10}	bs_{11}	bs_{12}
1											c	
2										c_1	c_2	
3		a								c	e	b
4		a_2	a_1						b_1	c	e	b_2
5		f	a						b	c	e	d
6		f	a_2	a_1		b_1			b_2	c	e	d
7		f_2	f_1	a		b			c_1	c_2	e	d
8			b_1	a_2		b_2	a_1		c	e_1	e_2	d
9			b			c_1	a		c_2	e		d
10			b_2	b_1		c	a		d_1	e		d_2
11			c_1	b		c_2	a		d	e		
12			c	b_2	b_1	d_1	a		d_2	e		
13			c_2	c_1	b	d	a		e_1	e_2		
14			d_1	c		d_2	a_2		e	a_1		
15			d	c_2	c_1	e_1			e_2	a		
16	d_1		d_2		c	e			a_1	a_2		
17	d		e_1			e_2			a			
18			e					a_1	a_2			
19	e_1		e_2					a				
20	e											

Tabelle 6: Positionstabelle während der 10. Iteration des Algorithmus. $A_{bearbeitet}$ = {b,c,d,e,a,d,f}, $A_{ausstehend}$ = (), $A_{nachfolger}(a)$ = {f}, $A_{nachfolger}(b)$ = {c}, $A_{nachfolger}(c)$ = {e,d}, $A_{nachfolger}(d)$ = {e}, $A_{nachfolger}(e)$ = {a}

Ohne Sicherheitsüberprüfung würde der Algorithmus alle Auftragsberechnungen löschen und mit einer leeren Positionstabelle von vorne beginnen, was wieder in den gleichen Zustand führte. Im Gegensatz zur Simulation ist diese Überprüfung jedoch trivial durchführbar, da die explizite Abbildung der zyklischen Abhängigkeit einfach aus den Datenstrukturen ablesbar ist.

4 Fazit

In der Praxis verspricht dieser Ansatz gegenüber herkömmlichen Simulationswerkzeugen mehrere Vorteile. Ein Analysewerkzeug kann die Parallelisierbarkeit zur effizienten, verteilten Programmausführung auf Mehrkernprozessorsystemen nutzen. Das Datenmodell lässt sich gezielt nach spezifischen Wechselwirkungen hin auswerten, beispielsweise um daraus auf die allgemeine Komplexität des Steuerungsproblems zu schließen.

Die Funktion des nachträglichen Verdrängens von bereits berechneten Aufträgen soll in zukünftigen Arbeiten so erweitert werden, dass die Berechnungen in Echtzeit ausgeführt werden. Dies bedeutet, dass der Algorithmus nicht nur in der Planungsphase (Simulation), sondern auch im Echtzeitbetrieb in der Materialflusssteuerung eingesetzt werden kann (vgl. [Lib08]).

Literatur

[Arn07] Arnold, D.; Furmans, K.: *Materialfluss in Logistiksystemen*, Berlin: Springer, 2007

[Gud05] Gudehus, T.: *Logistik, Grundlagen - Strategien -Anwendungen*, Berlin: Springer, 2005

[Lib08] Libert, S.; Nettsträter, A.; ten Hompel, M.: *Das RTL-Modell für dezentrale Materialflusssteuerungen*. In: Crostack, H.-A.; ten Hompel, M. (Hrsg.): *Belastungsabhängige Auslegung, Überwachung und Steuerung von intralogistischen Systemen - Berichte aus dem SFB 696*, Dortmund : Praxiswissen, 2008, S. 137-161

[Lie09] Liekenbrock, D.: *Untersuchung der Selbstorganisationsfähigkeit von Materialflüssen im Internet der Dinge am Beispiel automatisierter Gepäckförderanlagen*, Dortmund: Lehrstuhl für Förder- und Lagerwesen, TU Dortmund, 2009

[May09] Mayer, S. H.: *Development of a completely decentralized control system for modular continuous conveyors*. Karlsruhe: Institut für Fördertechnik und Logistiksysteme, Universität Karlsruhe (TH), 2009

[Sch08a] Schenk. M.; Tolujew, J.; Reggelin, T.: *Mesoskopische Simulation von Flusssystemen - algorithmisch steuern und analytisch berechnen*. In: Nyhuis, P. (Hrsg.): Beiträge zu einer Theorie der Logistik. Berlin: Springer-Verlag, 2008

[Sch08b] Scholz-Reiter, B.; de Beer, C., Freitag, M., Haman, T., Rekersbrink, H., Tervo, J.T.: *Dynamik logistischer Systeme*. In: Nyhuis, P. (Hrsg.): Beiträge zu einer Theorie der Logistik., Berlin: Springer-Verlag, 2008

[Wus10] Wustmann, D.; Vasyutynskyy, V.; Schmidt, T.; Kabitzsch, K.: *Diagnose und Optimierung von Materialflusssteuerungen*, Schlussbericht zum AiF-Forschungsvorhaben 15770, Dresden: Institut für Technische Logistik und Arbeitssysteme, TU Dresden, 2010