Parameteroptimierung von Materialflusssimulationen durch Partikelschwarmalgorithmen

Christoph Laroque[1], Barthold Urban[1], Markus Eberling[2]

[1]Heinz Nixdorf Institut,
[2]Institut für Informatik,
(beide) Universität Paderborn

1 Einführung

Die Wirtschaftsinformatik bietet insb. im Bereich Operations Reasearch unterschiedliche Methoden, um ein weites Spektrum an Problemstellungen zu lösen. Typische Probleme sind Traveling-Salesman, Scheduling, Standortplanung, etc. Sie werden heute über Modellen abgebildet und mit Algorithmen, u.a. der Simulation und Optimierung, unter Berücksichtigung von Restriktionen auf Optimalität hin gelöst. In der Simulation werden wie in der Optimierung Modelle genutzt, die ein Abbild der Realität darstellen. Sie werden jeweils verwendet, um eine möglichst zieloptimale Belegung der Entscheidungsvariablen zu finden, beispielsweise den maximalen Durchsatz für ein Produktionsnetzwerk oder -system. Bisher werden diese Belegungen im Fall der Optimierung durch mathematische Optimierungsalgorithmen oder (Meta-)heuristiken bestimmt (Suhl und Mellouli 2006). In der Simulation wird diese Belegung durch Evaluierung verschiedener Szenarien und Vergleich der Simulationsergebnisse iterativ erzielt (Suhl und Mellouli 2006). Dieser Vorgang wird zumeist manuell wiederholt, bis ein (evtl. stochastisches) Simulationsmodell ausreichend gute experimentelle Werte liefert.

Aus dem hohen Zeit- oder Berechnungsaufwand dieser Methoden resultierte ein kombiniertes Vorgehen beider Methoden, in dem die Belegung der Entscheidungsvariablen im Simulationsmodell, die Modellkonfiguration, mit Hilfe eines Optimierungsalgorithmus automatisiert bestimmt wird (Law und Kelton 2000). Sinn dieser Methodik ist es, den Entwicklungsprozess zu beschleunigen und trotz Veränderungen am Modell schnell den verbundenen, maximalen Durchsatz zu bestimmen.

Ziel der hier vorgestellten Arbeit war eine Machbarkeitsstudie für die Kombination der Simulation & Optimierung, die unter Verwendung der Gruppe der Partikelschwarmalgorithmen (Kennedy und Eberhart 1995) möglichst schnell eine

gute Modellkonfiguration eines Simulationsmodells ermöglicht. Dies ist vor allem im Hinblick auf Situationen interessant, in denen die konventionelle, mathematische Modellierung und Optimierung nicht anwendbar sind. Konkret wurde die Machbarkeitsstudie am Beispiel der Simulationssoftware d3FACT insight und der Partikelschwarmoptimierung (PSO) durchgeführt.

2 Grundbegriffe

2.1 Diskrete, ereignisgesteuerte Materialflusssimulation

Eine etablierte Methode zur Planung, Absicherung und Verbesserung von Produktionsprozessen ist die Materialflusssimulation als Anwendungsgebiet der ereignisdiskreten Simulation (Law und Kelton 2000). Typische Probleme, die in dieser Domäne untersucht werden, sind Absicherungen von Produktionsprogrammen, Losgrößenplanung, Layoutplanung, Dimensionierung von Puffern, Personaleinsatzplanung, etc. Insbesondere eignet sich Simulation, wenn es schwierig oder sehr kostenaufwendig wäre, das Realsystem direkt zu verändern. Zeitabschnitte lassen sich in verkürzter Zeit betrachten, erlauben Sensitivitätsanalysen des Modells und eine grafische Animation zur Absicherung von Steuerungsregeln. Folglich empfiehlt sich die Anwendung, wenn analytische Methoden versagen, oder es keine andere Möglichkeit gibt, ein System zu untersuchen (Law und Kelton 2000).

Bei d³FACT insight handelt es sich um eine ereignisgesteuerte, diskrete Simulationsumgebung zur Materialflusssimulation, die am Heinz Nixdorf Institut der Universität Paderborn entwickelt wird. Ein wesentliches Merkmal der Umgebung ist ihre Mehrbenutzerfähigkeit, die es ermöglicht, gleichzeitig und kooperativ zu modellieren und simulieren. d³FACT insight setzt sich aus verschiedenen Modulen zusammen. Die Software basiert auf der Eclipse *Rich Client Platform* (RCP) und ist in Java implementiert. Das implementierte Simulationsmodell wird in einem XML-Format abgelegt. Nach und während dem Durchführen eines Simulationslaufs können die Ergebnisse über verschiedene Visualisierungsformen animiert und ausgewertet werden (Dangelmaier und Laroque 2007, S. 253-268).

Die Genauigkeit und Korrektheit des Modells entscheiden über die Anwendbarkeit der Ergebnisse auf die Realität. Gerade im Hinblick auf die simulationsgestützte Optimierung ist der Aufwand, der durch Datensammlung und Analyse entsteht, größer als bei anderen Optimierungsverfahren. Abgesehen von den Schwierigkeiten, systematisch die Lösung von Hand zu verbessern, besteht bei stochastischen Einflüssen im Modell vor allem das Problem der Anzahl benötigter Simulationsläufe. Mit der Evaluierung jeder Konfiguration ist ein mehr oder weniger großer Rechen- und Zeitaufwand verbunden, weil durch die stochastischen Einflüsse für jedes Szenario mehrere Simulationsläufe benötigt werden (Law und Kelton 2000). Optimierung findet hier durch iterative Modellmodifikation statt.

Im Hinblick auf die Zielsetzung stellt sich die zentrale Frage: Kann man die Simulation mit Optimierungsalgorithmen integrieren um eine automatisierte Optimierung des Materialflussnetzwerkes zu erzielen?

2.2 Partikelschwarmoptimierung (PSO)

Die PSO wurde ursprünglich von Dr. Russel C. Eberhart und Dr. James Kennedy formuliert. Es handelt sich um einen evolutionären Algorithmus, der einen populationsbasierten Ansatz verfolgt (Kennedy und Eberhart 1995). Sie unterscheiden sich von den klassischen Metaheuristiken[1], weil hier die Population eine Menge von Lösungen bildet. Die Zielfunktionswerte werden jedem Individuum als Fitnesswert zugeordnet. Die Individuen haben eine Position im Lösungsraum, der aus allen Entscheidungsvariablen gebildet und durch Restriktionen eingeschränkt wird. Bei der PSO bleibt die Anfangspopulation über die gesamte Dauer des Verfahrens erhalten.

Jedes Partikel x_i ist ein Punkt in einem multidimensionalen Raum, seine Koordinaten werden durch einen Positionsvektor (im folgenden Position genannt) $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ repräsentiert. Dabei ist x_{in} die n-te Komponente des i-ten Partikels. Die Belegung dieser Komponente stellt den Wert einer Entscheidungsvariable dar. Jedem Partikel wird durch eine Funktion $f(x_i)$, dem Fitnesswert zugeordnet, der dem Wert der Zielfunktion gleichkommt. Die Partikel bewegen sich durch den Lösungsraum, wobei ihre Geschwindigkeit durch den Geschwindigkeitsvektor $\vec{v}_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ (im Folgenden Geschwindigkeit genannt) beschrieben wird. Jedes Partikel kennt hier die Positionen aller anderen Partikel und somit auch die global beste Position. Sobald die Position des Partikels *i* in einer Iteration *k* einen besseren Fitnesswert liefert als ihr bisher bester gefundener Fitnesswert im Lösungsraum, wird x_i^k die persönlich beste Position p_i.

In jeder Iteration wird auch die global beste Position p_g aus allen p_i ermittelt und durch diese gesetzt. Sobald p_g und p_i bestimmt wurden, wird die Geschwindigkeit v_i des Partikels wie folgt aktualisiert:

$$v_{id}^{k+1} = v_{id}^k c_1 r_1 \left(p_{id} - x_{id}^k\right) + c_2 r_2 \left(p_{gd} - x_{id}^k\right).$$

Der Geschwindigkeitsvektor wird nicht direkt, sondern komponentenweise bestimmt. Hierbei sind r_1 und r_2 gleichverteilte Zufallsvariablen aus dem Intervall [0 ; 1]. c_1 und c_2 werden *Beschleunigungskoeffizienten* genannt und werden in der klassischen Variante auf 2 gesetzt. c_1 steuert hierbei die Schrittweite in Richtung p_i, während c_2 analog die Schrittweite in Richtung p_g steuert. Durch die komponentenweise Bestimmung des Geschwindigkeitsvektors fließen jeweils andere Zufalls-

[1] Vgl. Tabu-Suche oder Simulierte Abkühlung, hier wird zu einem gegebenen Zeitpunkt nur eine Lösung mit der bisherigen Besten verglichen.

variablen r_1 und r_2 in jede Vektorkomponente ein. Zum Schluss werden noch die Positionen aller Partikel mit der neuen Geschwindigkeit nach folgender Gleichung aktualisiert:

$$\vec{x}_i^{k+1} = \vec{x}_i^k + \vec{v}_i^{k+1}$$

Bei der Initialisierung des PSO-Algorithmus werden die Startpositionen der Partikel so gesetzt, dass sie einer gültigen Lösung entsprechen. Positionen innerhalb der festgelegten Dimensionsgrenzen stellen Restriktionen dar. Die Behandlung einfacher Restriktionen wie Wertebereichen stellt sich somit bei PSO recht einfach dar. Restriktionen allgemein erfordern bei der PSO einer gesonderten Behandlung[2]. Ist eine Position ungültig ist die Lösung folglich unbrauchbar.

Aufgrund ihrer Geschwindigkeit werden Partikel an ihrem Ziel „vorbeifliegen", wobei sie weiterhin den Lösungsraum absuchen. Das reduziert die Wahrscheinlichkeit in lokalen Optima stecken zu bleiben und erhöht gleichzeitig die Wahrscheinlichkeit, dass der Lösungsraum im Ganzen durchsucht wird, anstatt nur eines Teilbereiches. Andererseits besteht die Gefahr, dass durch das ständige „Vorbeifliegen" der Algorithmus nicht konvergiert. Dieses Problem und andere Schwachstellen wurden in der Forschung adressiert und es wurden einige Modifikationen der ursprünglichen PSO vorgestellt.

Eine der bekanntesten Modifikationen des PSO-Algorithmus ist die *Trägheit* (Eberhart und Shi 1998), ein skalarer Faktor ω. Mit der Trägheit wird der Explosion der Geschwindigkeit entgegen gewirkt[3]. Formal:

$$v_{id}^{k+1} = \omega v_{id}^k + c_1 r_1 \left(p_{id} - x_{id}^k\right) + c_2 r_2 \left(p_{gd} - x_{id}^k\right)$$

Shi und Eberhart (1998) haben Untersuchungen für ω im Intervall [0 ; 1,4] gemacht. Die Forschungen ergaben, dass mit kleineren Werten im Intervall [0,8 ; 1,2] eine schnellere Konvergenz erreicht wird, wobei größere Werte (> 1,2) Fehler bei der Konvergenz ergeben. Wenn zu kleine Werte genommen werden tendiert der Schwarm dazu in lokalen Optima stecken zu bleiben, was wir ebenfalls beobachten konnten. Die Parameter für unsere Implementierung wurden aufbauend auf den Ergebnissen von Zhang et al. (2005) ausgewählt[4].

Angeline (1999) hat bemerkt, dass bei der ursprünglichen PSO eine schwache implizite Selektion der Partikel, in Bezug auf die persönlich beste Position p_i, stattfindet. Der Zweck der *Selektion* der Partikel ist, die Positionen der Partikel gezielt in Regionen des Lösungsraums zu platzieren, die aus früheren Ergebnissen vielversprechend waren. Dadurch können diese Regionen intensiver untersucht werden. Dieser Prozess soll ausgeführt werden, bevor die Geschwindigkeiten der Partikel

[2] Vgl. Macas et al. 2006 Constraints in Particle Swarm Optimization of Hidden Markow Models.
[3] Vgl. Suresh et al. 2008 Inertia-Adaptive Particle Swarm Optimizer for Improved Global Search.
[4] Vgl. auch Parsopoulos und Vrahatis 2002, S. 240.

aktualisiert werden. Die vorgestellte Modifikation von Angeline hat deutliche Verbesserungen im Bereich lokaler Optima ergeben, schränkt aber wiederum die Fähigkeit globale Optima zu finden ein.

3 Ziel: Partikelschwarmoptimierung in der Materialflusssimulation

Aufbauend auf den im vorherigen Kapitel beschriebenen Modifikationen wird ein Lösungsverfahren beschrieben mit dem die Parametrierung von Materialflusssimulationen ermöglicht wird. Es wird ebenfalls auf die Problematiken im Vergleich zu bisherigen Vorgehensweisen eingegangen.

3.1 Kombination der PSO mit der Materialflusssimulation

PSO-Algorithmen arbeiten mit Partikeln, wobei jedes Partikel eine Lösung – und damit eine Konfiguration des Systems – darstellt. Da jedes Partikel in jeder Iteration einer Evaluierung bedarf, bleibt die Frage offen, ob sich das Problem in vertretbarer Zeit lösen lässt. Einige der im vorherigen Abschnitt vorgestellten Varianten und Modifikationen eignen sich in ihrer Grundidee, um den Rechenaufwand zu reduzieren, in dem nach wenigen Iterationen bereits gute Ergebnisse erzielt werden können. Aufgrund der Konvergenzeigenschaft von Metaheuristiken führen am Anfang wenige Iterationen zu wesentlichen Verbesserungen in der Zielsetzung, während zum Schluss viele Iterationen nur marginale Verbesserungen erzielen.

Für die PSO wurde auf eine vorgefertigte Implementierung in Java von Pablo Cingolani[5] zurückgegriffen. Diese stellt nur eine Implementierung der ursprünglichen und initial beschriebenen PSO dar und wird im weiteren Verlauf als *Standard-PSO* bezeichnet. Da die Anzahl der Iterationen des Algorithmus entscheidend für den Rechenaufwand in Verbindung mit Simulation ist, wurden einige Modifikationen vorgenommen die sich bei Voruntersuchungen als geeignet erwiesen:

1. Intelligente Positionierung - Die Partikel werden gezielt in bestimmte Bereiche des Lösungsraums platziert[6].
2. Unterteilung in drei Phasen - Erkundung, Ausbeutung und Intensivierung. (Im weiteren als *3-Phasen-PSO* bezeichnet)
3. Adaptive Geschwindigkeit - In der ersten Phase (3 Phasen-PSO) wird die Geschwindigkeit der Partikel reduziert, um die Explosion der Partikel im Lösungsraum einzudämmen.
4. Selektion - Die Partikel aus vermeintlich schlechten Regionen des Lösungsraums in gute Regionen platzieren.

[5] http://jswarm-pso.sourceforge.net/ Lizenziert unter der General Public Licence.

[6] Die "Intelligenz" basiert hier darauf eine möglichst große Dispersion am Anfang sicherzustellen und einer ungünstigen Initialisierung des Algorithmus entgegenzuwirken.

5. Mehrfachabtastung - Die Positionen der Partikel werden mehrfach evaluiert und Durchschnitte gebildet, um stochastische Einflüsse zu reduzieren.
6. Schwellenwert - Ein zusätzliches adaptives Abbruchkriterium zu fest vorgegebenen Begrenzungen der Iterationsanzahl.

Der Basisalgorithmus wurde um die beschriebenen Konzepte erweitert. Für die Realisierung einer GUI zur Steuerung und Konfiguration des Algorithmus wurde die *Eclipse Rich Client Plattform* (RCP) eingesetzt.

3.2 Konstrukt der 3-Phasen-PSO

Die Entscheidungsvariablen sind die Parameter des Modells und bilden mit den zulässigen Bereichen einen Lösungsraum, welcher die Ausgangsbasis für den PSO-Algorithmus darstellt. Wegen der simulationsbasierten Fitnessevaluierung unterliegt die Zielfunktion keinen Einschränkungen bzgl. der Form[7]. Es können also beliebige Zielsetzungen formuliert werden, die aus den gewonnen Kennzahlen der Simulationsläufe gebildet werden können. Die Art der Zielfunktion hat hierbei auf die Laufzeit des Algorithmus insgesamt einen vernachlässigbaren Einfluss.

Phase I:

1. Initialisiere Anfangspopulation S der Größe n auf intelligent bestimmten Anfangspositionen.
2. Initialisiere Geschwindigkeitsvektoren als Nullvektoren.
3. Werte Fitness der Partikel aus und aktualisiere Geschwindigkeitsvektoren, wobei hier nur eine schwache Anziehung durch das bisherige global beste Partikel ausgeübt wird.
4. Speichere bisherige Fitnesswerte in der Historie.
5. Bestimme neue Position.
6. Führe Schritte 3 - 5 durch, bis Abbruchkriterium erfüllt.

Phase II:

1. Suche aus der Historie die n besten Fitnesswerte und setze die Population S auf diese Positionen.
2. Initialisiere Geschwindigkeitsvektor, so dass er in Richtung Gruppenbester zeigt.
3. Werte Fitness aus und aktualisiere Historie.
4. Aktualisiere Geschwindigkeitsvektoren.
5. Führe Schritte 5 - 7 durch, solange eine Verbesserung auftritt.

[7] Vgl. Völker (2003) Reduktion von Simulationsmodellen zur simulationsbasierten Optimierung in der Termin- und Kapazitätsplanung S. 37f.

Phase III:

1. Identifiziere beste Gruppe.
2. Lösche die Partikel der anderen Gruppen.
3. Initialisiere Geschwindigkeitsvektoren in Richtung global bestes Partikel.
4. Aktualisiere Position und exportiere XML-Dateien.
5. Evaluiere Fitness und berechne neue Vektoren.
6. Fahre Fort bis Abbruchkriterium erfüllt.

Die in Abschnitt 2.2 vorgestellte Selektion wurde in (Angeline 1999) untersucht mit dem Ergebnis, dass die PSO in lokalen Optima steckenbleibt. Die intelligente Initialisierung mit größtmöglicher Dispersion soll dieses Risiko stark reduzieren. Hierbei werden die Partikel gleichmäßig im zulässigen Lösungsraum verstreut, damit sich der Schwarm einen "Überblick" verschaffen kann. Außerdem kann die ursprüngliche zufällige Initialisierung der Startpositionen dazu führen, dass sich alle Partikel von Anfang an in einem Bereich konzentrieren und es zu keiner Erkundung des restlichen Lösungsraums kommt. In diesem Fall soll Selektion in Verbindung mit einer intelligenten Initialisierung und adaptivem Schwarmverhalten die Anzahl der Fitnessevaluierung stark reduzieren.

3.3 Herausforderungen & Adaptionen

Problematisch ist die Anwendung dieses Lösungsansatzes auf komplexe Systeme mit stochastischen Einflüssen. Diese haben auf Grund ihrer stochastischen Natur immer ein anderes Ergebnis – und somit eine andere Fitness. Ein Problem, was hierbei auftritt, ist die Entstehung von *Rauschen*. Eine Möglichkeit dieses Problem zu adressieren, ist es eine Position mehrfach auszuwerten und einen Durchschnitt zu bilden (*Mehrfachabtastung*), oder die Historie für *sequentielle Abtastung* auszunutzen. Hierbei werden die letzten Werte geglättet um einen Trend zu errechnen – letztere Variante vervielfacht den Rechenaufwand im Gegensatz zur Mehrfachabtastung nicht. Das Rauschen verursacht, dass Partikel nicht das globale Optimum erreichen, da dieses nicht genau identifiziert werden kann (Bartz-Beielstein 2005).

Das vorgestellte, kombinierte Verfahren aus Simulation und Metaheuristik, unterscheidet sich von herkömmlichen Optimierungsansätzen in dem Punkt, dass die Evaluierung der Fitness eines Partikels nicht durch eine mathematische Funktion möglich ist, sondern durch einen Simulationslauf in d³FACT insight ermittelt werden muss. Als Kernproblematik bleibt die Tatsache bestehen, dass die Simulationsläufe die meiste Zeit und Rechenkapazität in Anspruch nehmen. Es gilt ihre Anzahl zu minimieren.

4 Umsetzung und Ergebnisse

Neben der Machbarkeit war als zweiter Untersuchungsgegenstand zu bestimmen, wie sich die vorgeschlagenen Modifikationen der PSO auf dessen Laufzeit und Lösungsqualität auf eine mathematische Testfunktion sowie auf d3FACT insight als Zielfunktion auswirken. Hierbei stand im Vordergrund den Algorithmus so einzustellen, dass eine möglichst geringe Iterationszahl für eine möglichst gute Lösung notwendig ist.

Zur Untersuchung wurde ein Testmodel für d3Fact insight mit neun Komponenten und eine Testfunktion genutzt. Das Testmodel besitzt neun Dimensionen, die sich aus den zu steuernden Parametern ergaben. Dies waren die Maschinen- und Förderbandzeiten, die Frequenz der Quelle und die prozentuale Aufteilung der Ausgänge an der Gabel[8].

Die Testfunktion besitzt zwei Dimensionen und ist wie folgt definiert:

$$f(x,y) = \sin(x) + \sin(y) + 0{,}1x + 0{,}2y$$

Die Funktion hat unendlich viele lokale Minima und Maxima und wurde eigentlich nur zum grundsätzlichen Testen des Algorithmus genutzt. Aufgrund der kurzen Rechenzeit haben wir die Laufzeittests auch mit dieser Funktion durchgeführt.

4.1 Ergebisse der Tests

Tabelle 1 zeigt einen Ausschnitt aus den GAP-Werten der gesamten gesammelten Daten. Auffällig ist, dass der GAP mit aktivierter intelligenter Positionierung in Verbindung mit d³FACT insight ab einer Anzahl von 11 Partikeln auf Durchschnittswerte unter 0,6% sinkt. Aufgrund der neun Dimensionen des Modells kommt die intelligente Positionierung erst ab dieser Populationsgröße stark zum Tragen.

Die in den oben beschriebenen Testläufen beobachtete Tendenz zu einer Vervielfachung des Evaluierungsaufwands wird in Abb. 1 veranschaulicht. Die Anzahl der notwendigen Evaluierungen steigt ca. linear mit der Anzahl der Partikel und ergibt sich als Produkt aus der Iterationszahl und der Populationsgröße. Hier zeigt sich auch die Wirkung der intelligenten Positionierung. Die Partikel, die zufällig eingestreut werden verbessern zwar das Ergebnis noch geringfügig, vervielfachen den Aufwand allerdings erheblich.

[8] Die Aufteilung bestimmt hier im engeren Sinn die Wahrscheinlichkeit für die Wahl eines Ausgangs durch den durchlaufenden Token.

Tabelle 1: Durchschnittliche GAP-Werte der Testläufe

	Fehler bei Testfunktion				Fehler bei d³FACT insight			
#	Standard PSO	Standard PSO (intell.)	3-Phasen-PSO	3-Phasen-PSO (intell.)	Standard PSO	Standard PSO (intell.)	3-Phasen-PSO	3-Phasen-PSO (intell.)
6	4,64%	0,65%	2,39%	0,25%	10,47%	9,00%	8,80%	4,79%
9	2,25%	0,42%	1,44%	0,12%	7,27%	3,51%	5,75%	0,77%
11	1,03%	0,35%	0,42%	0,07%	6,85%	0,60%	3,86%	0,75%
22	0,30%	0,10%	0,10%	0,03%	2,61%	0,26%	1,42%	0,07%
33	0,26%	0,06%	0,01%	0,01%	5,70%	2,82%	4,11%	1,29%
GAP	**1,70%**	**0,32%**	**0,87%**	**0,10%**	**6,58%**	**3,24%**	**4,79%**	**1,54%**

Ein Vergleich der besten Konfigurationen für den jeweiligen Algorithmus hat ergeben, dass die 3-Phasen-PSO verbesserungswürdig bleibt. Im Schnitt arbeitet diese 3,7% langsamer bei d³FACT insight als die Standard-PSO. Allerdings sind die GAPs 1,5-1,6% niedriger. Daher kann die Verbesserung der Lösungsqualität des 3-Phasen-Algorithmus als erste Beobachtung festgehalten werden – relativ werden die GAPs ca. halbiert, was eine 3,7% höhere Laufzeit durchaus vertretbar macht.

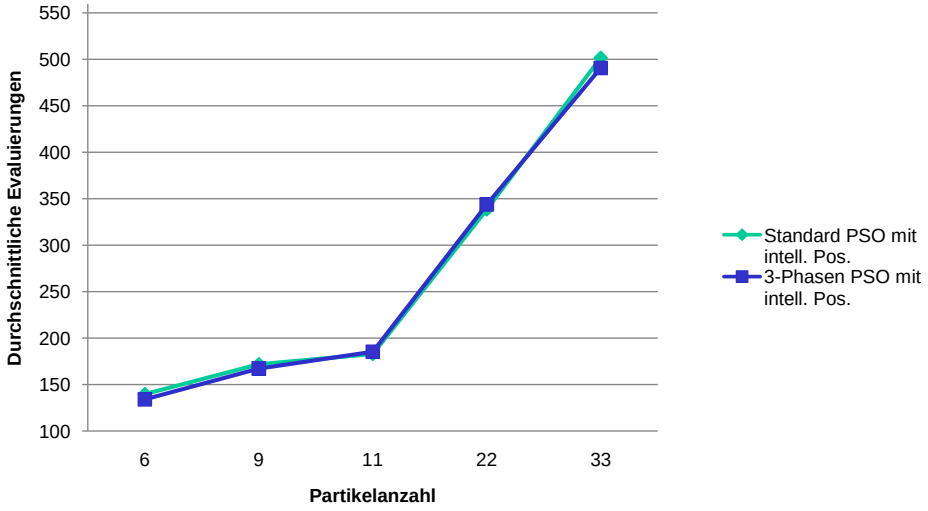

Abbildung 1: Evaluierungen bei 3-Phasen-PSO mit Simulationsmodell

4.2 Schlussfolgerungen

Nicht alle implementierten Modifikationen haben zum gewünschten Erfolg geführt. Die Testläufe zeigen jedoch als erste Schlussfolgerung, dass die Konfiguration des Schwarms nicht beliebig erfolgen kann. Vielmehr ist auf Parameter wie Populationsgröße und die Wahl des Schwellenwertes zu achten. Während die Populationsgröße stark über den Aufwand und Lösungsqualität entscheidet – hier gilt es einen guten Kompromiss zu wählen – erhöht ein höherer Schwellenwert die Lösung im Schnitt unwesentlich, den Aufwand jedoch beträchtlich. Die untersuchten Schwellenwerte waren 10 und 20 Iterationen. Die Differenz der über alle Standardtestläufe ermittelten GAPs für diese beiden Werte beträgt lediglich 0,17%, während sich die Laufzeit im Schnitt fast verdoppelt.

Die Lösungsqualität lässt sich durch Aktivierung der intelligenten Positionierung signifikant verbessern. Die Untersuchungen zeigen jedoch, dass sich je nach Partikel- und Dimensionsanzahl, Schwächen ergeben. Hier spielt insbesondere eine Rolle, wie die letzten beiden (intelligenten) Partikel gesetzt werden. Als mögliche Verbesserung sollten diese beiden Positionen immer besetzt werden.

Die Untersuchungen haben ferner gezeigt, dass die im 3-Phasen-Algorithmus umgesetzte Selektion in Verbindung mit einer Historie für evaluierte Positionen zu einer Verbesserung bzgl. des GAP führt. Jedoch bleiben die Fragen, ob die Selektion mehrfach und wann diese durchgeführt werden sollte, offen. Insgesamt belegen die Untersuchungen, dass das 3-Phasen-Konstrukt eine gute Basis für weitere Entwicklungen darstellt.

5 Fazit und Ausblick

Mit der implementierten Kombination aus PSO und d³FACT insight konnte erfolgreich gezeigt werden, dass sich Simulationsmodelle durch PSO parametrieren lassen. Das Testmodell weicht in seiner Komplexität noch weit von den in der Realität zu betrachtenden Modellen ab. Mit dem geschaffenen Prototyp ist die prinzipielle Machbarkeit gezeigt; es bleibt allerdings zu prüfen, ob sich komplexere Modelle in vertretbarer Zeit parametrieren lassen.

Erste Erkenntnisse über das Verhalten und die Laufzeit der 3-Phasen-PSO wurden gesammelt und zeigen, dass dieser Entwurf weiter bzgl. der Laufzeit verbessert werden kann. Potenziale dafür liegen in einer intensiveren Nutzung der Selektion, bzw. in einer besseren Anpassung der Schwarmparameter in den einzelnen Phasen. Die Qualität der Lösungen konnte bereits durch die hier beschriebenen Modifikationen signifikant verbessert werden.

Literatur

Angeline PJ (1999) Using Selection to Improve Particle Swarm Optimization. In: Proceedings of International Joint Conference on Neural Networks 1999. IEEE CS Press.

Bartz-Beielstein T, Blum DD, Branke J (2005) Particle swarm optimization and sequential sampling in noisy environments. In: Metaheuristics International Conference. University of Vienna.

Dangelmaier W, Laroque C (2007) Immersive 3D-Ablaufsimulation von richtungsoffenen Materialflussmodellen zur integrierten Planung und Absicherung von Fertigungssystemen. In: Leobener Logistik Cases - Management komplexer Materialflüsse mittels Simulation. DUV Verlage.

Kennedy J, Eberhart RC (1995) Swarm Intelligence. MK-Publishers.

Kaushik S, Ghosh S, Kundu D, Sen A, Das S, Abraham A (2008) Inertia-Adaptive Particle Swarm Optimizer for Improved Global Search. IEEE CS Press.

Law AM, Kelton WD (2000) Simulation Modeling and Analysis. McGraw-Hill.

Macas M, Novalak D, Lhotska L (2006) Constraints in Particle Swarm Optimization of Hidden Markov Models. In: Lecture Notes In Computer Science Vol. 4224. Springer, Heidelberg.

Parsopoulos KE, Vrahatis MN (2002) Recent approaches to global optimization problems through particle swarm optimization. In: Natural Computing Vol.1. Springer, Heidelberg.

Shi Y, Eberhart RC (1998) A Modified Particle Swarm Optimizer. IEEE CS Press.

Suhl L, Mellouli T (2006) Optimierungssysteme. Springer, Heidelberg.

Völker S (2003) Reduktion von Simulationsmodellen zur simulationsbasierten Optimierung in der Termin- und Kapazitätsplanung. Lang, Frankfurt.

Zhang L, Yu H, Hu S (2005) Optimal choice of parameters for particle swarm optimization. Journal der Zhejiang Universität. SCIENCE 6A(6):528ff.