Schnelles Rescheduling von Gridjobs mit Heuristiken und dem Evolutionären Algorithmus GLEAM

Wilfried Jakob, Alexander Quinte, Karl-Uwe Stucky, Wolfgang Süß

Forschungszentrum Karlsruhe GmbH
Institut für Angewandte Informatik (IAI)
Postfach 3640, 76021 Karlsruhe
Tel. (07247) 82-4663
Fax (07247) 82-2602
E-Mail: {wilfried.jakob, alexander.quinte, uwe.stucky, wolfgang.suess}@iai.fzk.de

Zusammenfassung

Auf Grund der dynamischen Struktur eines Grids bestehend aus heterogenen Rechner-Ressourcen ist eine permanente Umplanung innerhalb weniger Minuten notwendig. Anlass dafür können beispielsweise neue Jobs, Ressourcenausfall oder neue Ressourcen sein. Dies kommt im Grid im Gegensatz zu vielen anderen Produktionsplanungsaufgaben häufiger vor, was die Anforderungen an die multikriterielle Optimierung unter den gegebenen Zeitbeschränkungen noch verschärft. Die hier vorgestellten Ergebnisse sind eine Fortführung von Arbeiten zur Neuplanung in einem quasi leeren Grid, mit denen wir unseren Global Optimising Resource Broker and Allocator GORBA erprobt und verbessert haben und über den in diesem Rahmen bereits berichtet wurde.
Nach einer formalen Definition und Klassifikation des Problems werden neue Heuristiken zur Umplanung vorgestellt, die die Informationen des alten Plans ausnutzen und es wird untersucht, inwieweit sie zur Umplanung beitragen. Außerdem wird über Experimente zur maximalen Last (Jobs und Rechnerressourcen), die in drei Minuten bearbeitbar ist, berichtet.

1 Einleitung

Der Begriff „computational grid“ oder „Grid“ bezeichnet ein heterogenes Rechnernetz, das als virtualisiertes und verteiltes Rechenzentrum verstanden werden kann [1]. Benutzer beschreiben ihre *Anwendungsjobs* durch so genannte *Workflows*, die aus einzelnen elementaren *Gridjobs* bestehen. Ein Workflow kann durch einen gerichteten azyklischen Graphen dargestellt werden, der die Reihenfolgebeziehungen zwischen den einzelnen Gridjobs beschreibt. Außerdem legt der Benutzer fest, welche Ressourcen, wie Anwendungsprogramme, Datenspeicher oder Rechnerleistung benötigt werden, um die einzelnen Gridjobs auszuführen. Dabei können Ressourcen andere Ressourcen benötigen. Z.B. erfordert ein Softwarewerkzeug ein bestimmtes Betriebssystem und eine Hardware, um darauf ausgeführt werden zu können. Daraus folgt das Konzept der Koallokation von Ressourcen. Weitere Angaben der Benutzer betreffen das zur Verfügung stehende Kostenbudget, eine späteste Fertigstellungszeit und eine Präferenz zwischen schneller und kostengünstiger Ausführung. Dazu kommt die erwartete Ausführungszeit der einzelnen Gridjobs, was bei neuen Anwendungen auf Schätzungen beruhen kann und bei älteren auf Erfahrungswerten. Die Betreiber von Ressourcen bieten diese zu unterschiedlichen Tages- oder Wochenzeiten an, wobei auch tageszeit- oder wochentagabhängige Kostenmodelle vorgegeben werden können. Weiterhin unterscheiden sich heterogene Ressourcen meist in ihrer Performance und weisen unterschiedliche Preis-

Leistungs-Verhältnisse auf. Schließlich können manche Ressourcen auf Grund vorhandener Lizenzen im Vergleich zur angebotenen Hardware nur eingeschränkt zur Verfügung stehen.

Ressourcennutzer und –anbieter haben unterschiedliche Interessen: So sind die Anbieter an einer hohen *Auslastung* und damit an kurzer *Gesamtbearbeitungszeit* interessiert. Die Nutzer wollen hingegen, dass die vorgegebenen *Kostenbudgets* und *Zeitlimits* eingehalten oder, noch besser, unterschritten werden. Ein Teil dieser Kriterien steht offensichtlich miteinander in Konflikt, wie z.B. der Wunsch nach schneller *und* billiger Ausführung.

Da davon ausgegangen werden kann, dass Gridjobs Bearbeitungszeiten benötigen, die eher im Stunden- als im Minutenbereich liegen, steht eine bestimmte, wenn auch begrenzte Zeit für Planung und Optimierung zur Verfügung. Im Rahmen dieser Untersuchung wurde ein Zeitraum von drei Minuten als vertretbar angesehen. Wenn ein Umplanungsereignis eintritt, werden alle bereits begonnen Gridjobs und alle, die innerhalb dieses Zeitraums begonnen werden sollen, fixiert und sind damit nicht Gegenstand der Umplanung. Somit wird die Abarbeitung der Gridjobs durch den wiederholten Prozess des Reschedulings nicht gestört.

Während die hier vorgestellte Rescheduling-Aufgabe auf Grund ihrer Besonderheiten vergleichsweise neu ist, ist die Behandlung von Schedulingaufgaben dagegen eine klassische Disziplin des Operations Research. Einen guten Überblick und eine Klassifikation mit entsprechenden Lösungsansätzen liefert z. B. Bruckner [2, 3].

Der zweite Abschnitt enthält eine Zusammenfassung der in [4] ausführlich behandelten formalen Definition des Problems zusammen mit der ebenfalls in [4] detailliert beschriebenen multikriteriellen Bewertungsfunktion. Anschließend erfolgt eine Klassifizierung der Aufgabe und es wird eine Übersicht über vergleichbare Arbeiten gegeben. Der dritte Abschnitt ist der Beschreibung der verwendeten Algorithmen und darunter insbesondere der neuen Umplanungsheuristiken gewidmet. Der Frage nach dem Effekt der neuen Heuristiken und nach Obergrenzen für die in drei Minuten bearbeitbarer Gridjobs und Ressourcenmengen wird in Abschnitt 4 nachgegangen.

2 Definition des Optimierungsproblems und Einordnung

Gegeben seien entsprechend der in [2] beschriebenen Notation eine Ressourcen-Menge $M=\{M_1, \ldots, M_m\}$, eine Menge $J=\{J_1, \ldots, J_l\}$ von Anwendungsjobs und eine Menge O von Gridjobs. Die n Gridjobs eines Anwendungsjobs J_i werden mit $O_{i1}, \ldots, O_{in}$ bezeichnet. Folgende Funktionen seien gegeben:

- Eine Vorgängerfunktion $p:O\times O \rightarrow \{TRUE, FALSE\}$ für die Gridjobs.
- Eine Zuordnungsfunktion $\mu : O \rightarrow \mathcal{P}(\mathcal{P}(M))$ von Gridjobs zu Ressourcenmengen, wobei $\mathcal{P}(M)$ die Potenzmenge von M bezeichnet. Mit μ_{ij} wird die Menge aller möglichen Kombinationen von Ressourcen aus M bezeichnet, die gemeinsam zur Bearbeitung von O_{ij} fähig sind.
- Eine Zeitbedarfsfunktion $t:O\times\mathcal{P}(M) \rightarrow \Re$, die für jeden Gridjob O_{ij} den Zeitbedarf zur Ausführung auf einer Ressourcenmenge $R_{ij}\in\mu_{ij}$ angibt.
- Eine Kostenfunktion $c:\Re\times\mathcal{P}(M) \rightarrow \Re$, die zu jedem Zeitpunkt $z\in\Re$ die Kosten pro Zeiteinheit einer Ressourcenmenge angibt.

Die Optimierung erfolgt durch eine geeignete Wahl der Startzeitpunkte $s(O_{ij}) \in \Re$ und der Ressourcenzuordnungen $R_{ij} \in \mu_{ij}$. Eine gültige Lösung muss folgende zwei Kriterien erfüllen:

1. Alle Gridjobs wurden geplant und die Ressourcen wurden exklusiv belegt:

$$\forall O_{ij} : \exists s(O_{ij}) \in \Re, R_{ij} \in \mu_{ij} : \forall M_j \in R_{ij} : \\ M_j \text{ ist in } [\, s(O_{ij}); s(O_{ij}) + t(O_{ij}, R_{ij}) \,] \text{ exklusiv belegt durch } O_{ij}. \quad (1)$$

2. Vorgängerbeziehungen wurden eingehalten:

$$\forall i, j \neq k : p(O_{ij}, O_{ik}) \Rightarrow s(O_{ik}) \geq s(O_{ij}) + t(O_{ij}, R_{ij}) \quad (2)$$

Die Bewertung einer Verletzung der beiden nachfolgenden Beschränkungen erfolgt durch Straffunktionen, welche sowohl den Umfang der Kosten- oder Zeitüberschreitungen bewerten als auch die Anzahl der davon betroffenen Anwendungsjobs. Sie ergeben einen Faktor zwischen Null und Eins mit dem die „unbestrafte Fitness“ multipliziert wird.

1. Fristen d_i werden eingehalten:

$$\forall J_i : d_i \geq s(O_{in}) + t(O_{in}, R_{in}) \quad \text{wobei } O_{in} \text{ der letzte Gridjob von } J_i \text{ ist.} \quad (3)$$

2. Kostenlimits c_i werden eingehalten:

$$\forall J_i : c_i \geq \sum_{j=1}^{n} \int_{s(O_{ij})}^{s(O_{ij})+t(O_{ij},R_{ij})} c(z, R_{ij})\, dz \quad (4)$$

Die in [4] ausführlicher beschriebene Fitnessfunktion basiert auf der gewichteten Summe der zuvor genannten vier Kriterien plus einem Hilfskriterium. Es unterstützt die Verkürzung der Bearbeitungszeit und damit die Einhaltung des *Zeitlimits* der Anwendungsjobs, indem es die frühere Fertigstellung nicht-terminaler Gridjobs belohnt. Ein Gridjob ist nicht-terminal, wenn er keine Nachfolger hat. Damit wird ein früherer Beginn der terminalen Gridjobs und in der Folge eine Verkürzung der Gesamtbearbeitungszeit ermöglicht. Zur Bewertung werden die Verzögerungen aller nicht-terminalen Gridjobs gegenüber ihrer frühest möglichen Startzeit ermittelt und der Durchschnitt gebildet.

Die Berechnung der Fitnesswerte der einzelnen Kriterien basiert auf Unter- und Obergrenzen für Kosten und Zeiten, die in der ersten Planungsphase durch eine Analyse des Workflows ermittelt werden. Mit Ausnahme der *Auslastung* ergibt sich der Fitnesswert f_i eines Kriteriums i aus dem aktuellen Wert $Kriterium_{i,akt}$ durch

$$f_i = \frac{Kriterium_{i,akt} - Kriterium_{i,\min}}{Kriterium_{i,\max} - Kriterium_{i,\min}} \quad (5)$$

Dadurch werden die Werte f_i unabhängig von der aktuellen Aufgabe. Die sich ergebenden prozent-ähnlichen Werte werden gewichtet und zur Rohfitness aufsummiert, welche bei Budgetverletzungen mit den entsprechenden Straffunktionen multipliziert wird oder andernfalls unverändert bleibt und die Endfitness darstellt. Um den bei der gewichteten Summe möglichen Kompensationseffekten entgegenzuwirken werden die Kriterien einzeln oder in Gruppen gemäß einer Priorität sortiert und erhalten einen so genannten Erfüllungswert. Die Kriterien der höchsten Priorität tragen immer zur Summenbildung bei, während die anderen erst zugeschaltet werden, wenn die Kriterien der nächst höheren Ebene jeweils mindestens den Erfüllungswert erreicht haben. Die Gewichtung beruht auf Erfahrung und ist derzeit so eingestellt, dass ein Ausgleich zwischen den Interessen der Gridakteure angestrebt wird.

Die geschilderte Aufgabenstellung enthält das NP-vollständige Job-Shop-Scheduling Problem als Spezialfall. Es gibt folgende Erweiterung gegenüber dieser klassischen Aufgabe des Operations Research:

- Ressourcenalternativen statt einer Ressource pro Gridjob
- Heterogene Ressourcenalternativen
- Zeitlich beschränkte statt unbeschränkter Verfügbarkeit von Ressourcen
- Gridjobs können mehrere statt nur einen Nachfolger oder Vorgänger haben (Parallele Bearbeitungsstränge in den Workflows)
- Koallokation von Ressourcen möglich statt nicht zugelassen
- Mehrere statt nur ein Bewertungskriterium, insbesondere Kosten

Allein schon auf Grund der NP-Vollständigkeit kann generell nur mit Näherungslösungen gerechnet werden.

Ein vergleichbares Problem konnte in der Literatur nicht gefunden werden, siehe auch [2, 3]. In [5] wird z.B. festgestellt, dass es nur wenig Arbeiten zu multikriteriellem Scheduling gibt, die sich meistens auch nur mit Ein-Maschinen-Problemen befassen. Im Grid-Kontext sind in jüngster Zeit weitere Arbeiten zu multikriterieller Bewertung erschienen. In [6] wird berichtet, dass sich die meisten mit nur zwei Kriterien beschäftigen, wovon das eine optimiert wird und das andere lediglich als Beschränkung dient, siehe z.B. [7, 8]. Der in [8] gewählte Ansatz matrix-förmiger Chromosomen dürfte mit dafür verantwortlich sein, dass nur eine geringe Anzahl von Gridjobs (ca. 30) innerhalb einer Stunde geplant werden konnte. Kurowski et al. [9] verwenden eine modifizierte Form der gewichteten Summe zur multikriteriellen Bewertung, planen aber keine Workflows. Insgesamt konnte keine Arbeit gefunden werden, die in wenigen Minuten eine Menge an Ressourcen und durch Workflows organisierte Gridjobs einer global optimierenden multikriteriellen Planung unterzieht, welche mit dem hier behandelten Umfang vergleichbar ist. Natürlich gibt es eine Vielzahl von Veröffentlichungen und Verfahren zu Teilaspekten des Problems. Zum Beispiel den bekannten Giffler-Thompson-Algorithmus [10, 11], der für die vorliegende Aufgabenstellung erweitert wurde, aber schlechtere Ergebnisse lieferte als die nachfolgend beschriebenen Heuristiken [12].

3 Algorithmen von GORBA und Ergebnisse früherer Arbeiten

GORBA [4, 12, 13] plant in zwei Schritten: Im ersten Schritt werden die Planungsdaten auf Plausibilität geprüft und einer statischen Analyse unterzogen, die erste Ergebnisse für die Unter- und Obergrenzen der einzelnen Kriterien liefert. Danach werden erste Schedules mit Hilfe von Heuristiken erzeugt, die die Unter- und Obergrenzen verfeinern und im Falle ausbleibender Belegungskonflikte bereits das Endergebnis darstellen. Andernfalls dienen sie zur Initialisierung der Startpopulation des im zweiten Schritt folgenden Laufs des Evolutionären Algorithmus (EA) GLEAM (General Learning Evolutionary Algorithm and Method) [14, 15, 16].

Zunächst werden die alten Heuristiken zur Erzeugung von Schedules in einem leeren Grid [4, 12, 13] beschrieben, da die neuen auf diesen aufbauen. Als erster Schritt wird eine Gridjobsequenz gemäß einer der folgenden drei heuristischen Regeln unter Beachtung der Vorgängerbeziehungen gebildet:

1. Gridjobs des Anwendungsjobs mit dem frühesten Fertigstellungstermin zuerst
2. Gridjobs des Anwendungsjobs mit der kürzesten Bearbeitungszeit zuerst
3. Gridjobs mit der kürzesten Bearbeitungszeit zuerst.

Danach werden die drei folgenden Ressourcenallokationsstrategien (RAS) angewandt, so dass insgesamt neun Schedules entstehen:

RAS-1: billigste Ressource, die zum frühest möglichen Zeitpunkt frei ist, wählen

RAS-2: schnellste Ressource, die zum frühest möglichen Zeitpunkt frei ist, wählen

RAS-3: Verwendung von RAS-1 oder RAS-2 für alle Gridjobs eines Anwendungsjobs gemäß seiner Präferenz für kostengünstige oder schnelle Ausführung

Die ersten beiden RAS gelten jeweils für alle zu planenden Anwendungsjobs, so dass deren Gridjobs die gleiche Ressourcenpräferenz anwenden, während die dritte diese Zuordnung differenzierter auf der Ebene der Anwendungsjobs vornimmt.

Die neuen Heuristiken für das Rescheduling verwenden die Gridjobsequenz des alten Plans für alle noch zu planenden Gridjobs. Das sind alle, die zum Zeitpunkt des Umplanungsereignisses noch nicht gestartet wurden und auch nicht innerhalb der Umplanungszeit von 3 Minuten gestartet werden. Gridjobs neuer Anwendungsjobs werden an die Sequenz der alten entsprechend den drei heuristischen Regeln zur Sequenzbildung angehängt. Auf die so entstandenen drei Sequenzen werden jeweils die drei RAS angewandt, woraus wieder neue Schedules resultieren, diesmal aber basierend auf den Vorgaben des alten Plans.

Zusammen mit den alten Heuristiken entstehen so insgesamt 18 Schedules, die bei der Initialisierung der Startpopulation des nachfolgenden GLEAM-Laufs Verwendung finden. Die Standardversion von GLEAM [15, 16] enthält bereits eine Reihe genetischer Operatoren, die für kombinatorische Aufgabenstellungen gut geeignet sind. Aus Platzgründen werden sie hier nur kurz beschrieben und der interessierte Leser wird auf [4, 13, 16] verwiesen. Neben der üblichen Mutation zur Veränderung der Genreihenfolge gibt es Mutationen, die ganze Gensequenzen (so genannte Segmente) verschieben oder ihre interne Reihenfolge umkehren. Die Segmente stellen eine der evolutionären Veränderung unterworfene Metastruktur der Chromosomen dar. 1- und n-Punkt Crossover setzen an den Segmentgrenzen an. Beiden Operatoren ist eine genotypische Reparatur nachgeschaltet, die dafür sorgt, dass bei den resultierenden Nachkommen keine Gene fehlen. Dazu kommt das *order-based crossover* [17], das die relative Genreihenfolge der Eltern bewahrt und hier an die Segmentierung angepasst wurde [12 ,4].

In einer ersten Entwicklungsphase von GORBA wurden die Algorithmen und Heuristiken basierend auf Neuplanungen getestet und bei zwei unterschiedlichen Codierungen und Reparaturarten evaluiert [13, 4]. Dazu wurden vier unterschiedliche Benchmark-Szenarien verwendet, die die Kombinationen von kleinen und großen Ressourcenalternativen und geringen und großen Gridjobabhängigkeiten wiedergeben. Sie werden mit *sR* und *lR* für *small/large Resource alternatives* und *sD* und *lD* für *small/large grid job Dependencies* bezeichnet. Diese wurden mit vier unterschiedlichen Lastmengen getestet, so dass alle Varianten an Hand von16 Benchmarks verglichen wurden. Die erste Codierung bildet die Ressourcenzuordnung auf die Parameter der Gene ab und die Gridjobsequenz auf die Genreihenfolge im Chromosom. Damit sind zwar alle Kombinationen erreichbar, aber der zugehörige Suchraum ist entsprechend groß. Bei der zweiten Codierung entfallen die Ressourcenparameter und die Ressourcenauswahl erfolgt durch die RAS, die durch ein spezielles Gen ausgewählt wird. Die zweite Codierung hat sich innerhalb der kurzen Bearbeitungszeit ab einer Last von 100 Gridjobs als günstiger erwiesen [13, 4]. Die phänotypische Reparatur von Reihenfolgeverletzungen hat bessere Resultate geliefert als eine genotypische. Die Details und die Erklärung dafür sind in [13, 4] zu finden. Als Ergebnis der Voruntersuchungen verwenden wir für die hier be-

handelten Fragestellungen die zweite Codierung, phänotypische Reparatur, das bereits erwähnte Hilfskriterium und das *order-based crossover* [13, 12, 4].

Da die zweite Codierung Grundlage der vorliegenden Untersuchung ist, soll sie hier näher vorgestellt werden. Jedem Gridjob entspricht ein Gen. Die Belegungsmatrix, die die zeitliche Belegung der Ressourcen durch Gridjobs abbildet und die man sich als Gantt-Chart vorstellen kann [13], wird in der Reihenfolge der Gridjob-Gene im Chromosom aufgebaut. Ein besonderes Gen legt dabei die zu verwendende RAS fest und bringt dadurch ein Element der Koevolution mit ein. Der Scheduling-Vorgang pro Gen sieht folgendermaßen aus:

1. Die früheste Startzeit des Gridjobs ist entweder die früheste Startzeit seines Anwendungsjobs oder die späteste Fertigstellungszeit seiner Vorgänger, sofern vorhanden.
2. Gemäß der durch das RAS-Gen festgelegten RAS wird eine Liste alternativer Ressourcen für jede primäre Ressource des Gridjobs angelegt.
3. Beginnend mit den ersten Ressourcen dieser Listen wird die Bearbeitungszeit berechnet und ein freies Zeitfenster bei allen primären und abhängigen Ressourcen gesucht, wobei mit der frühesten Startzeit von Schritt 1 begonnen wird.
4. Die gefundenen Ressourcen werden für den Gridjob und die berechnete Zeit reserviert, d.h., er wird entsprechend in die Belegungsmatrix eingetragen.

4 Experimentelle Ergebnisse für schnelles Rescheduling

Es gibt eine ganze Reihe von Gründen für ein Rescheduling, wobei das Eintreffen eines oder mehrerer neuer Anwendungsjobs das häufigste sein dürfte. Weitere derartige Ereignisse sind Jobabbruch, neu hinzukommende Ressourcen, Ressourcenausfall, deutliche Abweichung von Jobausführungszeiten von der Planzeit, Preisänderungen bei Ressourcen, usw. Die Experimente basieren auf dem am häufigsten erwarteten Umplanungsereignis neuer Jobs und sollen die folgenden drei Fragen beantworten:

1. Profitiert das Rescheduling vom alten Schedule? Wenn ja, bis zu welchem Anteil beendeter und neuer Gridjobs?
2. Wie effektiv sind die alten und neuen Heuristiken und der nachfolgende EA-Lauf?
3. Bis zu welcher Menge an Gridjobs und Ressourcen verbessert GLEAM den besten heuristisch erzeugten Schedule?

Da sich die beiden Benchmark-Szenarien basierend auf einem großen Abhängigkeitsgrad der Gridjobs als die schwierigeren erwiesen haben [4, 12], werden sie für die Experimente zu Grunde gelegt. Sie werden mit *sRlD* und *lRlD* (*s*mall oder *l*arge *R*esource alternatives / *l*arge *D*ependencies) bezeichnet. Die Zeit- und Kostenbudgets wurden bewusst so gewählt, dass die Heuristiken keine Schedules erzeugen konnten, bei denen die Budgets vollständig eingehalten wurden. Ein Kriterium für den Erfolg des Einsatzes von GLEAM war daher das Auffinden von Schedules ohne Budgetverletzungen, was auch in fast allen Benchmarkszenarien erreicht wurde [13, 12, 4].

Die Rescheduling-Experimente zu den ersten beiden Fragestellungen basieren auf diesen Benchmarks, so dass zwei Bewertungskriterien verglichen werden können: Budgeteinhaltung und Fitness-Wert der Schedules. Der einzige EA-Parameter, der bei den Versuchen variiert wurde, ist die Populationsgröße, die für die ersten beiden Fragestellungen zwischen 90 und 900 lag. Bei der Ermittlung der maximal verarbeitbaren Last musste, wie nachfolgend dargestellt, auch mit kleineren Größen gearbeitet werden. Für

jede Benchmark-Situation und Populationsgröße wurden 50 Läufe durchgeführt und die Ergebnisse gemittelt. Zur Absicherung der statistischen Relevanz unterschiedlicher Mittelwerte wurden Konfidenzintervalle und t-Tests bei 99% Sicherheit verwendet.

Zur Beantwortung der ersten beiden Fragen wurde die Abarbeitung der Planungen für die beiden Benchmarkszenarien bei 100 und 200 Gridjobs sowie 10 Ressourcen nach Abarbeitung von 10 oder 20% der Gridjobs gestoppt. Dann kamen jeweils 10, 20, 30, oder 50% neue Gridjobs hinzu, wobei sich der Prozentsatz auf die ursprüngliche Größe bezieht. Daraus ergeben sich 32 Benchmarks für die Umplanung. Die Konzentration auf relativ wenig abgearbeitete Gridjobs wurde gewählt, da diese Situation in der Praxis wahrscheinlicher ist. Außerdem würden mehr abgearbeitete Gridjobs der bereits bearbeiteten Situation „Neuplanung" näher kommen.

Bild 1 vergleicht die Ergebnisse aller 32 Benchmarks. Zur Interpretation der dargestellten normalisierten Fitness muss darauf hingewiesen werden, dass bestimmte geringe Schwankungen wegen Abweichungen bei den Benchmarks unerheblich sind. Werte zwischen 50 und 70% können als gute Ergebnisse interpretiert werden, da die oberen und unteren Grenzen theoretischer Natur sind und in der Regel nicht erreicht werden können. Ergebnisse nahe bei 100% würden entweder auf einen trivialen Fall oder einen Softwarefehler hindeuten. Das wichtigste Ergebnis ist, dass für 10% neue Gridjobs alle acht Szenarien gute Resultate liefern. Der Beitrag der beiden Heuristikgruppen ist deutlich situationsabhängig und in Fällen schwacher Ergebnisse kann GLEAM diese kompensieren. Mit anderen Worten, wenn die Heuristiken gut funktionieren, bleibt wenig Raum für weitere Verbesserungen innerhalb der geringen Planungszeit.

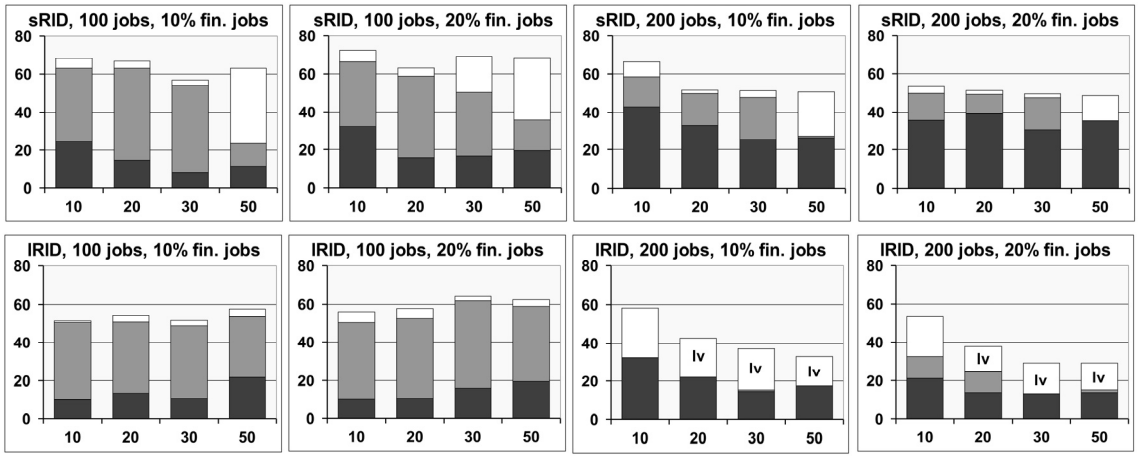

Bild 1: Vergleich der Fitnessanteile, die von den alten Heuristiken (dunkelgrau), den neuen Umplanungsheuristiken (hellgrau) und von GLEAM (weiß) stammen. X-Achse: Anteil neuer Gridjobs in Prozent relativ zur ursprünglichen Anzahl, Y-Achse: normalisierte Endfitness.
Alle GLEAM-Läufe verbessern die Fitness signifikant. Selbst im Falle der geringsten Verbesserung von Benchmark lRlD bei 100 Gridjobs und 10% fertigen und neuen Gridjobs ist das beste heuristische Resultat eindeutig außerhalb des Konfidenzintervalls des GLEAM-Ergebnisses (Der Abstand ist dreimal größer als notwendig).
Abkürzungen soweit nicht im Text erklärt: lv: Budgetverletzungen (limit violations), wobei meistens nur noch ein bis drei Anwendungsjobs Endtermine in geringem Maße überschreiten.

Ein weiteres erfreuliches Ergebnis ist, dass diese Kompensationen auch in gewissem Umfang bei mehr neuen Gridjobs als 10% stattfinden, auch wenn die Budgetverletzungen nicht immer vollständig beseitigt werden können (siehe die sechs mit *lv* gekenn-

zeichneten Säulen). Es kann erwartet werden, dass mehr neue Gridjobs den Beitrag der Umplanungsheuristiken schmälern und Bild 1 bestätigt dies auch tatsächlich für 50% neue Gridjobs. Der Fall *lRlD* bei 200 Gridjobs und 10% abgearbeiteten ist insofern eine Ausnahme, als die Umplanungsheuristiken keinen oder nur einen geringen Beitrag leisten können.

Tabelle 1 fasst den Anteil aller Heuristiken am Gesamtergebnis der heuristischen Planung zusammen. Für alle 32 Benchmarks wird die Fitness jeder Heuristik als Prozentwert der besten Heuristik dieser Parametrierung berechnet. Aus den vier Werten für die beiden Gridjobmengen und *sRlD* sowie *lRlD* wird der Durchschnitt gebildet und in die Tabelle eingetragen. Die rechte Spalte enthält dann den Durchschnitt der Werte für die verschiedenen Anteile beendeter und neuer Gridjobs. Die alten Heuristiken basierend auf kurzen Arbeitszeiten in der unteren Hälfte der Tabelle zeigen das gleiche schlechte Verhalten wie bei der Neuplanung eines leeren Grids [12]. Setzt man sie dagegen zum Rescheduling ein, liefern sie durchaus einige gute Beiträge. Gemäß der Tabelle könnte vermutet werden, dass RAS-3 immer am besten abschneidet. Das hier aus Platzgründen nicht gezeigte Rohmaterial enthält aber dreizehn Fälle, in denen die beiden anderen RAS die besseren Ergebnisse liefern. Daher ist es sinnvoll, sie alle beizubehalten, zumal die Berechnung eines Schedules sehr schnell in Millisekunden erfolgt. Es könnte sich aber als sinnvoll erweisen, die sechs alten Heuristiken basierend auf kurzen Arbeitszeiten wegzulassen.

Tabelle 1: Vergleich der Beiträge aller Heuristiken für unterschiedliche Anteile fertiger und neuer Gridjobs. Die besten Werte jeder Spalte sind dunkelgrau hinterlegt und Werte, die mindestens 90% des besten Wertes erreichen, hellgrau.
Abkürzungen: RH: Rescheduling-Heuristik, KBZ: kürzeste Bearbeitungszeit, RAS: Ressourcenallokationsstrategie, siehe Abschnitt 3.

Abgearbeitete Gridjobs:	10%				20%				Durch-
Neue Gridjobs:	10%	20%	30%	50%	10%	20%	30%	50%	schnitt
RH früheste Fertigstellung & RAS-3	0.90	0.96	0.88	0.92	0.86	0.83	0.89	0.92	0.90
RH früheste Fertigstellung & RAS-2	0.70	0.44	0.73	0.80	0.64	0.59	0.75	0.61	0.66
RH früheste Fertigstellung & RAS-1	0.48	0.44	0.54	0.45	0.59	0.22	0.53	0.45	0.46
RH KBZ des Gridjobs & RAS-3	0.97	0.86	0.81	0.69	0.94	0.78	0.81	0.62	0.81
RH KBZ des Gridjobs & RAS-2	0.74	0.42	0.63	0.53	0.66	0.50	0.68	0.39	0.57
RH KBZ des Gridjobs & RAS-1	0.47	0.41	0.46	0.28	0.57	0.24	0.54	0.26	0.40
RH KBZ des Anwendungsjobs & RAS-3	0.90	0.88	0.82	0.70	0.86	0.83	0.77	0.70	0.81
RH KBZ des Anwendungsjobs & RAS-2	0.70	0.41	0.70	0.56	0.64	0.51	0.57	0.46	0.57
RH KBZ des Anwendungsjobs & RAS-1	0.48	0.44	0.57	0.31	0.59	0.24	0.49	0.43	0.44
früheste Fertigstellung & RAS-3	0.45	0.42	0.35	0.56	0.45	0.41	0.47	0.51	0.45
früheste Fertigstellung & RAS-2	0.58	0.52	0.38	0.72	0.51	0.43	0.56	0.69	0.55
früheste Fertigstellung & RAS-1	0.42	0.42	0.39	0.54	0.39	0.37	0.37	0.45	0.42
KBZ des Gridjobs & RAS-3	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
KBZ des Gridjobs & RAS-2	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01
KBZ des Gridjobs & RAS-1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
KBZ des Anwendungsjobs & RAS-3	0.02	0.02	0.02	0.02	0.03	0.02	0.02	0.02	0.02
KBZ des Anwendungsjobs & RAS-2	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
KBZ des Anwendungsjobs & RAS-1	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02

Zur Untersuchung der dritten Fragestellung wird das Umplanungsereignis mit 10% abgearbeiteten und neuen Gridjobs basierend auf den beiden Benchmarkszenarien *sRlD* und *lRlD* bei einer proportional steigenden Zahl von Gridjobs und Ressourcen zu Grunde gelegt. Die Bewertung basiert erstens auf dem durch GLEAM gewonnenen Fitnesszuwachs im Vergleich zum besten heuristischen Ergebnis und zweitens auf der Erfolgsrate. Letztere ist der Anteil der Läufe ohne Budgetverletzung an allen 50 Läufen pro Benchmarkparametrierung. Bild 2 zeigt die Ergebnisse. Wie erwartet, fallen Erfolgsrate und EA-Verbesserung mit zunehmender Last. Bis etwa 400 - 500 Gridjobs und 40 - 50 Ressourcen können bei beiden Benchmarkszenarien Budgetverletzungen beseitigt werden. Die relativ großen Schwankungen der EA-Verbesserungen können mit den unterschiedlichen Fähigkeiten der Heuristiken zur Vermeidung von Budgetverletzungen erklärt werden. Da Verletzungen vergleichsweise hart bestraft werden, können relativ kleine Unterschiede große Fitnessdifferenzen erzeugen, was dann seinerseits die Verbesserungsrate entsprechend schwanken lässt. In zwei Fällen erzeugen bereits die Heuristiken straffreie Schedules, sodass dem EA wenig Raum für Verbesserungen bleibt. Sie sind in Bild 2 durch ein weißes statt graues Drei- bzw. Rechteck gekennzeichnet.

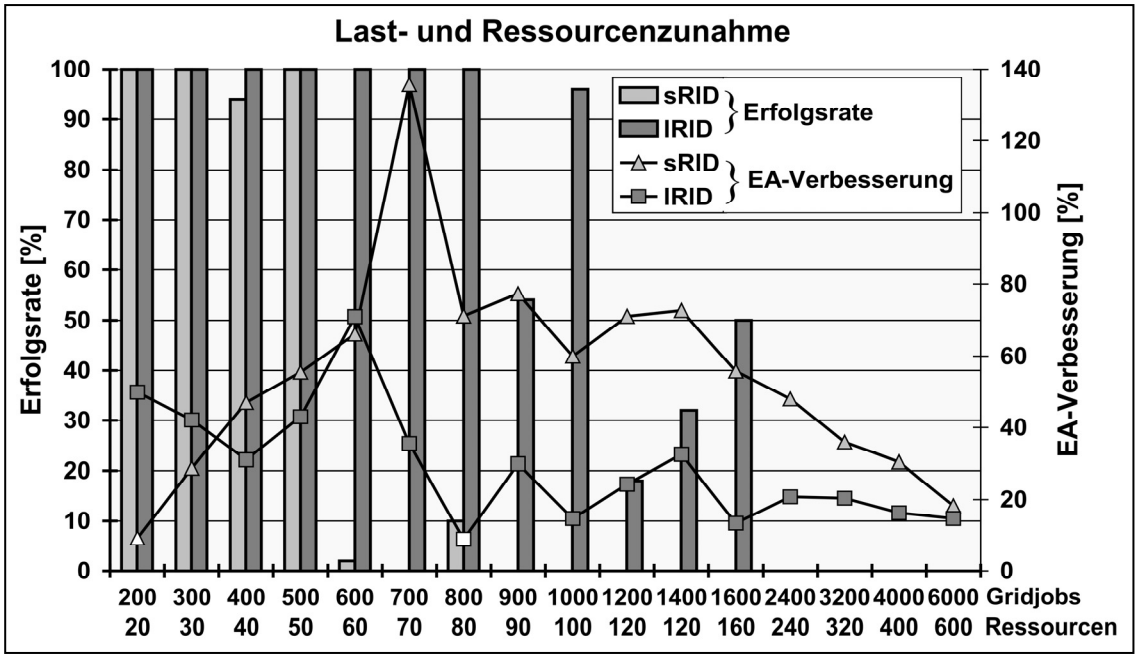

Bild 2: Auswirkungen von steigender Last auf die Erfolgsrate und die EA-Verbesserung gegenüber der besten heuristischen Planung bei beiden untersuchten Benchmarkszenarien und 10% abgearbeiteten und neuen Gridjobs.

Bild 2 zeigt auch, dass es mit wachsender Last beim *sRlD*-Benchmark im Vergleich zum *lRlD*-Fall schwieriger ist, die Budgetvorgaben einzuhalten. Es ist offenbar leichter dieses Ziel zu erreichen, wenn mehr Ressourcen zur Auswahl stehen. Selbst für mehr als 1600 Gridjobs und 160 Ressourcen können noch Verbesserungen durch den EA erreicht werden. Ab dieser Last sinkt die Verbesserungsrate kontinuierlich. Daraus kann geschlossen werden, dass selbst bei bis zu 6000 Gridjobs und 600 Ressourcen immer noch Verbesserungen unterhalb des Niveaus der vollständigen Einhaltung der Budgets durch GLEAM erreicht werden können. Mit anderen Worten, die Anzahl verletzungsfreier Anwendungsjobs wird durch den EA-Lauf gegenüber der heuristischen Planung immer noch vergrößert.

Je mehr Gridjobs geplant werden müssen und je mehr Ressourcen zur Verfügung stehen, desto größer wird die Belegungsmatrix und desto weniger Schedules können in der gegebenen Zeit von drei Minuten durchgerechnet werden. Bild 3 zeigt, wie dieser Betrag bei steigender Last kontinuierlich sinkt. Je größer die Anzahl alternativer Ressourcen, desto mehr Ressourcen müssen durch die RAS geprüft werden. Das dauert länger und erklärt die geringeren Werte für *lRlD* im Vergleich zu *sRlD*. Am Ende sinkt die Anzahl möglicher Evaluationen so stark ab, dass die Populationsgröße auf 20 bis 30 reduziert werden muss, um wenigstens noch etwa zwei Dutzend Generationen durchrechnen zu können. Dies sind geringe Werte für einen EA ohne Unterstützung durch ein lokales Suchverfahren, siehe [18]. Daraus ergibt sich auch, dass die vorliegende Implementierung kaum größere Lasten wird verkraften können. Abhilfe kann hier zum Beispiel eine verbesserte Implementierung der Evaluation oder ein schnellerer Rechner schaffen.

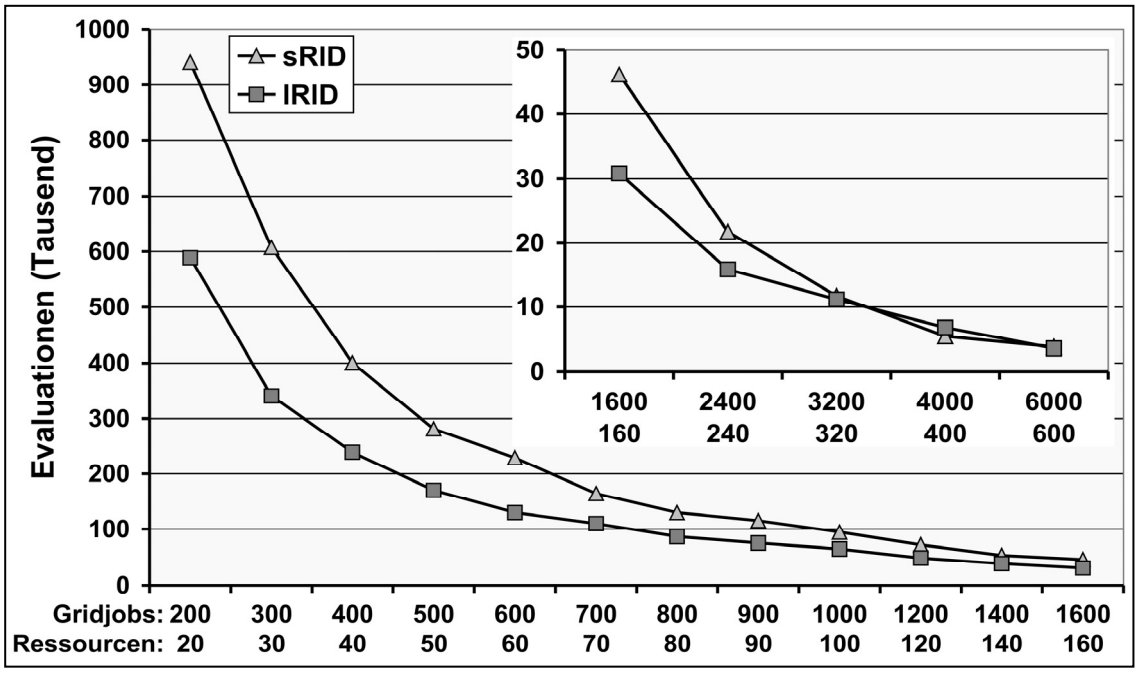

Bild 3: Innerhalb von drei Minuten bewertbare Schedules bei wachsender Last. Zur besseren Darstellung wird die Kurve ab 2400 Gridjobs getrennt mit angepasster Skalierung der Y-Achse dargestellt.

5 Zusammenfassung und Ausblick

Es konnte gezeigt werden, dass das Scheduling-Problem von Gridjobs unter realistischen Voraussetzungen komplexer ist als das klassische NP-vollständige Job-Shop-Scheduling Problem. Beim Gridjob-Scheduling kommen zur klassischen Aufgabenstellung noch alternative und heterogene Ressourcen, Ressourcen-Koallokation, Workflows mit paralleler Abarbeitung und eine multikriterielle Bewertung hinzu. Letzteres hat es bisher verhindert, ein lokales Suchverfahren zu finden, dass erfolgreich als Meme in die evolutionäre Suche integriert werden kann [19, 20]. Das liegt daran, dass der Effekt kleiner Änderungen, wie z.B. das Vorziehen eines Gridjob-Gens oder die Wahl einer anderen alternativen Ressource nicht abgeschätzt werden kann, ohne die Belegungsmatrix zumindest bis zu der vorgenommenen Änderung aufzubauen. Bei anderen kombina-

torischen Aufgabenstellungen, wie z.B. dem „Problem des Handlungsreisenden“ ist es dagegen leicht zu entscheiden, ob eine Teilstrecke durch eine Veränderung kürzer wird oder nicht.

Das untersuchte Problem des Reschedulings anlässlich neuer Gridjobs nach einem relativ geringen Grad der Abarbeitung des aktuellen Schedules ist einer der Standardfälle beim Scheduling und Ressourcenmanagement im Grid. Neben dem Ressourcenausfall dürfte er auch einer der schwierigeren sein. Denn beim Hinzukommen neuer Ressourcen oder beim Jobabbruch verringert sich die Last gegenüber dem ursprünglichen Schedule. Für den Fall neuer Gridjobs wurden neue Heuristiken vorgestellt, die die im alten Plan enthaltene Information ausnutzen. Es konnte gezeigt werden, dass sie für den häufigeren Fall geringerer Änderungen von bis zu 20% abgearbeiteten und neuen Gridjobs das Ergebnis der heuristischen Planungsphase in der Regel deutlich verbessern.

Die verarbeitbare Last wurde für den Fall von 10% abgearbeiteten und neuen Gridjobs bei steigender Gridjob- und Ressourcenanzahl untersucht. Bis zu einer Last von 6000 Gridjobs und 600 Ressourcen konnten mit Hilfe des EA-Laufs Verbesserungen gegenüber der heuristischen Planung erzielt werden, auch wenn diese gegen Ende hin immer geringer ausfielen. Da bei der maximalen Last nur noch etwa zwei Dutzend Generationen bei einer kleinen Population von 20 oder 30 Individuen innerhalb der gewählten drei Minuten möglich war, kann die erfolgreiche Bearbeitung größerer Lasten mit der gegebenen Soft- und Hardwarekonfiguration (ein Prozessor eines AMD Athlon 64 4400+, 2.0 GHz) nicht erwartet werden. Ein Tuning der ersten prototypischen Implementierung oder schnellere Hardware könnte hier Abhilfe schaffen oder bei gegebener Last und Zeit bessere Ergebnisse liefern. Letzteres ist deshalb zu erwarten, da der EA ab etwa 200 Gridjobs bei weitem noch nicht in die Konvergenzphase gelangt.

Aktuelle Arbeiten haben die Untersuchung weiterer Heuristiken und Schedulingverfahren zum Gegenstand. Dies soll auch die Grundlage für ein adaptives GORBA liefern, dass entsprechend dem aktuellen Gridzustand geeignete Heuristiken und Optimierungsverfahren auswählt und durch Erfolg und Misserfolg lernt.

6 Literatur

[1] Foster, I., Kesselman, C.: *The Anatomy of the Grid: Enabling Scalable Virtual Organisations*. Int. J. of Supercomputer Applications, vol. 15(3), 2001, S.200-222.

[2] Brucker, P.: *Scheduling Algorithms*. Springer, Berlin Heidelberg, 2004.

[3] Brucker, P.: *Complex Scheduling*. Springer, Berlin Heidelberg, 2006.

[4] Jakob, W., Hahnenkamp, B., Quinte, A., Stucky, K.-U., Süß, W.: *Schnelles Scheduling mit Hilfe eines hybriden Evolutionären Algorithmus*. Automatisierungstechnik (57)3, Oldenbourg-Verlag, 2009. DOI:10.1524/auto.2009.0759

[5] Setamaa-Karkkainen, A., Miettinen, K., Vuori, J.: *Best Compromise Solution for a New Multiobjective Scheduling Problem*. Computers & Operations Research, vol. 33(8), 2006, S.2353-2368.

[6] Wieczorek, M., Hoheisel, A., Prodan, R.: *Taxonomy of the Multi-criteria Grid Workflow Scheduling Problem*. In: Talia, D., Yahyapour, R., Ziegler, W. (eds.): Grid Middleware and Services - Challenges and Solutions. Springer, New York, 2008, S.237-264. DOI:10.1007/978-0-387-78446-5_16

[7] Tsiakkouri, E., Sakellariou, S., Dikaiakos, M.D.: *Scheduling Workflows with Budget Constraints*. In: Gorlatch, S., Danelutto, M. (eds.): Conf. Proc. CoreGRID Workshop “Integrated Research in Grid Computing”, 2005, S.347-357.

[8] Yu, J., Buyya, R.: *A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms*. In: Conf. Proc. 15th IEEE Int. Symp. on High Performance Distributed Computing (HPDC 2006), IEEE CS Press, Los Alamitos, 2006.
[9] Kurowski, K., Nabrzyski, J., Oleksiak, A., Weglarz, J.: *Scheduling Jobs on the Grid - Multicriteria Approach*. Computational Methods in Science and Technology 12(2), Scientific Publishers OWN, Polen, 2006, S.123-138
[10] Giffler, B., Thompson, G.L.: *Algorithms for Solving Production Scheduling Problems*. Operations Research 8, 1960, S.487-503.
[11] Neumann, K., Morlock, M.: *Operations Research*. Carl Hanser, München, 2002.
[12] Jakob, W., Quinte, A., Stucky, K.-U., Süß, W.: *Fast Multi-objective Scheduling of Jobs to Constrained Resources Using a Hybrid Evolutionary Algorithm*. In: Rudolph, G., et al. (eds.): PPSN X, LNCS 5199, Springer, Berlin, 2008, S.1031-1040.
[13] Jakob, W., Quinte, A., Stucky, K.-U., Süß, W., Blume, C.: *Schnelles Resource Constrained Project Scheduling mit dem Evolutionären Algorithmus GLEAM*. In: Mikut, R., Reischl, M. (Hrsg.): Conf. Proc. 17. Workshop Computational Intelligence, Universitätsverlag Karlsruhe, S.152-164, 2007.
[14] Blume, C.: *GLEAM - A System for Intuitive Learning*. In: Schwefel, H.P., Männer, R. (eds.): Proc. of PPSN I, LNCS 496, Springer, Berlin, 1991, S.48-54
[15] Blume, C., Jakob, W.: *GLEAM – An Evolutionary Algorithm for Planning and Control Based on Evolution Strategy*. In: Cantú-Paz, E. (ed.): GECCO 2002, vol. LBP, 2002, S.31-38.
[16] Blume, C.: *GLEAM - Ein EA für Prozessabläufe am Beispiel von Steuerungen für Industrieroboter*. In: Mikut, R., Reischl, M. (Hrsg.): Conf. Proc. 16. Workshop Computational Intelligence, Universitätsverlag Karlsruhe, S.11-24, 2006.
[17] Davis, L. (eds): *Handbook of Genetic Algorithms*. V. Nostrand Reinhold, New York, 1991.
[18] Jakob, W.: *Towards an Adaptive Multimeme Algorithm for Parameter Optimisation Suiting the Engineers' Need*. In: Runarsson, T. P., et al. (eds.): PPSN IX, LNCS 4193, Springer, Berlin, 2006, S.132-141.
[19] Hahnenkamp, B.: *Integration anwendungsneutraler lokaler Suchverfahren in den adaptiven memetischen Algorithmus HyGLEAM für komplexe Reihenfolgeoptimierung*. Diplomarbeit, Uni Karlsruhe, Fak. f. Wirtschaftswissenschaften, AIFB, 2007
[20] Sonnleithner, D.: *Integration eines Giffler-Thompson-Schedulers in GORBA*. Studienarbeit, Uni Karlsruhe, Fak. f. Maschinenbau, AIA, 2008