In-Grid

Innovative Grid-Entwicklungen für ingenieurtechnische Anwendungen

AP 3.3: Verteilte simulationsbasierte Produkt- und Prozessoptimierung

Bericht 3.3.2:

- Beschreibung der entwickelten Optimierungsverfahren
- Anwendung der Verfahren auf zwei realistische Probleme des Anwendungsgebiets 2.5 (Grundwasser)
- Entwurf des entsprechenden Grid Services

Projektpartner:

Universität Siegen

Institut für Wirtschaftsinformatik

Hölderlinstr. 3

57068 Siegen

Bearbeiter:

Dipl.-Ing. Frank Thilo

1. Inhalt

Nachdem im ersten Jahr des Projekts im Arbeitspaket „3.3 - Verteilte simulationsbasierte Produkt- und Prozessoptimierung“ typische Anwendungsszenarien analysiert und darauf basierend drei erste, für diese Szenarien geeignete, Optimierungsverfahren in einer Testumgebung implementiert und analysiert worden waren, wurden in den Monaten 13-18 fünf weitere Suchverfahren eingebunden. Zusätzlich wurde eine prototypische Anbindung der Optimierung an das Grundwassersimulationspaket FEFLOW (AP 2.5) und den Gießprozesssimulator CASTS (AP 2.2) entwickelt.

Sowohl die schon in Bericht 3.3.1 [1] erläuterten als auch die fünf zusätzlichen Verfahren werden im Folgenden kurz beschrieben. Die grundsätzliche Funktionsfähigkeit und Skalierbarkeit der Verfahren in einer verteilten Gridumgebung wird anhand der bereits in Bericht 3.3.1 verwendeten Testfunktion belegt. Darüberhinaus werden Optimierungsergebnisse für zwei Problemstellungen aus der Praxis des Grundwassermanagements präsentiert. Abschließend werden erste Entwurfsaspekte einer Umsetzung der Verfahren als Service in einer SOA-Architektur im Rahmen einer Globus-Toolkit-4-basierten Gridinfrastruktur genannt.

2. Anforderungen an die Verfahren

In [1] wurden vier Problemklassen identifiziert, von denen die ersten drei (optimaler Entwurf, optimale Steuerung, optimaler Entwurf hybrider Steuerungssysteme) in diesem Arbeitspaket behandelt werden, während die vierte (optimaler Entwurf diskreter mehstufiger Prozesse) Teil des AP 2.3 ist. Die drei relevanten Problemklassen lassen sich aus Sicht des Algorithmus alle auf ein statisches Entwurfsproblem zurückführen und können daher prinzipiell einheitlich behandelt werden. Dabei ist eine feste Menge von reellwertigen Entscheidungsvariablen vom Algorithmus derart optimal zu belegen, dass eine ebenfalls reellwertige Zielfunktion einen minimalen (oder maximalen) Wert annimmt. Für die Entscheidungsvariablen ist dabei ein gültiges Intervall mit Unter- und Obergrenzen gegeben. Weiterhin sind nur solche Lösungen gültig, die bestimmte Nebenbedingungen erfüllen.

Im allgemeinen Fall muss davon ausgegangen werden, dass die Zielfunktion multimodal ist, d. h. mehrere Optima aufweist. Daher sollte das Verfahren versuchen, einen möglichst großen Raum abzusuchen, um nicht im ersten lokalen Minimum hängenzubleiben.

Weder für die Zielfunktion noch für die Nebenbedingungen können dabei Annahmen gemacht werden, sie müssen vom Algorithmus als Black Box behandelt werden. Insbesondere ist i. A. keine Linearität gegeben und Ableitungsinformationen liegen nicht vor. Die Bestimmung von Gradienten auf Basis von Differenzenquotienten ist möglich, aber sehr zeitaufwendig und kann aufgrund von numerischem Rauschen mit hohem Fehler behaftet sein.

Durch die in der Regel lange Laufzeit der einzelnen Simulationen ist es entscheidend, dass das Optimierungsverfahren eine große Anzahl von Auswertungen simultan vornehmen kann, um durch diese Parallelisierung die Gesamtlaufzeit auf ein praktikables Maß zu drücken.

Bei einem Einsatz in einer Gridumgebung muss davon ausgegangen werden, dass heterogene Ressourcen zum Einsatz kommen. Dies ist für die Optimierung insbesondere bzgl. der Laufzeitunterscheide der einzelnen Simulationsberechnungen von Bedeutung. Daher sollte das Verfahren nach Möglichkeit weitestgehend asynchron arbeiten, um nicht durch einzelne, spät eintreffende Ergebnisse stark verzögert zu werden.

Um auch große Probleme lösen zu können, sollte das Verfahren darüber hinaus auch auf eine große Anzahlen von Prozessoren nutzen können, wenn die Anzahl der Entscheidungsvariablen wächst.

Weiterhin sollte das Verfahren im Sinne einer serviceorientierten Architektur möglichst generisch und robust sein, um auf verschiedene Probleme anwendbar zu sein, ohne dass Parameter des Optimierungsverfahrens zeitaufwendig manuell an die Problemstellung angepasst werden müssen.

Aus den oben skizzierten Anforderungen ergibt sich, dass die zu entwickelnden Optimierungsverfahren mit beliebigen Black-Box-Funktionen umgehen können müssen und keine expliziten Ableitungsinformationen verwenden sollten. Die Klasse der sogenannten direkten Suchverfahren erfüllt diese Anforderungen. Zusätzlich müssen allgemeine Nebenbedingungen unterstützt werden.

Das Verfahren muss in dem Sinne verteilt sein, als dass es simultan mehrere Auswertungen anfordern können muss. Dabei sollte der effektiv einsetzbare Parallelitätsgrad vor allem bei größeren Problemen hoch sein. Wünschenswert ist, dass der Algorithmus möglichst wenige Synchronisationspunkte aufweist, die den weiteren Fortschritt verhindern, wenn noch Simulationsergebnisse ausstehen.

3. Entwickelte Verfahren

Auf Basis der oben dargestellten Anforderungen wurden prototypisch acht Verfahren weiterentwickelt bzw. ausgewählt, die die Anforderungen weitestgehend erfüllen. Im Folgenden werden diese Verfahren kurz skizziert.

3.1. Distributed Polytope Search (DPS)

DPS [2] gehört zur Klasse der direkten Suchverfahren. Die Suche läuft in drei Phasen ab: Initialisierung, Exploration und lokale Suche. In der Initialisierungsphase werden zunächst durch eine globale Zufallssuche $2n$ (n ist die Anzahl der Entscheidungsvariablen) Punkte erzeugt, die die Nebenbedingungen erfüllen. In der Explorationsphase werden durch die geometrischen Operationen Spiegelung am und Kontraktion zum gewichteten Schwerpunkt des Polytops neue Lösungskandidaten erzeugt. Ungültige Punkte, die Nebenbedingungen verletzen, werden durch einen Reparaturmechanismus in Richtung des gültigen Bereichs verschoben.

Die besten so gefundenen Punkte bilden den Polytop für die nächste Iteration, bis die Punkte schließlich so weit konvergiert sind, dass auf eine einfache lokale Suche umgeschaltet wird, die versucht, die Lösung noch leicht zu verbessern.

Durch zwei Parameter, die bestimmen, wie viele Punkte des Polytops in jedem Schritt wie oft gespiegelt und kontrahiert werden, kann sich der Algorithmus an verschiedene gewünschte Parallelitätsgrade anpassen.

3.2. Parallel Scatter Search (PSS)

Die Meta-Heuristik Scatter Search [3] beschreibt einzelne Komponenten eines Suchverfahrens, die auf unterschiedliche Weise implementiert werden können. Das hier eingesetzte Verfahren PSS ist eine solche Implementierung, bei der die erzeugten Lösungskandidaten simultan ausgewertet werden.

Den Kern das Algorithmus bildet das sogenannte *Reference Set*, welches in dieser Implementierung aus 20 Lösungskandidaten besteht. Als *Diversifikationsverfahren* wird eine gesteuerte Zufallssuche verwendet, um das initiale Reference Set aufzubauen sowie bei einer vorzeitigen Konvergenz der Punkte die Streuung wieder zu erhöhen.

Die zweite Komponente ist eine *Subset-Erzeugungsmethode*, die aus den bestehenden Punkten Paare und Tripel bildet. Auf jede dieser Untermengen wird dann ein *Kombinationsverfahren* angewendet, welches die bestehenden Punkte zu neuen Punkten kombiniert. Typischerweise entstehen so 100-400 neue Punkte, die simultan ausgewertet werden können. Nach der Auswertung werden die ungültigen Punkte mittels eines Verfahrens namens *Path Relinking* modifiziert und wieder in die Auswertungsschlange eingefügt.

Aus den gültigen alten und neuen Punkten wird anhand von Regeln, die sowohl die Lösungsqualität als auch die Streuung berücksichtigen, das neue Reference Set aufgebaut. Wenn diese ausgewählten Punkte bezüglich des Zielfunktionswerts nicht mehr hinreichend voneinander abweichen, so wird versucht, die Streuung durch das schon zur Initialisierung verwendete Diversifikationsverfahren zu erhöhen. Nach einer vorgegebenen Anzahl solcher Schritte terminiert das Verfahren schließlich.

3.3. Asynchronous Parallel Pattern Search (APPS)

APPS gehört zur Klasse der Pattern-Search-Verfahren. Dieses Verfahren wurde nicht selbst implementiert, sondern es wurde die Software APPSPACK 4.0 [4] verwendet und eine Anbindung an die Optimierungstestumgebung geschaffen.

Das Verfahren beginnt in einem Startpunkt und erzeugt neue Punkte, indem es Vektoren aus einer Menge von Suchrichtungen auf den Punkt addiert. Im Standardfall sind diese Vektoren parallel zu den Achsen, jeweils in negativer und positiver Richtung, also insgesamt $2n$ Vektoren, die ein rechtwinkliges Gitter aufspannen. Die Punkte werden dann simultan ausgewertet. Der beste gültige der so neu erzeugten Punkte dient dann wiederum als Startpunkt für den nächsten Schritt. Ungültige Punkte werden verworfen. Wurde kein besserer Punkt gefunden, so wird die Schrittweite verringert, bis eine definierte Minimalschrittweite erreicht ist, womit das Verfahren terminiert.

Wenn das Verfahren im asynchronen Modus betrieben wird, blockiert der Algorithmus nicht, bis alle $2n$ Punkte ausgewertet sind, sondern setzt die Suche fort, sobald ein besserer neuer Punkt gefunden wurde, auch wenn noch Auswertungen ausstehen. Dies erhöht die durchschnittliche Auslastung der verwendeten Rechenressourcen.

3.4. Simulated Annealing (SA)

Simulated Annealing (simulierte Abkühlung) [5] ist ein klassisches heuristisches Optimierungsverfahren. Die Grundidee ist inspiriert durch den Abkühlvorgang eines heißen Metalls: Wird die Temperatur hinreichend langsam abgesenkt, so führt die zufällige thermische Bewegung der Moleküle dazu, dass das Material zu einem nahezu energieoptimalen Zustand auskristallisiert.

Auf das Suchverfahren übertragen bedeutet dies, dass in jedem Schritt eine bestehende Lösung durch eine zufällige Perturbationsfunktion modifiziert wird, um eine neue Lösung zu erzeugen. Ist die Qualität der neuen Lösung besser als die der alten, so wird die neue Lösung akzeptiert und das Verfahren geht in den nächsten Durchlauf. Ist die Qualität hingegen geringer, so wird die Lösung dennoch mit einer bestimmten Wahrscheinlichkeit akzeptiert, um aus lokalen Minima des Suchraums zu entkommen. Diese Wahrscheinlichkeit ist dabei abhängig von einem Parameter, eben der „Temperatur“. Die Temperatur wird dabei im Laufe des Verfahrens kontinuierlich verringert, so dass die Wahrscheinlichkeit der Akzeptanz schlechterer Lösungen immer weiter sinkt.

Die hier implementierte Variante [6] erzeugt dabei im Gegensatz zum klassischen, sequentiellen Algorithmus simultan mehrere neue Lösungskandidaten, indem die zufällige Perturbationsfunktion mehrfach auf die aktuelle Lösung angewendet wird.

3.5. Great Deluge Algorithm (GDA)

Der Great Deluge Algorithm (große Sintflut) [7] funktioniert weitestgehend analog zu Simulated Annealing. Der Unterschied besteht in der sekundären Akzeptanzfunktion, also in der Entscheidung, wann eine qualitativ schlechtere Lösung trotzdem akzeptiert wird, um lokalen Minima zu entkommen. Bei GDA wird hierzu der Zielfunktionswert mit einem imaginären „Wasserstand“ verglichen. Im Falle eines Maximierungsproblems wird die Lösung immer dann akzeptiert, wenn sie oberhalb des Wasserstandes liegt. Im Laufe der Suche wird der Wasserstand dann langsam angehoben, um eine höhere Qualität der Lösungen zu erzwingen.

Ähnlich wie bei SA wurde das Verfahren parallelisiert [6], indem mehrere neue Lösungskandidaten simultan erzeugt werden, die dann gleichzeitig ausgewertet werden können.

3.6. Genetic Algorithms (GA)

Ein Genetic Algorithm (genetischer Algorithmus) [8] ist ein populatonsbasiertes, heuristisches Optimierungsverfahren. Das grundsätzliche Vorgehen basiert auf der biologischen Evolution von Arten in der Natur. Wichtige Elemente davon finden sich auch bei GA wieder: In einer Initialisierungsphase wird eine Menge von Lösungskandidaten erzeugt. Man spricht hier auch von einer Population von Individuen, die Belegung der Entscheidungsvariablen entspricht dem Genom. Neue Individuen werden nun erzeugt, indem Mutations- und Rekombinations-Schritte auf die bestehende Population angewendet werden.

Ein Mutationsschritt erzeugt aus genau einem Individuum ein neues, indem eine Kopie erzeugt wird und dabei die Gene zufällig verändert werden. Bei der Rekombination hingegen wird ein neues Individuum erzeugt, indem die Genome von zwei (oder mehreren) Eltern kombiniert werden. Dabei gibt es sowohl für die Mutation als auch für die Rekombination eine Vielzahl von möglichen Operationen. Die durch Mutation und Rekombination neu erzeugten Individuen werden dann ausgewertet, um den jeweiligen Zielfunktionswert (die Fitness) zu bestimmen. Anschließend erfolgt ein Selektionsschritt zur Auswahl der Individuen für die nächste Generation der Population.

Wenn wie im Falle der simulationsbasierten Optimierung die Bestimmung der Fitness der zeitbestimmende Faktor ist, kann das Verfahren parallelisiert werden, indem die in jedem Schritt erzeugte Menge von neuen Individuen parallel ausgewertet wird. Eine genaue

Beschreibung der für die Experimente verwendeten Mutations-, Rekombinations- und Selektionsverfahren findet sich in [6].

3.7. Evolution Strategies (ES)

Evolution Strategies (Evolutionsstrategien) [9] sind GA sehr ähnlich. Die wesentliche Innovation ist die Einführung sogenannter Strategieparameter. Dabei wird die Menge der zu optimierenden Variablen um zusätzliche Parameter erweitert, so dass ein höherdimensionales Optiierungsproblem entsteht. Diese Variablen haben aber keinen Einfluss auf die Fitness der Individuen, sondern beeinflussen ausschließlich Operationen des Algorithmus selbst, die auf das entsprechende Individuum angewendet werden. Typischerweise bestimmt ein solcher Strategieparameter z.B. die Stärke der Mutationsoperation. Die Idee ist, dass das Suchverfahren dadurch selbstkalibrierend wird und im Laufe der Optimierung lernt, welche Parameter erfolgreich gute Lösungen hervorbringen.

Der Parallelisierungsansatz ist derselbe wie bei GA. Details zur verwendeten Implementierung finden sich wiederum in [6].

3.8. Particle Swarm Optimization (PSO)

Particle Swarm Optimization (Partikelschwarmoptimierung) [10] ist ein weiteres populationsbasiertes, heuristisches Suchverfahren. Der Algorithmus simuliert einen Schwarm von Partikeln, die sich durch den Suchraum bewegen. Jeder Partikel hat dabei eine aktuelle Position sowie einen Geschwindigkeitsvektor. Beide Werte werden in einer Initialisierungsphase beispielsweise zufällig bestimmt. Die Position der Partikel wird in jedem Schritt angepasst, indem der Geschwindigkeitsvektor auf die alte Position addiert wird. Die Partikel beeinflussen sich dabei gegenseitig, da solche Partikel mit hoher Fitness eine anziehende Wirkung auf benachbarte Partikel ausüben. Dadurch werden deren Geschwindigkeitsvektoren in einem gewissen Umfang angepasst, so dass sich gekrümmte Kurven durch den Suchraum ergeben. Neben der lokalen Anziehungswirkung ist es auch möglich, den Partikel mit der besten Lösungsqualität als globalen Attraktor einzusetzen.

Die simultane Auswertung mehrerer Lösungskandidaten ist wiederum sehr einfach möglich, indem die Fitness jedes Partikels unabhängig voneinander berechnet wird. Der maximale Parallelisierungsgrad wird dabei durch die Größe des Schwarms begrenzt, so dass es sich anbietet, die Anzahl der Partikel mindestens auf die Anzahl der verfügbaren CPUs zu setzen [6].

4. Ergebnisse

Im Folgenden wird zunächst kurz die Softwareumgebung OpTiX beschrieben, in die die Algorithmen integriert worden sind. Anschließend werden mit dieser Software erzeugte Ergebnisse für die beschriebenen Algorithmen und unterschiedliche Problemstellungen dargestellt und diskutiert. Dabei wird mit der verallgemeinerten Rosenbrock-Funktion zunächst ein Testproblem gelöst, bevor anschließend zwei unterschiedlich komplexe wasserwirtschaftliche Optimierungsprobleme vorgestellt werden.

4.1. Softwarebeschreibung

Die acht oben skizzierten Verfahren wurden in Standard-C++ implementiert und in die zuvor geschaffene Optimierungsumgebung OpTiX [1] integriert Diese stellt den Algorithmen u.a. eine einheitliche abstrakte Schnittstelle zum Zugriff auf die Problemstellung zur Verfügung. Um das grundsätzliche Verhalten der Algorithmen zu analysieren, stehen in der Umgebung verschiedene einfache Testfunktionen als Optimierungproblemstellungen zur Verfügung. Da die Berechnung dieser Testfunktionen kaum Rechenzeit benötigt, sind auch ohne großen Ressourceneinatz umfassende Testläufe möglich. Weiterhin lässt sich bei einigen Testfunktionen die Dimensionalität anpassen, so dass auch dieser Aspekt untersucht werden kann.

Um auch mit den Testfunktionen, die zur Auswertung selbst nahezu keine Rechenzeit benötigen, Verteilungsaspekte wie Skalierbarkeit testen zu können, wurde eine Umgebung zur verteilten Ausführung realisiert. Dabei dienen künstlich verlangsamte Testfunktionen als Stellvertreter für die eigentlich zu untersuchenden Optimierungsprobleme auf Basis der Simulationspakete aus AP 2.x.

Zusätzlich wurde zur schnelleren Ergebnisgewinnung eine „virtuelle Gridumgebung“ geschaffen, in der dieselben Testfunktionen lokal ausgewertet werden und die Verteilung lediglich auf Basis einer ereignisgesteuerten Simulation emuliert wird. Die Laufzeit der Auswertungen kann dabei mit einer Zufallsverteilung belegt werden, um Laufzeit-schwankungen in einer heterogenen Umgebung zumindest ansatzweise nachzubilden und deren Auswirkung auf die Performance der Algorithmen zu untersuchen.

Neben den Testfunktionen wurden prototypische Anbindungen zur Gießsimulation CASTS (AP 2.2) und Grundwassersimulation FEFLOW (AP 2.5) implementiert. Damit ist es möglich, innerhalb einer Clusterumgebung entsprechende Optimierungsprobleme verteilt zu lösen, d.h. unter simultanem Einsatz mehrerer Simulationsinstanzen auf verschiedenen Rechnerknoten des Clusters. Während die Anbindung an CASTS sehr einfach gehalten ist und im Wesentlichen aus dateigebundene Datenaustausch besteht, ist die Kopplung an FEFLOW mit Hilfe der dort vorhandenen Programmierschnittstelle IFM (Interface Manager) vorgenommen worden und erlaubt komplexe Eingriffe in die Simulation zur Laufzeit.

Die Software selbst liegt als komprimiertes tar-Archiv vor und enthält alle notwendigen C++-Quellcodedateien. Eine kurze Installationsanleitung und ein Überblick über die Struktur des Codes ist als Textdatei im Archiv enthalten.

4.2. Testfunktion

Die beschriebenen Verfahren wurden mit der im vorigen Abschnitt kurz beschriebenen Softwareumgebung u.a. anhand der n-dimensionalen Rosenbrock-Funktion

$$min\, f(x) = \sum_{i=1}^{n-1} \left(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$$

getestet.

Exemplarisch werden im Folgenden die Ergebnisse für ein 10- und ein 50-dimensionales Problem mit jeweils 10 und 100 virtuellen CPUs diskutiert. Die Abbildungen stellen dabei den Verlauf der erreichten Lösungsqualität über die Zeit als Durchschnitt von 200 Einzelläufen dar.

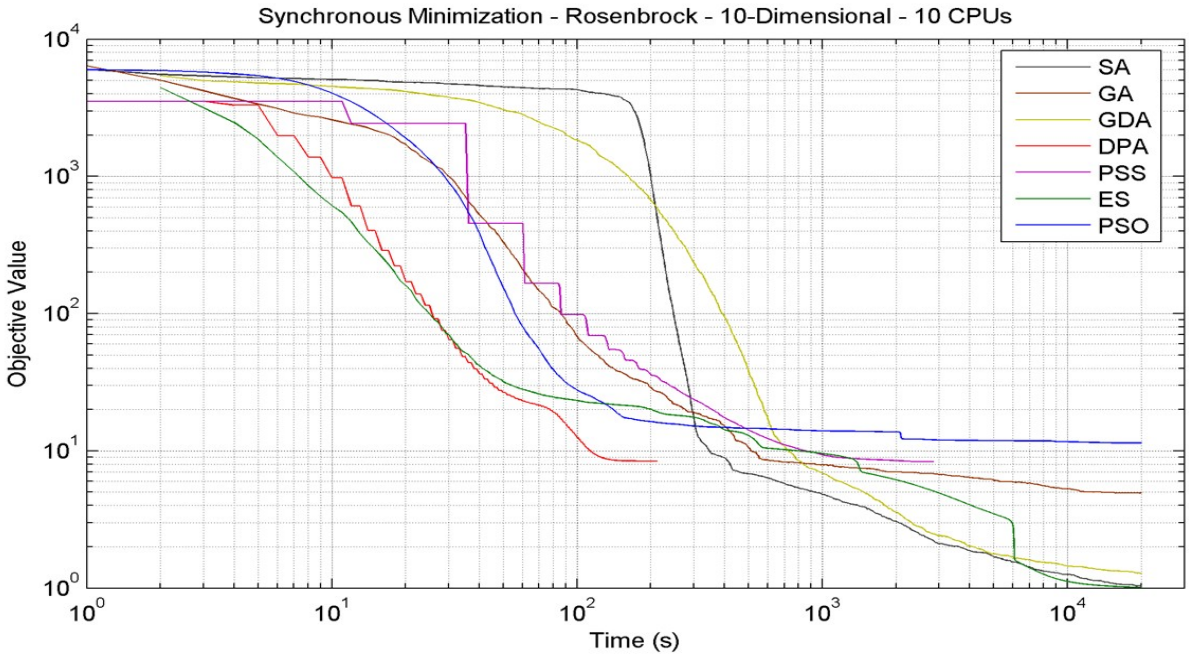

Abbildung 1: Verlauf der Lösungsqualität über die Zeit beim Lösen eines 10-dimensionalen Rosenbrock-Problems mit 10 CPUs; Durchschnitt aus 200 Läufen

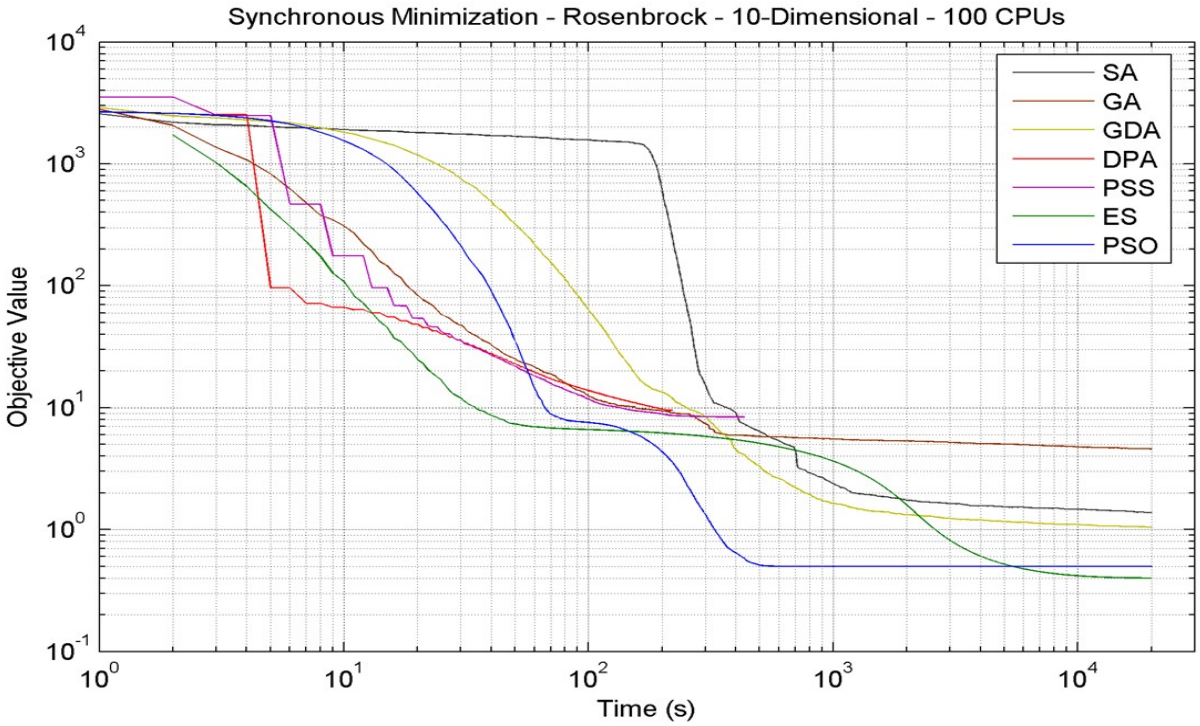

Abbildung 2: Verlauf der Lösungsqualität über die Zeit beim Lösen eines 10-dimensionalen Rosenbrock-Problems mit 100 CPUs; Durchschnitt aus 200 Läufen

In den Abbildungen 1 und 2 ist der Verlauf für ein 10-dimensionales Rosenbrock-Problem zu sehen, jeweils unter Verwendung von 10 CPUs bzw. 100 CPUs. Man erkennt die unterschiedlichen Charakteristika der Verfahren. So ist DPS im Falle von 10 CPUs der Algorithmus, der am schnellsten eine akzeptable Lösungsqualität von unter 10 erreicht. Allerdings konvergiert das Verfahren bei einigen Läufen noch bei recht hohen Zielfunktionswerten, so dass es in der hier angestellten Durchschnittsbetrachtung bzgl. der Qualität der Lösung hinter ES, SA und GDA zurückbleibt. Speziell SA und GDA benötigen eine lange „Warmlaufphase“, erreichen aber schließlich gute Lösungen.

Vergleicht man das Verhalten beim Wechsel von 10 auf 100 CPUs, so können zwar alle Verfahren von der Erhöhung der Parallelität profitieren, das Ausmaß des Gewinns unterscheidet sich aber deutlich. Legt man als Maßstab wiederum das Erreichen eines Zielfunktionswerts von unter 10 an, so wird nun dieser Wert zuerst von ES und PSO erreicht. Auch PSS und GA profitieren deutlich und erreichen den Wert nun in etwa nach der gleichen Zeit wie DPS. D.h., dass DPS bei Verwendung nur geringer Ressourcen sehr schnell Lösungen mittlerer Qualität finden kann, aber weniger gut als andere Verfahren wie ES oder PSO skaliert, sobald der Parallelitätsgrad die Problemdimension stark übersteigt.

Ein weiterer typischer Effekt ist, dass PSO mit 100 CPUs deutlich bessere Lösungen findet. Dies lässt sich durch die größere Schwarmgröße erklären. Es wäre möglich, auch bei geringerem Parallelitätsgrad mit entsprechend vielen Partikeln zu arbeiten und somit eine ähnliche Lösungsqualität zu erreichen – allerdings auf Kosten zusätzlicher Auswertungen.

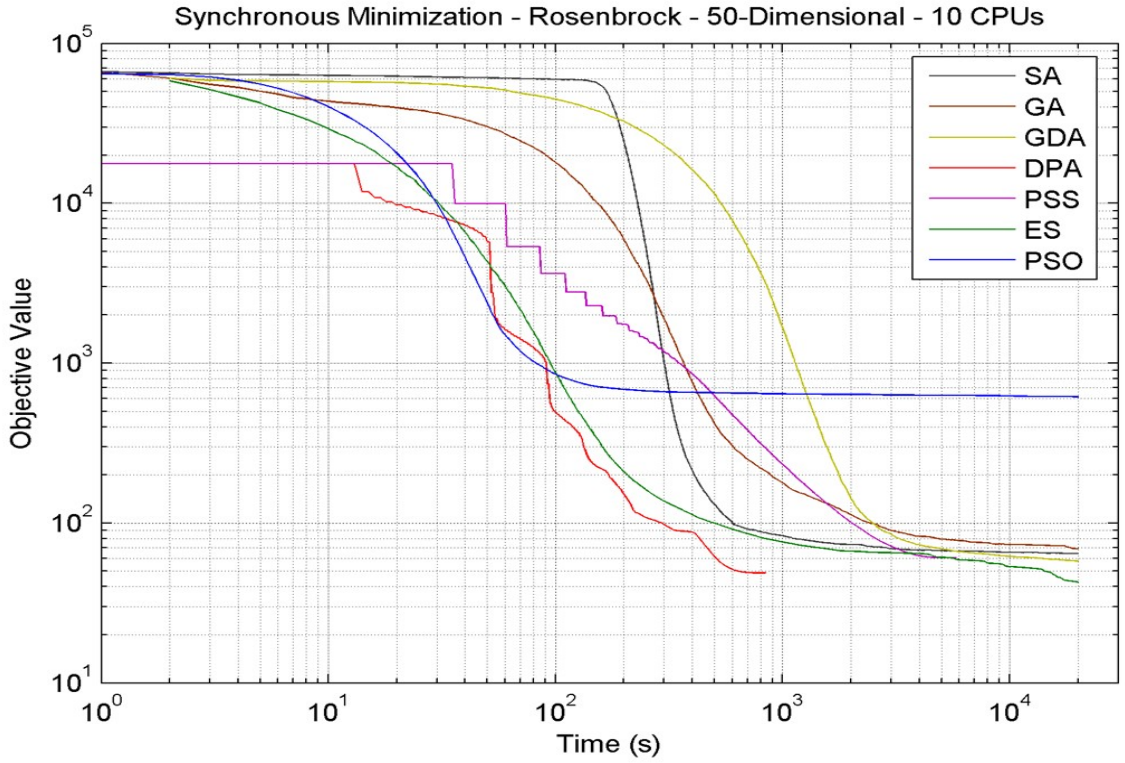

Abbildung 3: Verlauf der Lösungsqualität über die Zeit beim Lösen eines 50-dimensionalen Rosenbrock-Problems mit 10 CPUs; Durchschnitt aus 200 Läufen

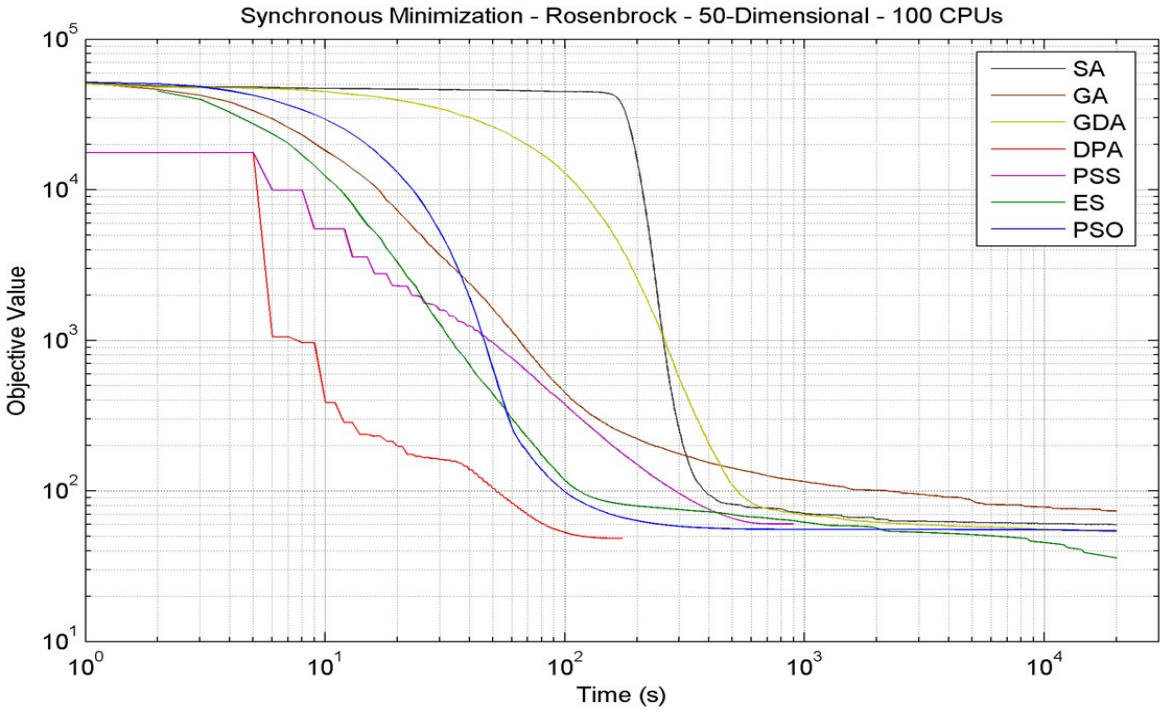

Abbildung 4: Verlauf der Lösungsqualität über die Zeit beim Lösen eines 50-dimensionalen Rosenbrock-Problems mit 100 CPUs; Durchschnitt aus 200 Läufen

Abbildungen 3 und 4 stellen dasselbe für ein nun 50-dimensionales Rosenbrock-Problem dar. Das prinzipielle Verhalten der Algorithmen ist ähnlich wie zuvor. Die Unterschiede in der erreichten Lösungsqualität sind generell geringer, allerdings bildet PSO beim Einsatz von 10 CPUs die Ausnahme, da die Schwarmgröße hier offenbar zu gering gewählt wurde. Beim zeitlichen Verhalten ist wie schon bei 10 Dimensionen das Verfahren DPS das schnellste bei geringem Parallelitätsgrad. Bei diesem höherdimensionalen Problem skaliert DPS nun allerdings viel besser und bleibt auch beim Einsatz von 100 CPUs voraus. Durch die erhöhte Schwarmgröße liefert PSO bei 100 CPUs auch gute Ergebnisse. SA und GDA benötigen wie schon beim kleineren Rosenbrock-Problem eine lange Startphase.

Für die weiteren Untersuchungen mit realistischen Problemstellungen aus den Anwendungsbereichen war eine Selektion der interessantesten Algorithmen notwendig, da die Experimente unter Verwendung von langlaufenden Simulationsläufen sehr aufwendig sind. Auf Basis der hier dargestellten und zusätzlicher Ergebnisse wurden die Algorithmen DPS, PSS, ES und PSO für die weiteren Experimente ausgewählt.

4.3. Problemstellungen aus dem Bereich des Grundwassermanagements

Vier der Algorithmen wurden verwendet, um zwei Optimierungsprobleme aus der wasserwirtschaftlichen Praxis zu lösen. Beim ersten Problem handelt es sich um ein recht einfaches Problem, das sich in die in [1] identifizierten Problemklassen als *Problem des optimalen Entwurfs* einordnen lässt, während das zweite eine anspruchsvolles Problem der Klasse *optimaler Entwurf hybrider Steuerungssysteme* ist.

Wie bereits erwähnt, wurde die Kopplung von OpTiX mit der Grundwassersimulationssoftware FEFLOW auf Basis der Programmierschnittstelle IFM durchgeführt. Dies ermöglicht es, zur Laufzeit einer Simulation sowohl Messdaten abzugreifen, als auch in den Ablauf der Simulation einzugreifen. Erst dadurch wurde es möglich, die für die zweite unten beschriebene Problemstellung (Binsheimer Feld) die dynamische Pumpensteuerung umzusetzen.

a) Charlottenburg

Das Optimierungsproblem „Charlottenburg“ besteht darin, den Auswirkungen eines Schleusenbaus auf den Grundwasserspiegel in benachbarten Gebieten entgegenzuwirken. In Abbildung 5 ist das Simulationsgebiet als Finite-Element-Netz dargestellt. Die fünf blauen Punkte im unteren Bereich repräsentieren Meßstellen, an denen ein minimaler Flurabstand (Abstand des Grundwasserspiegels zur Oberfläche) eingehalten werden muss, um einen geschützten Baumbestand zu erhalten. Die vier roten Punkte sind zu diesem Zweck installierte Pumpanlagen. Die konstante Fördermenge jeder dieser Brunnen sind die vier Entscheidungsvariablen. Die zu minimierende Zielfunktion ist die Gesamtfördermenge. Um zu bestimmen, ob bei gegebenen Förderraten die Nebenbedingungen eingehalten werden, ist jeweils ein FEFLOW-Simulationslauf notwendig, um den Flurarbstand in den fünf Meßstellen zu berechnen.

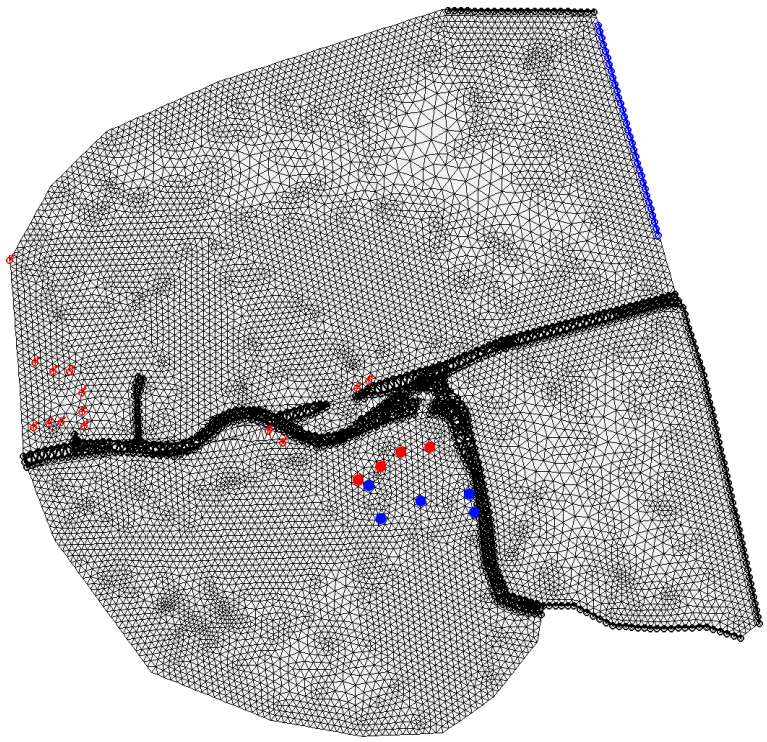

Abbildung 5: Modellgebiet des Problems „Charlottenburg“ mit den 4 zu optimierenden Pumpanlagen (rot) und den 5 Messpunkten (blau)

Abbildung 6 und 7 zeigen den Verlauf der Optimierung für die vier Algorithmen DPS, PSS, ES und PSO unter Einsatz von 6 bzw. 40 CPUs. Eine einzelne Simulation des Modells inklusive Startupzeit des Simulators benötigte auf dem verwendeten Cluster „Rubens“[1] der Universität Siegen zwischen 12 und 20 Sekunden. DPS findet wiederum beim Einsatz weniger Prozessoren am schnellsten zu guten Lösungen, ES und vor allem PSS erreichen aber nach einiger Zeit im Mittel bessere Zielfunktionswerte. ES hat in diesem Szenario Probleme und finde nur Lösungen mit durchschnittlicher Qualität.

Erhöht man die Zahl der CPUs auf 40, so nähert sich die Laufzeit der Verfahren einander stärker an. ES und DPA sind in der Anfangsphase nun etwa gleich schnell, werden aber auf lange Sicht sowohl von PSO als auch von PSS geschlagen. Insgesamt zeigt PSS hier das ausgeglichenste Verhalten.

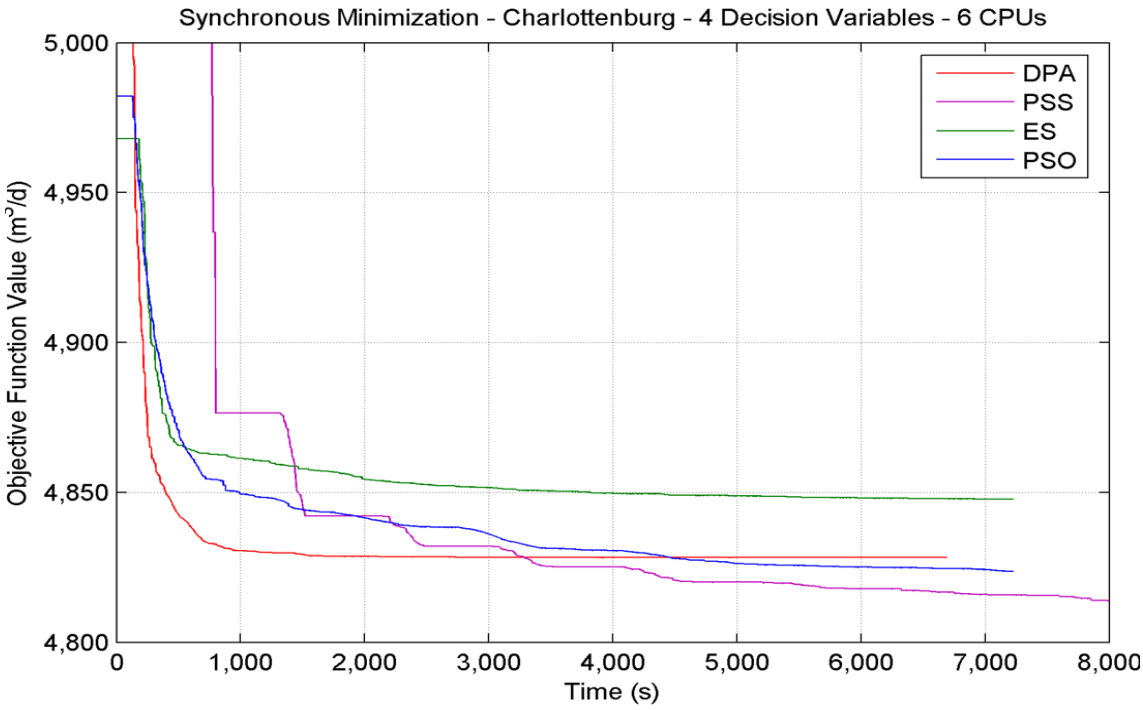

Abbildung 6: Verlauf der Lösungsqualität über die Zeit beim Lösen des 4-dimensionalen wasserwirtschaftlichen Charlottenburg-Problems mit 6 CPUs; Durchschnitt aus je 20 Läufen

1 148 Dual-Opteron-Knoten, 2 - 2.8 GHz, 2 – 4 GB RAM

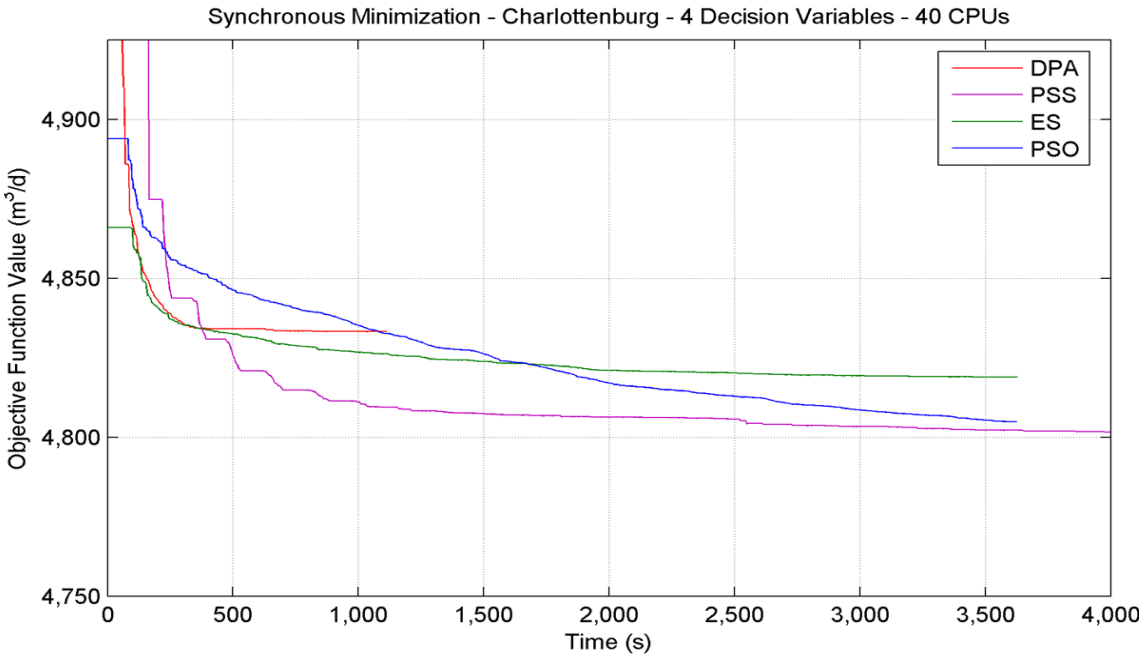

Abbildung 7: Verlauf der Lösungsqualität über die Zeit beim Lösen des 4-dimensionalen wasserwirtschaftlichen Charlottenburg-Problems mit 40 CPUs; Durchschnitt aus je 20 Läufen

b) Binsheimer Feld

Das zweite wasserwirtschaftiche Problem ist ebenso eines der Einhaltung maximaler Wasserstände, also minimaler Flurabstände. Als Folge von Bergbauaktivitäten kommt es im betrachteten Gebiet „Binsheimer Feld“ immer wieder zu Oberflächenabsenkungen und in deren Folge zur Verringerung des Flurabstandes. Um dem entgegenzuwirken, werden Brunnenanlagen betrieben, die das Grundwasser aus den kritischen Bereichen abfördern. Abbildung 8 zeigt eine schematische Ansicht des betrachteten Gebiets, das im Osten durch den Rhein begrenzt ist. Es enthält eine Vielzahl unterschiedlicher Brunnenanlagen, von denen im Rahmen der Optimierung drei betrachtet werden.

Im Gegensatz zum Charlottenburg-Problem sind die Förderraten der Pumpanlagen hier nicht statisch auf einem festen Wert. Stattdessen werden sie durch eine adaptive Regelung bestimmt. Dazu wird der Grundwasserpegel in einem Messpunkt kontinuierlich gemessen und die Förderrate des Brunnens bei Über- bzw. Unterschreiten bestimmter Schaltpegel angepasst (s. Abbildung 9). Diese Schaltpegel, die bestimmen, wann eine zusätzliche Pumpstufe hinzu- bzw. abgeschaltet wird, sind die zu optimierenden Entscheidungs-variablen, insgesamt 22. Der minimal geforderte Flurabstand ist dabei in 11 Messpunkten einzuhalten. Die zu minimierende Zielfunktion sind die Gesamtenergiekosten zum Betrieb der Anlagen über einen Zeitraum von fünf Jahren.

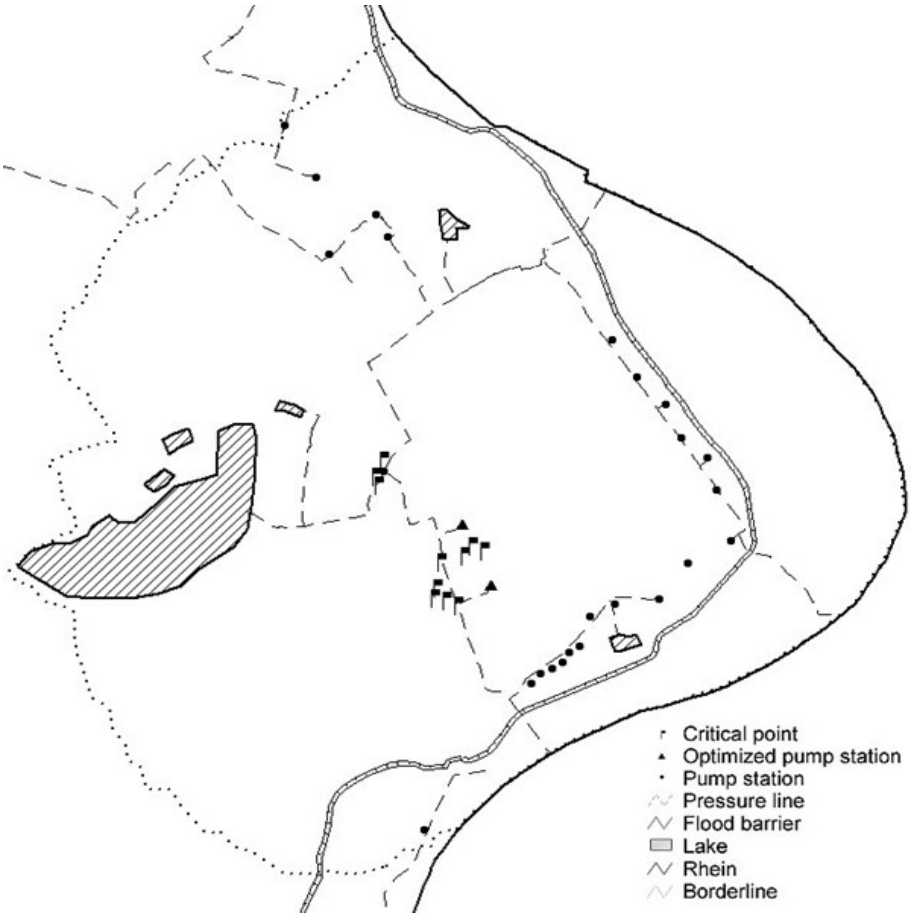

Abbildung 8: Schematische Ansicht des Gebiets „Binsheimer Feld“

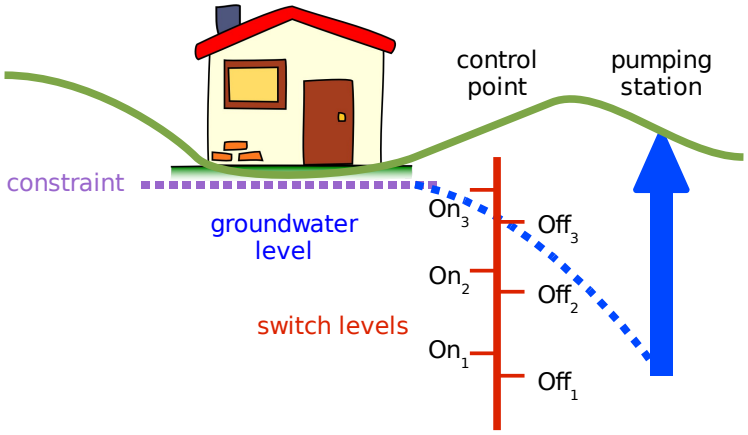

Abbildung 9: Schema der adaptiven Pumpensteuerung in Abhängigkeit des Wasserstands im Messpunkt und der zu optimierenden Schaltpegel

In Abbildung 10 und 11 ist wiederum der Verlauf der Lösungsqualität über die Zeit für die vier Algorithmen DPS, PSS, ES und PSO dargestellt. Eine einzelne Simulation benötigt dabei auf dem Rubens-Cluster zwischen 210 und 300 Sekunden. Im linken Bild unter Einsatz von 10 CPUs weist ES das beste Laufzeitverhalten auf und findet auch die qualitativ besten Lösungen. DPS und PSO zeigen ein sehr ähnliches Verhalten, während PSS deutlich mehr Zeit benötigt. Die Verläufe für PSO und ES enden nach 14000s, ohne dass bereits Konvergenz eingetreten wäre, weil aus pragmatischen Erwägungen die Laufzeit begrenzt werden musste.

Geht man nun zum im rechten Bild dargestellten Fall von 80 CPUs über, so nähert sich die Laufzeit von PSS der der anderen Algorithmen stärker an. PSO kann abermals von der Vergrößerung des Schwarms profitieren und erreicht signifikant bessere Lösungen als mit 10 CPUs.

Zusätzlich wurde hier eine asynchrone Variante von PSO eingesetzt, die im Gegensatz zum synchronen Standardalgorithmus keine feste Iterationsstruktur aufweist. Anstatt auf die vollständige Auswertung aller neu erzeugter Lösungen in jedem Schritt zu warten, wird im asynchronen Modus unmittelbar nach Verfügbarkeit des ersten Simulationsergebnisses bereits ein neuer Lösungpunkt erzeugt und dessen Auswertung gestartet. Dadurch ist die Ausnutzung der gegeben Hardwareressourcen insbesondere bei einem hohen Grad an Heterogenität deutlich höher. Selbst im Fall des hier verwendeten, recht homogenen Rubens-Clusters erzielt PSO in der asynchronen Spielart das beste Ergebnis. Für den Einsatz in der Gridumgebung ist es daher wünschenswert, den asynchronen Aspekt in möglichst allen Algorithmen umzusetzen.

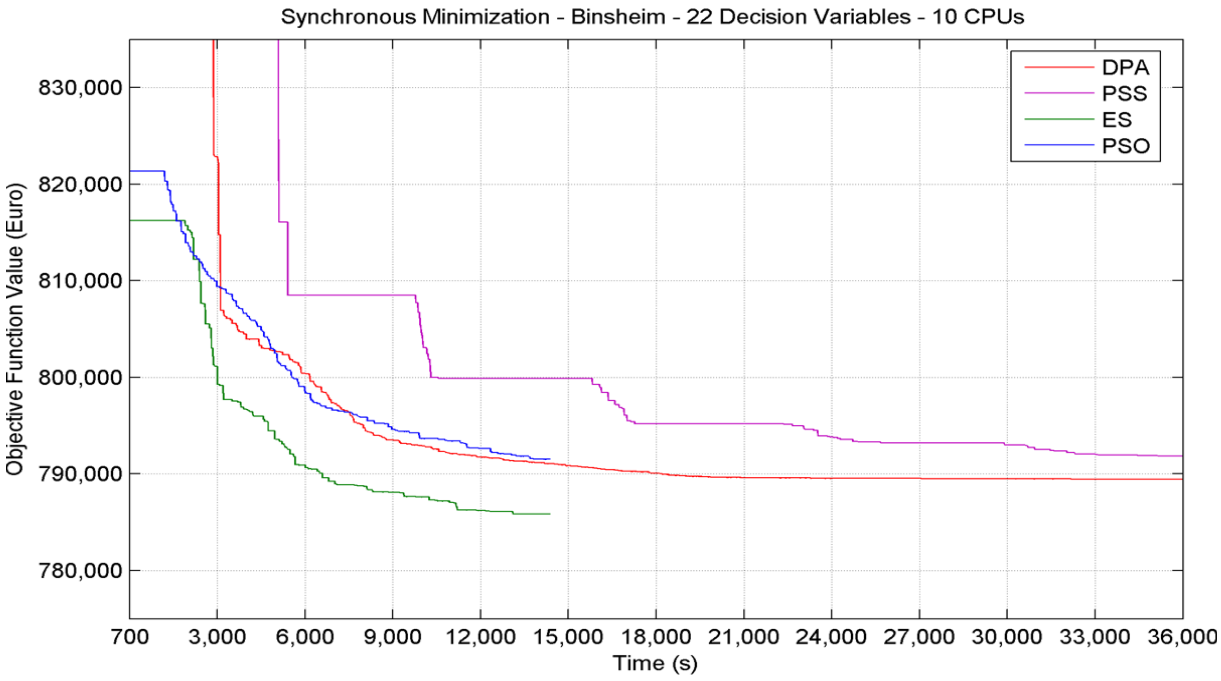

Abbildung 10: Verlauf der Lösungsqualität über die Zeit beim Lösen des 22-dimensionalen wasserwirtschaftlichen Binsheimer-Feld-Problems mit 10 CPUs; Durchschnitt aus jeweils 10 Läufen

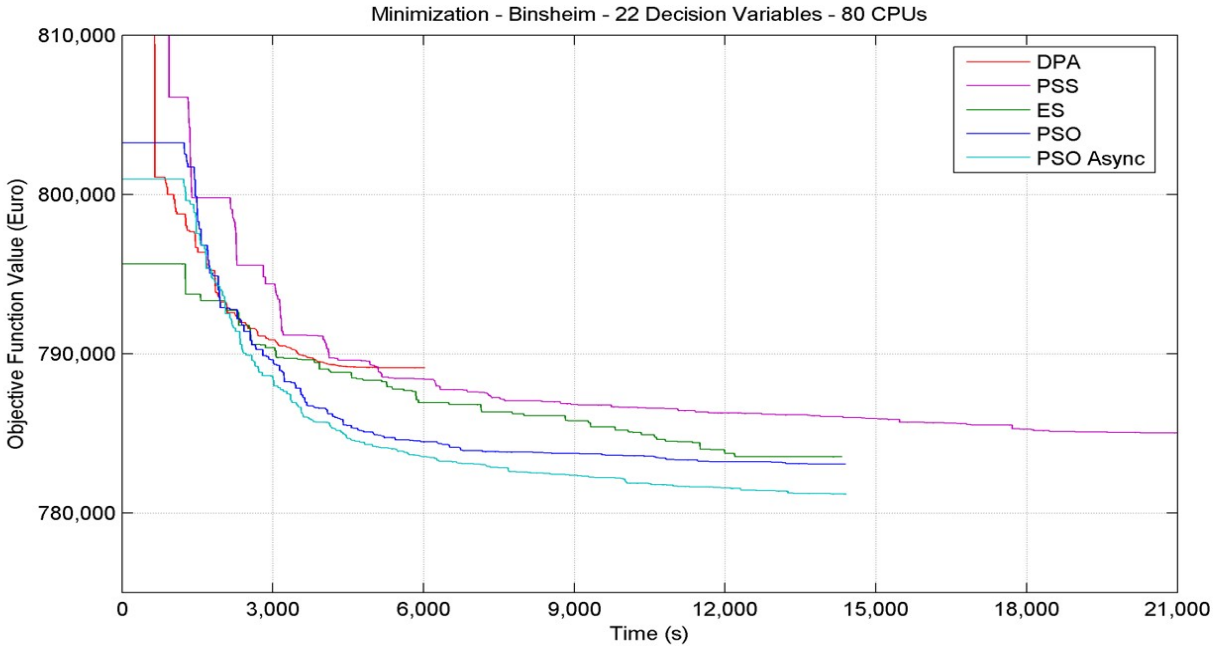

Abbildung 11: Verlauf der Lösungsqualität über die Zeit beim Lösen des 22-dimensionalen wasserwirtschaftlichen Binsheimer-Feld-Problems mit 80 CPUs; Durchschnitt aus jeweils 10 Läufen

5. Entwurf eines Grid-Service

Eine Teilmenge der hier vorgestellten Algorithmen soll in den nächsten Monaten als Komponente in einer serviceorientierten Achitektur zur Verfügung gestellt werden. Konkret ist dazu die Implementierung als WSRF-konformer Grid-Service als Teil eines Grids auf Basis von Globus-Toolkit-4. Da es das Ziel sein muss, die Abhängigkeit der Komponente im System zu minimieren, sollten zunächst die notwendigen, elementaren und generischen Schnittstellen identifiziert werden. Dazu wird als erster Schritt kurz die Einordnung dieses Service in das Gesamtsystem betrachtet.

Der „Optimization Service“ hat die Aufgabe, ein abstraktes Optimierungsproblem zu lösen, das ihm als eines der Klasse „optimaler Entwurf“ präsentiert wird. Dazu wird er initiale Lösungskandidaten erzeugen und diese sukzessiv unter Berücksichtiung des Zielfunktionswerts und der Gültigkeit der Lösung zu verbessern suchen. Die Auswertung, d.h. die Berechnung des Zielfunktionswertes und die Bestimmung der Gültigkeit oder Ungültigkeit, gehört nicht zu seinen Aufgaben, sondern wird an eine andere Komponente deligiert. Der schematische Ablauf einer Optimierung ist in Abbildung 12 als Aktivitäts-diagramm dargestellt. Der Optimierungsservice interagiert hier mit dem sogenannten „Optimization Job Manager“, dessen Aufgabe es ist, die Berechnung der vom Optimierungsservice generierten Lösungskandidaten auf die im Grid verfügbaren Simulationsservices zu verteilen (hier am Beispiel von FEFLOW), die Ergebnisse einzusammeln und dem Optimierungsservice mitzuteilen.

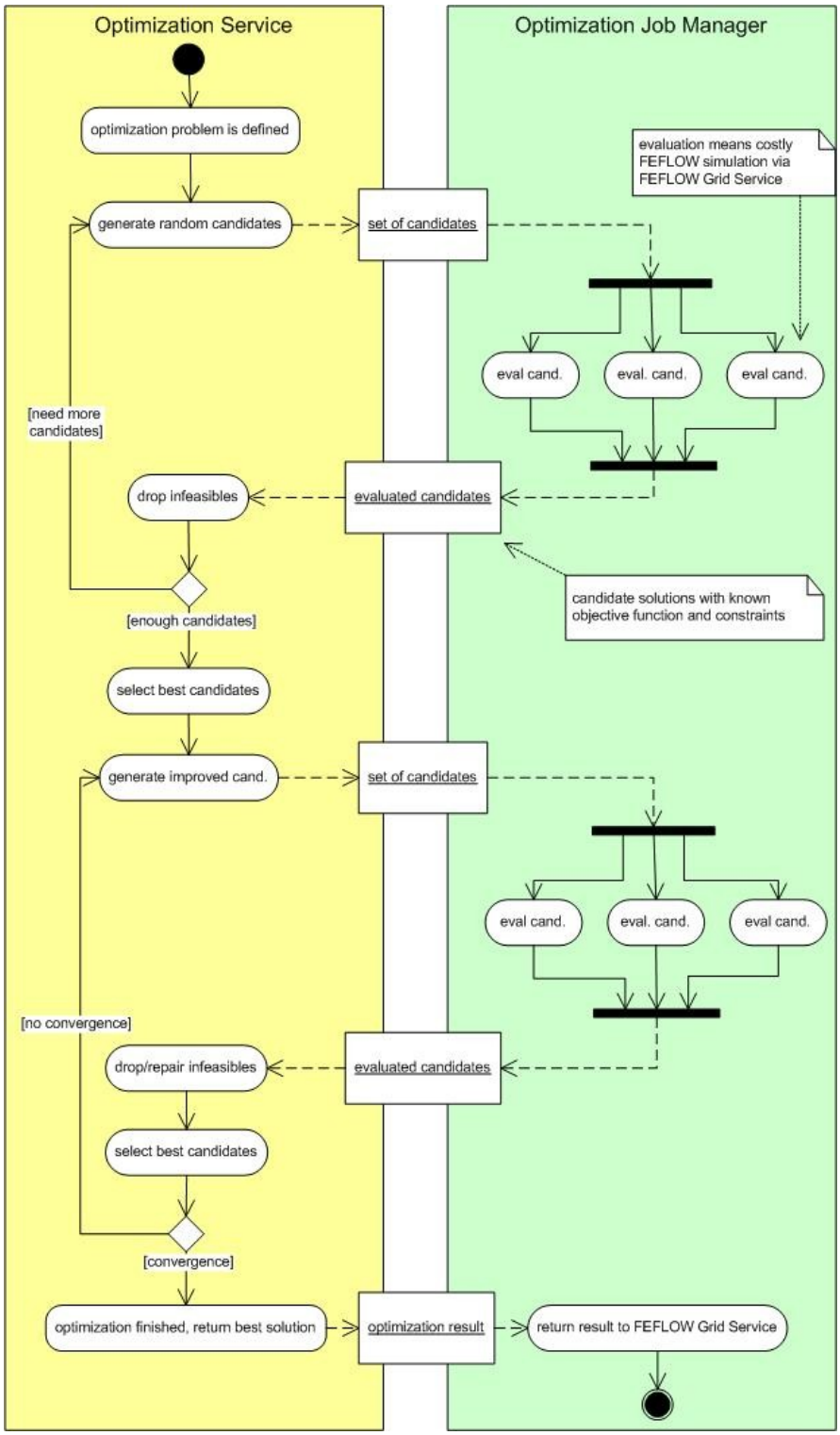

Abbildung 12: Aktivitätsdiagramm für den Ablauf einer verteilte nOptimierung im Grid zwischen dem Optimierungsservice und einem Jobmanager, der für die Auswertung der Lösungskandidaten verantwortlich ist

Der genaue Ablauf im Gesamtsystem muss noch mit den in AP 3.2 entwickelten Workflowtechniken abgestimmt werden. Aufbauend auf dem in Abbildung 12 dargestelltem logischen Ablauf können aber konkretisierend folgende elementare Schnittstellen bereits identifiziert werden. Dabei agiert der Optimierungsservice als reaktiver Service, d.h. er reagiert lediglich auf Serviceaufrufe mit internen Zustandsänderungen und Antworten, ohne selbst aktiv Serviceaufrufe durchzuführen:

Serviceaufruf	Beschreibung	Parameter	Antwort
define problem	Zu Beginn müssen die elementaren Parameter des Problems mitgeteilt werden. Neben der Beschreibung des Problems selbst beinhaltet dies auch den Parallelitätsgrad.	● Anzahl Entscheidungsvariablen ● Unter- und Obergrenzen der Variablen ● Anzahl der Nebenbedingungen (optional) ● max. Parallelitätsgrad	● gewünschter Parallelitätsgrad
get best solution	Abfrage der besten bisher gefunden Lösung	---	● Entscheidungsvariablen ● Zielfunktionswert
new candidates	Erzeugen einer Menge von neuen Lösungskandidaten, die parallel ausgewertet werden können	---	Entweder ● k Kandidaten mit je n Entscheidungsvar. oder ● Optimierung beendet
tell results	Das Ergebnis der Auswertung von Lösungskandidaten wird dem Algorithmus mitgeteilt	k Ergebnisse, jeweils: ● Entscheidungsvariablen ● Zielfunktionswert ● Gültigkeit ● Verletzungsstärke der einzelnen Nebenbedingungen (optional)	---
terminate	Optimierung beenden	---	---

Die Anzahl der Nebenbedingungen sowie die Stärke der Verletzung der einzelnen Nebenbedingungen wurden als optional markiert, da z.Z. keiner der Algorithmen Gebrauch von dieser Information macht. Stattdessen wird lediglich zwischen gültigen und ungültigen Lösungskandidaten unterschieden. Prinzipiell kann diese Information aber verwendet werden, z.B. bei der Reparatur von Lösungen oder zur Anpassung von Schrittweiten o.ä. in der Nähe des gültigen Bereichs.

Die Schnittstellen zur Abfrage neuer Lösungskandidaten und der Mitteilung der Auswertungsresultate sollten zudem um die Möglichekit erweitert werden, einen asynchronen Ablauf zu unterstützen. Dazu würde dann zwar z.B. initial eine Menge von Lösungskandidaten angefordert, aber sobald die erste Simulation beendet ist, würde das Ergebnis dem Optimierungsservice direkt mitgeteilt, der dann unter Umständen

unmittelbar mit einem neuen Lösungskandidaten antworten kann. Dadurch werden Leerlaufzeiten vermieden.

6. Ausblick

Bei den Untersuchungen anhand der Rosenbrocktestfunktion haben sich alle acht verwendeten Algorithmen als prinzipiell für den Einsatzzweck tauglich erwiesen. Die Läufe mit Problemen aus dem Grundwassermanagement unter Einsatz des Simulationspakets FEFLOW bestätigen die Ergebnisse im Wesentlichen. In den nächsten Monaten soll nun eine Auswahl der entwickelten Algorithmen in Form eines Grid-Service in einer Globus-Toolkit-4-basierte Gridinfrastruktur zur Verfügung gestellt werden.

7. Literatur

[1] Thilo, F., AP 3.3, Bericht 3.3.1, Innovative Grid-Entwicklungen für ingenieurtechnische Anwendungen (InGrid), 2006.

[2] Barth, T. ; Freisleben, B. ; Grauer, M. ; Thilo, F., „Distributed Solution of Simulation-Based Optimization Problems on Networks of Workstations“, *Journal on Computer Science and Systems*, pp. 94-105, 2000.

[3] Laguna, M, Marti, R., *Scatter Search - Methodology and Implementations in C,* Kluwer, 2003.

[4] Gray, G.A., Kolda T.G., „APPSPACK 4.0: Asynchronous Parallel Pattern Search for Derivation-Free Optimization“, *Technical Report SAND2004-6391*, Sandia National Laboratories, Livermore, 2004.

[5] Kirkpatrick, S., Gelatt, C.D. und Vecchi, M.P., „Optimization by Simulated Annealing“, *Science*, 220(4598), 671-680, 1983.

[6] Müller, C., „Analysis and Implementation of Nondeterministic, Naturally-Inspired Meta-Heuristics for Simulation-Based Optimization on Parallel Computing“, Diploma Thesis, Siegen, 2007.

[7] Dueck, G., „New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel“, *Journal of Computational Physics*, 104(1), 86-92, 1993.

[8] Ahn, C.W., *Advances in Evolutionary Algorithms: Theory, Design and Practice*, Springer, 2006.

[9] Schwefel, H.-P. und Beyer, H.-G., „Evolution strategies – a coprehensive introduction“, *Natural Computin*g 1(1), 3-52, 2002.

[10] Kennedy, J., Eberhart, R. und Shi, Y., *Swarm Intelligence*, Morgan Kaufmann Publishers, 2001.