Entwicklung eines multi-kriteriellen, hybriden Optimierungs-algorithmus für den Einsatz in der Kanalnetzsteuerung

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Diplomarbeit

Dominik Kerber

16. Februar 2009

Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Algorithmik

Technische Universität Darmstadt
Fachbereich Bauingenieurwesen und Geodäsie
Fachgebiet für Ingenieurhydrologie und Wasserbewirtschaftung
Institut für Wasserbau und Wasserwirtschaft

Betreuender Hochschullehrer: Prof. Dr. Karsten Weihe, TU Darmstadt
Betreuer: Dipl.-Ing. Steffen Heusch, TU Darmstadt
Dr.-Ing. Dirk Muschalla, Université Laval

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 16. Februar 2009

Dominik Kerber

Danksagung

Ich möchte an dieser Stelle allen danken die mich in meiner Studienzeit persönlich, fachlich und wirtschaftlich unterstützt haben.

Ein besonderer Dank gilt meinen Betreuern die eine konstruktive Arbeitsumgebung geschaffen haben in der ich meine Ideen verwirklichen konnte auch wenn sie über die Aufgabenstellungen hinausgingen. Besonders zu erwähnen sind hierbei Felix Fröhlich, Dirk Muschalla, Steffen Heusch und Christoph Hübner.

Meine Eltern die mich all die Jahre unterstützt haben und mir persönlich alle Voraussetzungen geschaffen haben, mich voll auf mein Studium konzentrieren zu können.

Meinen Geschwistern für ihre Motivation in schwierigen Phasen.

Dipl.-Ing. Steffen Heusch, Prof. Dr. Karsten Weihe, Andreas Thiem und Andreas Brett für die hilfreichen Korrekturvorschläge.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

In unserer Gesellschaft existieren zunehmend komplexe Aufgaben deren Berechnung die bestehenden Computersysteme ausreizen und oft sogar überfordern. Dennoch sind solche Aufgaben in Zukunft von zunehmend zentraler Bedeutung und in ihrer Komplexität nicht ohne Informationsverlust zu reduzieren. Man denke an Wetterprognosen, Windkanalberechnungen oder materialsparende Konstruktionen.

All diese rechenintensiven Aufgaben basieren auf Modellen, deren Komplexität in direktem Zusammenhang zur Qualität der Lösung stehen. Es liegen also Rechenzeit und Güte der Lösungen als zwei konkurrierende Ziele vor, deren Gewichtung je nach Anforderung und Situation stark variieren kann.

Konventionell würde für ein solches System zunächst eine Bestimmung der Parametermenge als Abstraktion für die Güte der Lösung durchgeführt werden. Danach würde diese zusammenhängende Menge von Parametern mit Hilfe von großem Rechenaufwand in ein Gradientensystem überführt werden und anschließend schnell gelöst werden können.

Dieses Vorgehen bietet jedoch einige Nachteile: Die Überführung stößt, wegen der erforderlichen Komplexität des Modells, schnell an die Grenzen des Berechenbaren, die Berechnung erfordert detaillierte Gradientenkenntnisse und die Lösungsmenge besteht aus genau einem exakt berechneten Element.

Für viele Anwendungen der heutigen Zeit ist jedoch ein exaktes Ergebnis nicht das eigentliche Ziel - es wird vielmehr nach mehreren Lösungen gesucht deren Auswahl dann durch zeitlich variable Anforderungen oder mathematisch schwer erfassbare Entscheidungskriterien, wie z.B. Design, erfolgen soll. Ein weiteres Problem der konventionellen Herangehensweise ist die starre Umsetzung bezogen auf eventuell später benötigte Freiheitsgrade der Lösung. Je mehr Freiheitsgrade man in das Ergebnis einfließen lässt, desto höher sind die Überführungskosten in ein Gradientensystem.

Eine Alternative für die beschränkten Möglichkeiten dieses starren Vorgehens stellt hierbei die Optimierung mit evolutionäre Algorithmen dar, da ihre Herangehensweise und der resultierende Lösungstyp sehr viel besser den Zielsetzungen der heutigen Problemstellungen entsprechen. Zum Einen ist kein Umsetzen in ein Gradientensystem notwendig, was eine enorme Flexibilität auf der Parameterseite ermöglicht, und zum Anderen steigt die Güte der Lösung im Allgemeinen mit der Rechenzeit und ist somit sehr flexibel kalkulierbar. Allerdings stellt diese Art der Optimierungsalgorithmen kein Allheilmittel für die Probleme der technisierten Gesellschaft dar, da solche Algorithmen keine Garantie für das Finden der besten Lösung geben können und die Rechenzeit im Vergleich zu einem vorliegenden Gradientensystem ungleich höher ist. Dennoch kann mit diesen strukturell bedingten Problemen wesentlich besser umgegangen werden, da sie keine harten Machbarkeitsgrenzen darstellen und noch viel Raum für Entwicklungen lassen.

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist es, einen hybriden, evolutionären Algorithmus zu entwickeln, der aufgrund seiner Konvergenzgeschwindigkeit zur direkten Kanalnetzsteuerung genutzt werden kann. Es bleibt zu klären, in welcher Ausprägung eine hybride Form des Algorithmus eine Erhöhung der Konvergenzgeschwindigkeit bewirken kann.

Bei der Nutzung einer Optimierungsstrategie ist von enormem Vorteil, dass ein Kanalnetz nicht in ein Gradientensystem überführt werden muss und so keinerlei formale und zusammenhängende Definitionen der zugrunde liegenden Struktur erforderlich sind um sinnvolle Ergebnisse zu produzieren.

Die Aufgabenstellung fordert allerdings auch, dass ein solches System eine Optimierung innerhalb weniger Minuten ermöglicht. Die Planung mit dem Fokus auf Rechenzeiten wird also einen wichtigen Punkt darstellen und sollte in alle strukturellen Entscheidungen einfließen. Um den Algorithmus anschließend umzusetzen, muss eine vorhandene Softwareplattform teilweise modular restrukturiert und erweitert werden um den Algorithmus an mehrere Simulationsmodelle koppeln zu können. Des Weiteren muss die Optimierungsfunktionalität so erweitert werden, dass mehrere Rechner gemeinsam und in einer dynamischen Struktur ein Problem bearbeiten können.
Konkret fordert die Aufgabenstellung:

1. Auswahl geeigneter Algorithmen, insbesondere im Hinblick auf die Zielsetzung online-Optimierung von Kanalnetzen
2. Entwicklung eines hybriden Algorithmus, der die ausgewählten Algorithmen verknüpft
3. Parallelisierung des entwickelten hybriden Algorithmus
4. Definition eines Benchmarksystems zur Echtzeitsteuerung (mit maßgeblicher Unterstützung des IHWB und modelEAU
5. Optimierung des Benchmark-Beispiels mit dem globalen und dem hybriden Algorithmus
6. Vergleich und Diskussion der beiden Ansätze

1.2 Struktur der Arbeit

Kapitel 2 wird zunächst eine Übersicht über den momentanen Stand der Technik im Bereich evolutionäre Algorithmen und deren Parallelisierung geben sowie die Arbeit in den Kontext der momentanen Entwicklungen einordnen.

Kapitel 3 beschäftigt sich mit den Grundlagen evolutionärer Algorithmen im Vergleich zu Gradientenverfahren und deren Vorteile für eine Simulationsberechnung. Dabei werden

die sehr verschiedenen Ansatzpunkte evolutionärer Algorithmen aufgezeigt und deren mögliche Auswirkungen auf die Entwicklung der Lösung dargestellt.

Kapitel 4 stellt die geplante Struktur des Programms dar und geht auf die einzelnen Funktionen ein. Die Grenzen der Techniken und verbleibende Freiheitsgrade werden hier ebenfalls dargestellt. Des Weiteren werden einige kritische Komponenten genauer untersucht und Schlussfolgerungen für die Umsetzung gezogen. Im Rahmen dieses Kapitels werden die Algorithmen im Sinne von Punkt 1 und 2 der Aufgabenstellung ausgewählt bzw. eine theoretische Automatisierung für den detaillierten Auswahlmechanismus entwickelt. Ebenfalls in diesem Kapitel wird Punkt 3 behandelt und eine Netzwerklösung, ebenfalls zunächst theoretisch, entwickelt.

Kapitel 5 gibt einen Einblick in die verwendeten Programmierparadigmen und deren Umsetzung. Eine Funktionsbeschreibung aller Klassen, insbesondere für die im Rahmen der Aufgabenstellung 1 bis 3 entwickelten Algorithmen, ist ebenso Teil der Lösung wie einige Hinweise auf Besonderheiten in diesem Anwendungsbereich.

Kapitel 6 bewertet die gewonnenen Ergebnisse. Dabei wird insbesondere auf die geforderte Nutzung als direkte Kanalsteuerung nach dem 4. Punkt der Aufgabenstellung eingegangen sowie weitere Aufgaben und deren Lösung, wie in Aufgabenstellung 5 und 6 gefordert, beispielhaft vorgestellt und untersucht.

Kapitel 7 beschäftigt sich mit den Erweiterungsmöglichkeiten und Grenzen der geschaffenen, modularen Struktur sowie mit weiteren Forschungs- und Entwicklungsansätzen.

2 Stand der Technik

Evolutionäre Algorithmen erlangen im Zusammenhang mit den heutigen, komplexer werdenden Problemstellungen eine immer höhere Bedeutung. Wurden anfangs solche Algorithmen eingesetzt um statische Probleme wie z.B. die Platzierung von Mobilfunkmasten zu optimieren, spielt heute ein weiterer Vorteil, nämlich die Variabilität der Eingabedaten, eine sehr wichtige Rolle. Die Bearbeitung solcher statischer Problemstellungen, wie z.B. Design von Werkstücken mit fest definierten Anforderungen, ist bereits als erfolgreich arbeitende Methode einer Kombination von evolutionären Algorithmen durch Kalyanmoy Deb und Tushar Goel [Kalyanmoy Deb, 2001] und vielen anderen nachgewiesen worden. Es wurde schon damals deutlich, dass die evolutionären Algorithmen einen vielversprechenden Ansatz bieten konnten und trotz ihres frühen Entwicklungsstands bereits erstaunliche Ergebnisse erzeugten.

Ein weiteres Forschungsgebiet stellen die Simulationsmodelle dar. An diese wird die Anforderung gestellt, die Wirklichkeit in ausreichender Genauigkeit bei geringen Auswertungszeiten darstellen zu können. Wie in [Muschalla, 2006] deutlich wird, sind Simulationsmodelle im Allgemeinen rudimentär umgesetzt und ermöglichen meist nur die Berücksichtigung einiger weniger Faktoren. Die bisherigen Modelle haben aufgrund der hohen Rechenanforderungen bzw. der nicht einfach zu bestimmenden Art und Anzahl der Einflussfaktoren oft nur den Fokus auf einige wenige resultierende Größen gelegt. Diese Spezialanwendungen stellen allerdings eine durchaus sinnvolle Grundlage dar, Vorhersagen zu treffen und die Planungssicherheit zu erhöhen ohne tatsächliche Bestätigung durch lange Messreihen zu benötigen. Im Hinblick auf die Vorhersagen im Zusammenhang mit Kanalnetzen und Gewässern existieren mehrere Modelle die sich meist im Hinblick auf die resultierenden Aussagen unterscheiden und dementsprechend verschiedene Eingabedaten benötigen. Zu diesen zählen beispielsweise REBEKA I [Rauch, 2000] und REBEKA II [Frankhauser, 2004], SYNOPSIS [Schütze, 1998] und WEST [Meirlaen, 2002a] [Meirlaen, 2002b] sowie die in dieser Diplomarbeit angesteuerten Modelle SMUSI [Ostrowski, 1998] und BlueM. Mit in Zukunft steigender Rechenleistung sowie höherer Effektivität und Qualität der Eingabedaten wird sich allerdings die Anzahl der Faktoren kontinuierlich erhöhen lassen und somit letztendlich die Aussagekraft der Lösungen steigern lassen.

Die Verbindung dieser beiden Forschungsgebiete wurde bereits von Rauch und Harremoes [WOLFGANG RAUCH, 1999] zur Sprache gebracht und stellte seitdem eine Lösung für neue Anwendungsgebiete, wie z.B. die direkte selbstständige Steuerung von Kanalnetzen, in Aussicht. Auch weitere Umsetzungen wie eine automatische Luftwiderstandsoptimierung für Fahrzeuge oder aktive Strukturerhaltende Maßnahmen für Bauwerke bei Erdbeben wären denkbar. Es liegt auf der Hand, dass eine Fülle von Anwendungen, basierend auf dieser Art der Kopplung von Simulationsmodellen und evolutionären Optimierungen möglich sind.

Bisher ist der Einsatz solcher Steuerungsmethoden allerdings noch selten zu finden und liegt zum Teil auch an den wenig standardisierten Modellen die die Wirklichkeit natürlich nur beschränkt und in begrenzten Gebieten wiedergeben können. Zum Anderen könnte man einen Grund darin sehen, dass die in der aktiven Steuerung aufgrund der Rechenzeit erforderlichen Modelle oder produzierten Lösungen noch wesentlich gröber sein müssen und daher mit steigendem Umfang keine exakte Nutzung mehr zulassen. In vielen komplexen Problemstellungen ist es allerdings unerheblich ob die gefundenen Lösungen die Besten oder 'nur', durch ein vereinfachtes Modell, sehr gut sind. Entscheidend ist, dass komplexe Daten effizient zur Verbesserung herangezogen werden können und eine Lösungsvielfalt produziert wird, aus der nach weiteren Kriterien die in der Situation beste Konstellation auszuwählen ist.

Es zeigt sich insgesamt, dass evolutionäre Algorithmen heute ein weites Forschungsfeld darstellen in welchem es viele Ziele gibt. Die Ansätze sind sehr verschieden, haben aber dennoch, durch Vorteile in spezifischen Anwendungen, jeweils ihre Daseinsberechtigung. Die Entwicklung von spezialisierten Anwendungen ist daher ein weiterer Schritt um diese Vorgehensweise in die Praxis zu überführen und die evolutionären Algorithmen im Allgemeinen als probates Mittel für komplexe Optimierungs- und Steuerungsaufgaben zu etablieren.

3 Grundlagen

In diesem Kapitel gilt es zunächst eine Übersicht über Optimierungsalgorithmen im Allgemeinen, sowie Gradientenverfahren und evolutionäre Algorithmen im Speziellen zu schaffen. Dabei wird die verschiedenartige Struktur beleuchtet und die grundsätzlichen Überlegungen aufgezeigt, aus Basisalgorithmen einen effizienten Berechnungsalgorithmus aufzubauen.

Gradientenverfahren sind dahingehend beschränkt, dass die Anzahl der generierten Lösungen für multikriterielle Aufgabenfelder oft nicht ausreicht ein Problem zufriedenstellend zu lösen oder gar die passende Gewichtung von einzelnen, konkurrierenden Teilzielen vorzunehmen. Jedoch ist eine multikriterielle Problemstellung die informativ am nächsten liegende und am wenigsten reduzierte Art eines Optimierungsansatzes. Vergleicht man solche Problemstellungen mit der Natur, wird schnell klar, dass die Natur in allen Bereichen aus nahezu idealen Strukturen besteht die im Hinblick auf mehrere Zielsetzungen optimiert sind. Es ist naheliegend, die erfolgreiche Optimierungsstrategie der Evolution als Grundlage der mathematischen Berechnung in Betracht zu ziehen und ein System zu schaffen, welches mehrere Optimierungsziele verfolgen kann und dementsprechend auch mehrere Lösungen zur Auswahl stellt falls diese existieren.

Bereits in den sechziger Jahren begannen verschiedene Arbeitsgruppen unabhängig voneinander die erfolgreichen Regeln der Evolution in eine Algorithmenklasse zu überführen um nach dem Vorbild der Natur eine neue Sicht auf Optimierungsprobleme zu entwickeln. Es zeigte sich schnell, dass das Potential bei multikriteriellen Problemstellungen enorm war. Jedoch auch, dass die nötige Spezialisierung einzelner evolutionärer Strategien keinen allgemeinen Durchbruch bringen konnte.

Evolutionäre Algorithmen als spezielle Grundlage von Optimierungsverfahren bieten die Möglichkeit, auch ohne detaillierte Kenntnis des Lösungsraums, mehrere unterschiedliche, aber ähnlich gute Lösungen zu finden. Die Algorithmen werden außerdem nicht durch Diskontinuitäten in der Problemstellung unbrauchbar und können mit weit höheren Komplexitäten umgehen als die algorithmischen Strukturen der Gradientenverfahren. Diesen klaren Vorteilen steht gegenüber, dass der Rechenaufwand bei multikriteriellen Optimierungsalgorithmen im Allgemeinen sehr hoch ist. Ein akzeptables Mittel die hohe Rechenzeit in den Griff zu bekommen stellt neben der eigentlichen Optimierung des Algorithmus selbst auf die spezifische Problemstellung die Parallelisierung des Berechnungsvorgangs auf mehreren Recheneinheiten dar.

3.1 Begriffe zu evolutionären Algorithmen

Im Umfeld der evolutionären Algorithmen sind folgende Begriffe gebräuchlich und werden daher in dieser Arbeit genutzt (siehe auch [Papageorgiou & von Stryk, 2009]):

Optimierungsparameter:
Stellen die Eingabewerte für Zielfunktion dar und sind die letztendlich durch die Optimierung gesuchten Werte.

Zielfunktion:
Berechnet jeweils aus einer Teilmenge der Optimierungsparameter eine Eigenschaft. Sind mehrere Zielfunktionen vorhanden die sich gegenseitig nachteilig beeinflussen, spricht man von einer multikriteriellen Problemstellung.

Individuum:
Ein Individuum besteht aus einem Satz von Optimierungsparametern und den aus den Zielfunktionen resultierenden Eigenschaften.

Eltern:
Individuen, die nach Bewertung der Ergebnisse der Zielfunktionen als Basis für die neue Generation dienen.

Generation:
Besteht aus einer Menge von Individuen die zum gleichen Zeitpunkt existieren.

Population:
Besteht aus einer zeitlichen Abfolge von Generationen.

Parameterraum:
Definiert die Grenzen der einzelnen Optimierungsparameter.

Lösungsraum:
Beinhaltet die Ergebnisse der Zielfunktionen.

Minima:
Minima sind im Lösungsraum lokal oder global niedrigste Werte **einer** Zielfunktion. Man spricht daher auch von globalen oder lokalen Minima. Ein Minimum wird in der Praxis oft als bestes Ergebnis definiert da der Lösungsraum oft als Abweichung vom perfekten aber nicht erreichbaren Idealverhalten definiert wird. Das Ziel ist also die Minimierung dieser Fehlerwerte.

3.2 Gradientenverfahren

Um die Extremstellen einer Funktion zu finden, wird im Allgemeinen die erste Ableitung gebildet und nach den Nullpunkten gesucht. Ist die Funktion zu komplex, nicht ableitbar oder einfach nicht bekannt, helfen nur Optimierungsverfahren die besten Parameter zu finden. Diese Verfahren sind die interessantesten was die Anwendung in komplexen Pro-

blemstellungen und speziell im Zusammenhang mit evolutionären Algorithmen angeht, denn das nicht-bekannt-sein der zugrunde liegenden Funktionen stellt den Normalfall in der Natur und bei reellen Optimierungsproblemen dar.

Optimierungsverfahren nutzen die Ergebnisse der letzten Berechnung um eine Bewertung der genutzten Eingabeparameter durchzuführen und diese so im Laufe der Zeit zu optimieren. Der generelle Nachteil an diesem Vorgehen ist, dass die fehlenden Informationen über die zugrunde liegende Funktion durch eine ungleich höhere Anzahl an Iterationen kompensiert werden muss. Ein großer Vorteil ist jedoch die robuste Konvergenz bei stetigen Funktionsabschnitten und damit das garantierte Auffinden eines eventuellen Minimums im untersuchten Bereich. Die Einschränkung auf einen bestimmten Bereich resultiert aus der Struktur der Gradientenverfahren, die nicht versuchen die Funktionsvorschrift zu rekonstruieren, sondern möglichst schnell dem Minimum nahe zu kommen. Das Resultat ist eine relativ geringe Rechenzeit aber auch das Risiko, bei unstetigen Funktionen in eine Endlosschleife zu geraten oder nur ein lokales Minimum zu finden.

Neben den einfachen und meist bekannten Verfahren [Papageorgiou & von Stryk, 2009] [1] wie Bisektion [Wikipedia, 2009c] und Sekantenverfahren [Wikipedia, 2009d] gibt es die spezielleren Hillclimbing-Suchverfahren [Wikipedia, 2009a] die sehr gut an multikriterielle Problemstellungen angepasst werden können. Als Beispiel zur Arbeitsweise einer skalaren Optimierung dient das Downhill-Simplex-Verfahren [Wikipedia, 2009b] [Nelder & Mead, 1965]:

Das Verfahren selbst ähnelt in gewisser Weise einem einfachen, evolutionären Algorithmus was die Verwendung der in Abschnitt 3.1. definierten Begriffe naheliegend macht. Das Downhill-Simplex-Verfahren nutzt die Funktionswerte um die Optimierungsparameter zu bewerten. Um einen Fortschritt zu erzielen werden immer mehrere Individuen (Punkte) gleichzeitig vorgehalten und das Individuum mit den geringsten Funktionswerten aussortiert.

Im folgenden schematischen Beispiel für das grundlegende Vorgehen bei zwei Optimierungsparametern und einer Zielfunktion werden die Parameter als Koordinaten in einem euklidischen Koordinatensystem eingetragen und der Funktionswert als Höhe in dieser Koordinate repräsentiert. Ziel ist das Finden des Minimums (tiefsten Tals) in der Umgebung:

Start:
Die zufällige Wahl von 4 Individuen stellt den ersten Schritt dar und bestimmt durch den Abstand der Optimierungsparameter und die Entfernung zum Minimum die Laufzeit des Algorithmus maßgeblich. Direkt darauf folgt die Simulation der Individuen.

Vergleich:
Durch den Vergleich der 4 Individuen im Lösungsraum wird bestimmt, welche der 4 möglichen Maßnahmen zur Generierung eines neuen Individuums durchgeführt wird.

[1] für ergänzende, animierte Darstellung der Algorithmen wird nachfolgend auf Wikipedia verwiesen

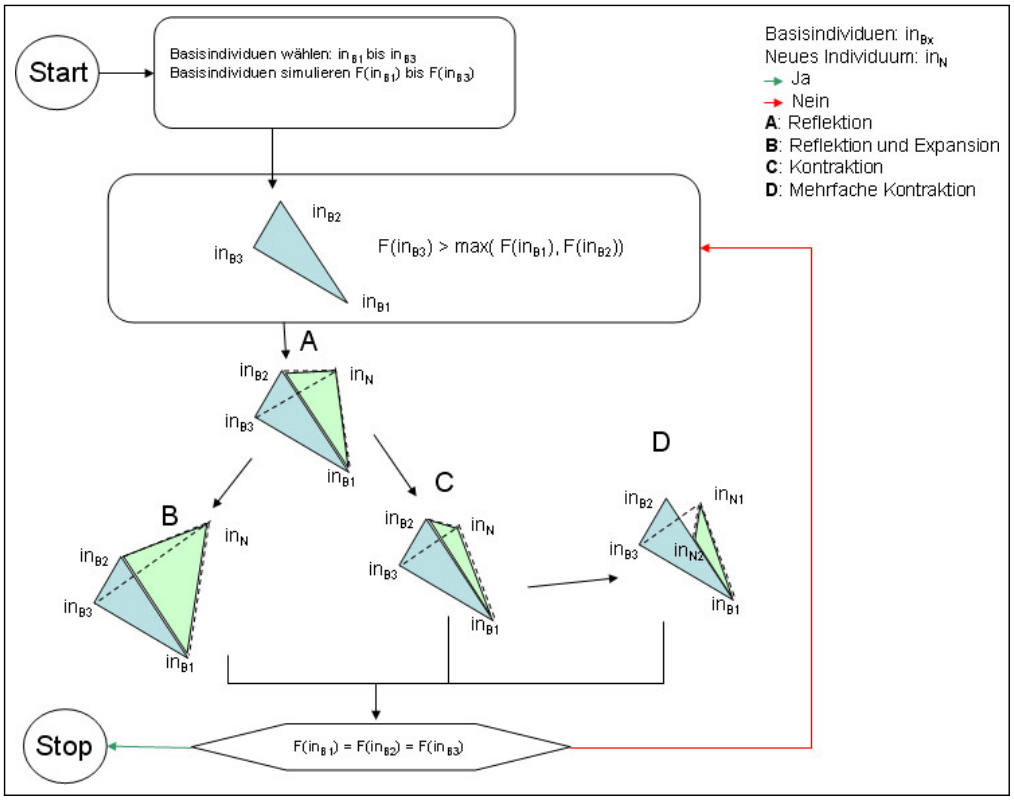

Abbildung 3.1 – Alg.: Downhill-Simplex-Algorithmus

A:
Es liegt vor: $F(in_{B4}) > F(in_{B3}) > max(F(in_{B2}), F(in_{B1}))$
→ Reflektion von in_{B3} auf in_N

B:
Falls für das in A neu gefundene Individuum in_N gilt:
$min(F(in_{B2}), F(in_{B1})) > F(in_N)$
→ Expansion von in_N

C:
Falls für das in A neu gefundene Individuum in_N gilt:
$max(F(in_{B2}), F(in_{B1})) > F(in_N) > min(F(in_{B2}), F(in_{B1}))$
→ Kontraktion von in_N

D:
Falls für das in C neu gefundene Individuum inimmer noch gilt:
$max(F(in_{B2}), F(in_{B1})) > F(in_N) > min(F(in_{B2}), F(in_{B1}))$

→ Mehrfache Kontraktion um den tiefsten Punkt

Abbruchkriterium erreicht:
Falls die 3 Individuen auf demselben Punkt liegen oder einen vorher definierten Abstand unterschreiten

Wie bei jedem Gradientenverfahren besteht immer die Gefahr, sich in lokalen Minima festzusetzen. Eine effiziente Methode dies zu vermeiden ist bei diesen Verfahren nicht möglich. Um solchen Fällen dennoch vorzubeugen, werden auch in anderen Optimierungsstrategien oft zwei Methoden angewandt:

- Gelegentliche Erhöhung des Faktors für den Richtungsvektor (hier: zufällige Expansion)
- Mehrmaliges Wiederholen des Verfahrens mit anderen Startwerten

Beide Methoden kann man lediglich als Hilfsmethoden ansehen um die strukturellen Schwächen der Algorithmen auszugleichen, da sie lediglich mit einer Erhöhung der Iterationen versuchen ein lokales Minimum zu überwinden. Das globale Minimum wird also nicht direkt entdeckt sondern mit längerer Rechenzeit die Wahrscheinlichkeit erhöht, ein lokales Minimum als solches zu umgehen.

3.3 Evolutionäre Optimierungsverfahren

Evolutionäre Optimierungsverfahren im Allgemeinen stellen als Untergruppe der Optimierungsverfahren die Grundlage des hier beschriebenen Ansatzes dar. Allen Varianten dieser Verfahren ist gemein, dass sie Zielfunktionen nutzen um eine Bewertung der gerade vorliegenden Optimierungsparameter zu generieren und auf Basis dieser Bewertung die Parameterwerte mit Hilfe der evolutionären Algorithmen zu modifizieren. Dabei kommen für die Modifikation nur drei einfachen Klassen von Basisschritten vor, die der biologischen Evolution nachempfunden sind:

Mutation:
Die Mutation ist ein Vorgehen, welches einen oder mehrere Parameterwerte mehr oder minder zufällig verändert.

Rekombination:
Die Rekombination benötigt mindestens zwei Individuen um ein neues Individuum durch zufälliges oder gesteuertes Rekombinieren der ursprünglichen Parametersätze zu erzeugen.

Selektion:
Die Selektion wertet nun die Ergebnisse der Bewertungsfunktionen aus um zu entscheiden, welche Parametersätze zukünftig weiter verwendet werden.

Man erkennt leicht, dass durch iteratives Ausführen dieser drei Schritte eine Optimierung der Parametersätze im Sinne der Zielfunktionen stattfinden kann. Da das Ziel der Rekombination und der Mutation grundsätzlich ähnlich ist, nämlich vielversprechende neue Individuen zu erzeugen, wird im Folgenden zur vereinfachten Darstellung ein Beispiel vorgestellt, in dem lediglich die Mutation zum Einsatz kommt.

Bsp: Aufgabenstellung:

Als Beispiel zur Funktionsweise dient eine Problemstellung, in der es darum geht, das Produkt zweier Variablen zu maximieren, wobei gilt: $0 \leq X,Y \leq 20$. Eine Schrittweite und diese beiden Variablen sind Optimierungsparameter eines Individuums und werden in ein Koordinatensystem eingetragen.

Bsp: 1. Initialisierung:

Optimierungsparameter:
Koordinate X, Koordinate Y,
Schrittweite d

Zielfunktion:
X*Y

Randbedingungen:
$0 \leq X,Y \leq 20$

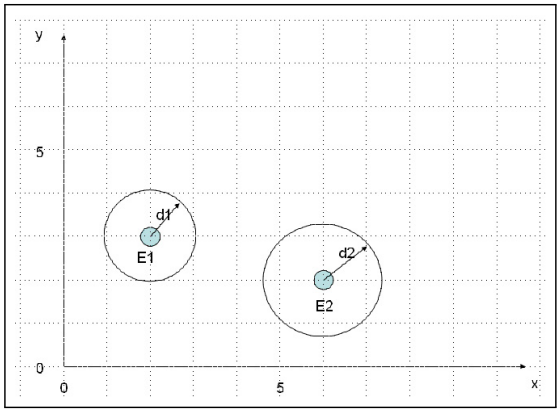

Abbildung 3.2 – Bsp. EA: Initialisierung

Individuum	X	Y	d	Eigenschaft
E1	2	3	1,1	6
E2	6	2	1,3	12

Tabelle 3.1 – Bsp. EA: Initialisierung

Bsp: 2. Mutation:

Bei der Mutation werden in Schrittweite des Individuums ($\pm 0{,}1$) zufällig drei neue Individuen platziert. Die neuen Individuen besitzen nun als Eigenschaften ihre Koordinaten und die Schrittweite mit der sie gesetzt wurden.

Individuum	X	Y	d	Eigenschaft
E1	2	3	1,1	6
E2	6	2	1,3	12
E3	1,2	4,1	1,2	4,92
E4	1,3	2,2	1,1	2,86
E5	2,8	2,8	1	7,84
E6	4,7	2	1,3	9,4
E7	6	3	1,2	18
E8	7,2	0,9	1,4	6,48

Tabelle 3.2 – Bsp. EA: Erste Generation

Aus der nun für jedes Individuum errechneten Eigenschaft werden durch die Selektion die beiden Besten ausgewählt (hier wären es E2 und E7).

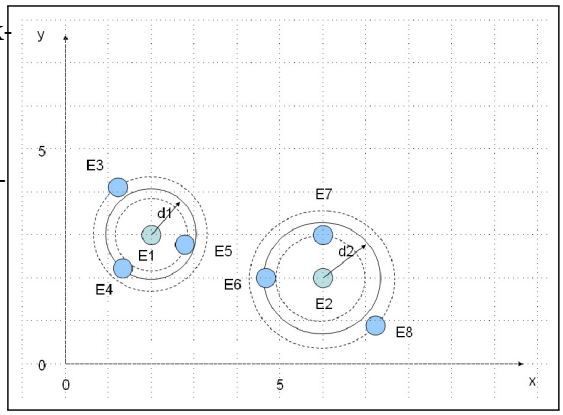

Abbildung 3.3 – Bsp. EA: Erste Generation

Der selbstadaptierende Parameter ist hierbei die Schrittweite: Sie wird sich so anpassen, dass bei einer großen Entfernung vom Optimum die Schrittweite groß werden wird und so die Entfernung schneller überbrückt werden kann. Sobald sich die Individuen dem Optimum annähern, wird sich die Schrittweite, durch die Einwirkung der Randbedingung, wieder verringern. Auf diese Weise kann sich dem Optimum kontrollierter genähert werden. Eine mögliche Annäherung eines Teils der Population könnte demnach wie folgt aussehen:

Hierbei werden die Grenzen der evolutionären Optimierungsverfahren deutlich: Je näher man sich dem Optimum annähert, desto geringer werden die Fortschritte innerhalb einer Generation. Aufgrund der Nähe, der durch die Randbedingung aufgestellten Gültigkeitsgrenze der Optimierungsparameter, steigt die Anzahl der neuen Individuen die wegen der Verletzung eben dieser Randbedingung nicht weiter verwendet werden. Dies bringt zweierlei negative Auswirkungen mit sich: Zum Einen zählen diese Individuen nicht zur Lösungsmenge, obwohl sie nach der Optimierungsfunktion möglicherweise sehr gut sind, zum Anderen werden sie nicht zur Rekombination oder Mutation herangezogen und schmälern somit die Bandbreite der zur Erzeugung der neuen Generation herangezogenen Individuen.

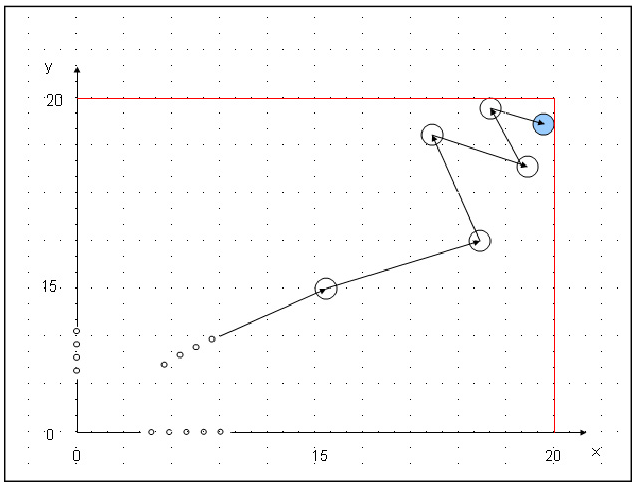

Abbildung 3.4 – Bsp. EA: Annäherung an das Optimum

3.4 Multikriterielle Problemstellung, Paretofront und Diversität

Im letzten Abschnitt wurde ein Beispiel vorgestellt, welches die Optimierung der Population basierend auf einer Zielfunktion und zwei Randbedingungen demonstrierte. In der Praxis ist es allerdings üblich, mehrere konkurrierende Zielfunktionen beachten zu müssen. Das Problem welches sich nun stellt ist von zentraler Bedeutung: Welche Individuen sind die 'besten' und werden bei der Verwendung mehrerer Zielfunktionen als Basis für die neue Generation herangezogen?

Klassisch würde man die beiden Zielfunktionen skaliert gewichten und so zu einer neuen Zielfunktion zusammenfassen. Mit diesem Vorgehen erhält man im Idealfall nacheinander eine Menge an Lösungen, die der Menge an verschiedenen Gewichtungen für die Zielfunktionen entspricht.

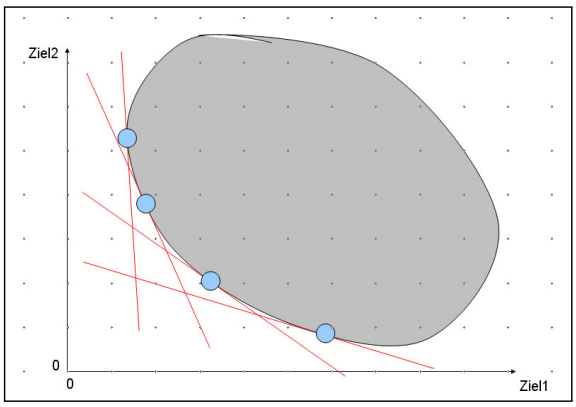

Abbildung 3.5 – Konventionell gefundene Lösungen einer konkaven Paretofront

Man erhält also folgende Lösung:

Auf dem Koordinatensystem sind die beiden konkurrierenden Zielfunktionen aufgetragen. Wie in der Praxis üblich, stellen diese Zielfunktionen Fehler dar, wobei eine Minimierung der Fehler das angestrebte Ziel darstellt. Wenn der graue Bereich die Menge der bisher erzeugten Individuen der Population darstellt, sind die blauen Individuen eine Auswahl der Individuen mit den geringsten Fehlerwerten. Jede rote Tangente symbolisiert dabei die Platzierung der besten und gleich guten Lösungen für eine Skalierung der Zielfunktionen. Ist die Menge der Lösungen in dieser Art

anzutreffen (Abbildung 3.5), ist es leicht, durch Variation der Gewichtung alle optimalen Individuen der bisherigen Population zu finden.

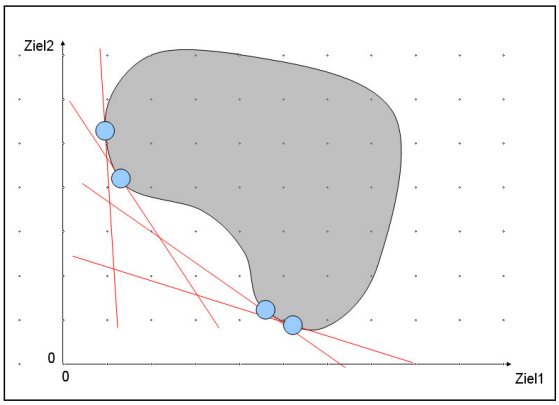

Abbildung 3.6 – Konventionell gefundene Lösungen einer konvex-konkaven Paretofront

Aus diesem Vorgehen kann allerdings eine Einschränkung der Lösungsvielfalt resultieren:

Liegt die Menge der Lösungen wie in diesem Beispiel vor (Abbildung 3.6), erkennt man leicht, dass die Methode der festen Gewichtungen an ihre Grenze stößt. Es ist nicht möglich die Lösungen im konkaven Bereich der Lösungsmenge zu finden obwohl diese bei einer gleichmäßigen Gewichtung beider Ziele durchaus als gewünschte Lösung in Betracht kommen.

Durch die Anwendung des Dominanzkriteriums kann man dieses Problem beseitigen:

Lösung X dominiert Lösung Y wenn:

- Lösung x ist nicht schlechter als Lösung y für alle Zielfunktionen
- Lösung x ist besser als Lösung y für mindestens eine Zielfunktion

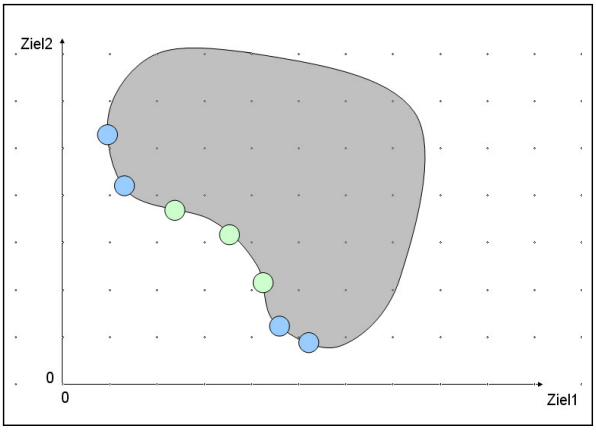

Abbildung 3.7 – Lösungen durch Anwendung des Dominanzkriteriums

Die so gefundenen, 'gleichguten' Individuen bilden die Paretofront (Abbildung 3.7) und stellen eine optimale Ausgangsbasis zur Erzeugung der nächsten Generation dar. Die nach abgeschlossener Optimierung (im Rahmen der Modellgenauigkeit) gefundene Paretofront wird hier als Paretofront der optimalen Lösungen bezeichnet und stellt die Simulationsergebnisse der besten Optimierungsparameter dar.

Diese Art der Klassifikation kann auch für die übrigen, von der Paretofront bereits dominierten Individuen erneut durchgeführt werden um so sukzessive eine klare Rangfolge zu erzeugen. Als Ergebnis wird jedem Individuum ein so genannter Pareto-Rang zugewiesen.

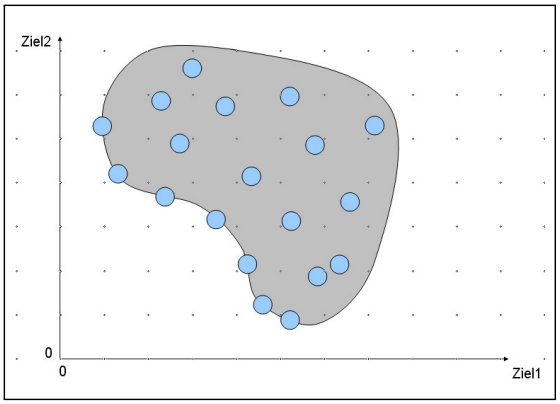

Abbildung 3.8 – Lösungen im Verlauf der Berechnung

Der Pareto-Rang ermöglicht es, eine genauere Analyse des Ist-Zustands zu erzeugen und gezielt den Rang zu nutzen um Optimierungsparameter zu beeinflussen.

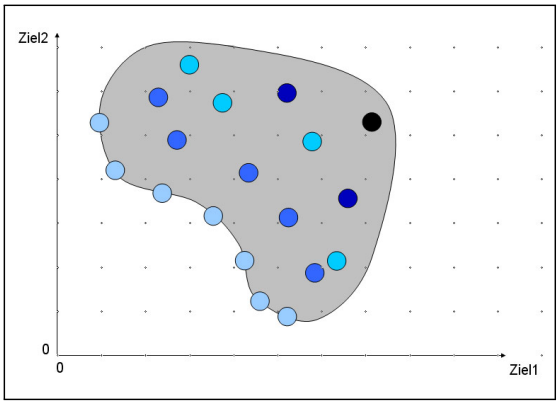

Abbildung 3.9 – Lösungen nach Pareto-Rängen

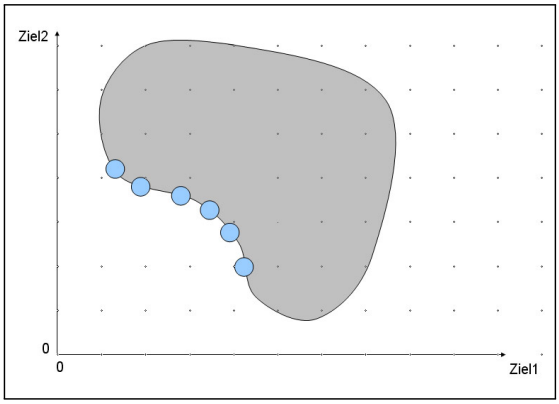

Abbildung 3.10 – Geringe Diversität

Die Diversität einer Paretofront ist eine wichtige Eigenschaft um die Vielfalt der Lösungen zu klassifizieren. Gemeint ist die Breite der aktuellen Paretofront und sagt aus, dass die Lösungen in Bezug auf die Zielfunktionenswerte sehr verschieden ausfallen.

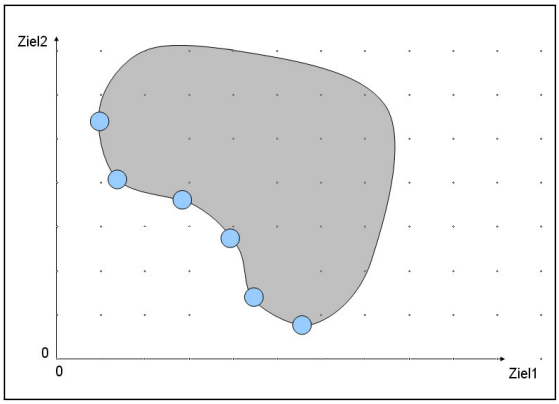

Abbildung 3.11 – Hohe Diversität

Eine hohe Diversität ist bereits zur Laufzeit eines Algorithmus erstrebenswert, da das Finden aller Minima auf diese Weise zuverlässiger und schneller funktioniert. Genauso ist die Wahrscheinlichkeit geringer, ausschließlich ein lokales Minimum zu treffen.

3.5 Ausprägungen evolutionärer Algorithmen

Seit den Anfängen in den 60er Jahren haben sich die evolutionären Algorithmen enorm weiterentwickelt und bis heute sind viele Varianten veröffentlicht.

Zu Beginn unterschieden sich die einzelnen Algorithmen lediglich durch die Anzahl an Individuen, der Größe der neuen Generation, Zufallsauswahl von Eltern, wie oft Eltern als Basis für die neue Generation herangezogen werden können oder die prozentualen Werte von Mutationen etc. Man erkennt leicht, dass sich bereits aus diesen Freiheitsgraden sehr viele Algorithmen mit verschiedenen Eigenschaften generieren lassen, die auf viele spezielle Probleme erfolgreich angewendet werden können. Mit diesen variablen Grundeigenschaften ist die Vielfalt der evolutionären Optimierungsalgorithmen zu begründen.

Je spezialisierter die Problemstellungen werden, desto größere Unterschiede treten zwischen den Algorithmen auf. Es werden verschiedenste Mechanismen zur Steuerung der Populationsentwicklung genutzt, die in Beeinflussung der Auswahl und Beeinflussung der Erzeugung unterteilt werden können. Hier einige Beispiele:

Individuen-Erzeugung:

- Wichtungsvektor: Mit einem aktiven Eingreifen in eine Wichtung der Zielfunktionen kann die Entwicklungsrichtung der Population beeinflusst werden.
- Beschränkte Rekombination: Die Rekombination ist ebenfalls in vielen Arten variierbar. Neben der Auswahl der zu rekombinierenden Individuen basierend auf Nähe im Lösungs- oder Optimierungsraum kann die Rekombination selbst als Austausch von Parameterwerten, Mittelung, zufälliger Gewichtung oder gar gewichteter Vektoren durchgeführt werden.
- Mutation: Die Mutation kann auf vielfältige Weise durchgeführt werden: Sie kann die ursprünglichen Werte berücksichtigen, den Pareto-Rang, den Abstand der Indivi-

duen, die Richtung etc. Es wird deutlich, dass die Mutation wie die Rekombination als Basisfunktionalität dennoch enorme Freiheiten in der Erzeugung der Individuen zulässt.

Insgesamt können auch alle Maßnahmen, die zur Generierung neuer Individuen vorhanden sind, lediglich auf der letzten Generation aufbauen oder einen mehrere Generationen umfassenden Speicher nutzen.

Individuen-Auswahl:

- Dominanz: Die Dominanz spielt eine große Rolle für den Fortschritt und die Diversität des Algorithmus. Oft kommt es vor, dass die benötigte Anzahl an Individuen als Basis für die nächste Generation höher ist als die vorhandenen, nicht Dominierten Individuen. Die Entscheidung, welche dominierten Individuen die Basis für die nächste Generation komplettieren, kann von vielen Faktoren abhängig gemacht werden:
  - Von wie vielen Individuen wird der Kandidat dominiert?
  - Wie viele Individuen aus den höheren Pareto-Rängen dominiert der Kandidat?
  - In welchem Pareto-Rang findet man den Kandidaten?
- 'Fitness Sharing' oder 'Niching' beschreibt eine Methode, die basierend auf der Individuendichte, die Auswahl der Basisindividuen für die nächste Generation zusätzlich beeinflusst - Ziel ist hierbei ebenfalls die Erhöhung der Diversität. Die dafür nötige Distanzmessung zwischen zwei Individuen kann im Parameter- oder Lösungsraum stattfinden, um verschiedene Gleichverteilungen (Parameter oder Ziele) zu bevorzugen. Die Distanzmessung selbst kann ebenfalls auf verschiedene Arten durchgeführt werden - neben dem klassischen Ansatz nach Pythagoras, kann eine rasterbasierte oder gewichtete Abstandsbestimmung das Ergebnis variieren. [Muschalla, 2006]

Bereits diese wenigen Beispiele lassen erahnen, in welcher Vielfalt diese Algorithmen erzeugbar sind, aber auch welche Herausforderung es ist, einen effizienten Algorithmus zu generieren, der in allen Phasen einer Optimierung einen schnellen Fortschritt wahrscheinlich macht. Darüber hinaus können alle Entscheidungen in den Operationen aufgrund von globalen Zusammenhängen wie der vermuteten Form der Paretofront der optimalen Lösungen oder Erfahrungen über Generationen hinaus getroffen werden.

Weiterhin lässt sich vermuten, dass die Konstruktion eines spezialisierten Algorithmus auf die beiden Bereiche, Erzeugung und Auswahl der Individuen, Einfluss haben muss. Eine Erzeugung der Individuen mit anschließend unpassender Auswahl wird den potentiell guten Fortschritt im Sinne der angestrebten Lösung nicht nutzen können, während eine ungünstige Erzeugung von Individuen selbst mit einer optimalen Auswahl im Sinne der Problemstellung nur einen geringen Fortschritt erzeugen kann.

3.6 Vergleich der beiden Algorithmentypen

In den vorangegangenen Abschnitten wurde deutlich, dass sich Optimierungsverfahren basierend auf evolutionären Algorithmen und Gradientenverfahren in vielen Punkten unterscheiden, aber auch einige Gemeinsamkeiten besitzen.

So sind evolutionäre Optimierungen zu Beginn der Berechnung einer Problemstellung durchaus ein effektives Mittel, um mit wenigen Iterationen eine Paretofront aufzubauen. Gradientenverfahren konvergieren dagegen kontinuierlich gegen lokale Minima. Nähert sich die Paretofront den tatsächlich optimalen Werten an, sinkt die Konvergenzgeschwindigkeit der evolutionären Optimierungen stark ab und benötigt oft viele Iterationen um einen weiteren, merklichen Fortschritt zu erzielen. Hier sind die Gradientenverfahren wiederum im Vorteil, die eine sehr viel konstantere Konvergenzgeschwindigkeit besitzen.

Wie bereits im letzten Abschnitt angesprochen, können Gradientenverfahren, ähnlich wie evolutionäre Optimierungen, mit wenigen Informationen über die Aufgabenstellung auskommen. Es werden keinerlei Informationen über die Funktion und deren Ableitung benötigt.

Eine nahe liegende Idee ist daher, die beiden Algorithmentypen in einer Art und Weise zu kombinieren, die die Vorteile beider Verfahren optimal nutzt, um insgesamt schneller zu konvergieren und qualitativ hochwertigere Ergebnisse zu produzieren als jede der beiden Gruppen von Optimierungsverfahren.

3.7 Hybride Algorithmen

Allgemein versteht man unter einem hybriden Ansatz die Nutzung unterschiedlicher Prinzipien in einer Kombination und erhofft sich in dieser Symbiose die Vorteile beider Verfahren nutzen zu können.

Das Hybrid-Prinzip ist in vielen Bereichen des täglichen Lebens umgesetzt - man denke an Hybridantrieb, dessen energetische Optimierung Kraftstoff einspart, Züchtung von Nutzpflanzen um die Vorteile verschiedener Ursprungssorten zusammenzubringen, Stahlbeton der die Druckfestigkeit von Beton mit der Zugfestigkeit von Stahl kombiniert oder hybride Festplatten die einen kleinen, schnellen Flash-speicher mit einer günstigen, herkömmlichen Festplatte kombinieren, welche deutlich mehr Speichervolumen aufweisen kann.

Das Prinzip solcher Entwicklungen hat sich in Natur und Technik bewährt und kann im Rahmen von hybriden Optimierungsalgorithmen einen Vorteil bringen.

Nun gilt es, den richtigen Ansatz zur Kombination von traditionellen Optimierungsverfahren mit evolutionären Algorithmen zu finden. Dazu werden zunächst die generellen Vor- und Nachteile, soweit generalisierbar, in einigen Punkten definiert:

Evolutionäre Optimierungen:

+ Schnelle Konvergenz zu Beginn
+ Überwindung von lokalen Minima
+ Findet mehrere, ähnlich gute Lösungen
+ Detektiert potentiell die ganze Paretofront
- Konvergenzgeschwindigkeit sinkt mit der Nähe zu den globalen Minima
- Hohe Rechenzeit
- Keine Garantie, das globale Minimum zu finden

Demgegenüber stehen die **Gradientenverfahren**:

+ Schnelle Berechnungszeiten (Im Vergleich zu evolutionären Optimierungen)
+ Gleichmäßige Konvergenzgeschwindigkeit
+ Minimum wird auf jeden Fall gefunden
- Anfällig für lokale Extremstellen
- Anfällig für Diskontinuitäten
- Findet genau eine Lösung

Beim Vergleich der beiden Listen fällt schnell auf, dass die Ansätze schon im Grundsatz verschiedene Stärken und Schwächen besitzen und sich daher potentiell sehr gut ergänzen könnten. Eine optimale, hybride Lösung würde alle Vorteile vereinen und einen neuen Algorithmus schaffen, der jedem einzelnen überlegen ist.

Wegen der eingangs erwähnten, jeweils auf das spezielle Problem zugeschnittenen, optimierten Algorithmen, kann zunächst also nicht festgestellt werden, ob ein spezielles Gradientenverfahren zur Kombination mit evolutionären Algorithmen prädestiniert ist. Es kann aber vermutet werden, dass die konkrete Problemstellung einen größeren Einfluss auf die Wahl des Algorithmus hat. Um also eine generelle Aussage treffen zu können, muss das Verhalten des hybriden Optimierungsverfahrens bei verschiedensten Problemen berücksichtigt werden.

3.8 Parallelisierung

Ein einfaches und bewährtes Mittel die Dauer einer Berechnung zu reduzieren, stellt die Parallelisierung dar. Die simulationsgestützte, evolutionäre Optimierung ist strukturell sehr gut dafür geeignet auf mehreren Recheneinheiten parallel ausgeführt zu werden, wobei es zu beachten gilt, dass es mehrere Ansätze für diese verteilte Bearbeitung gibt (beschrieben z.B. in [Muschalla, 2006]):

1. Globale **Master-Slave** Optimierungssysteme mit einer einzelnen Population
2. **'Fine-Grained'** Optimierungssysteme mit einer einzelnen Population
3. **'Coarse-Grained'** Optimierungssysteme mit mehreren Populationen

Die **Master-Slave** Parallelisierung stellt die einfachste und intuitivste Form der Parallelisierung dar, da lediglich die benötigte Rechenleistung der Clients genutzt wird und alle Steuerungen vom Server aus erfolgen.

'Fine-Grained' nennt man eine Struktur, bei der im Lösungsraum benachbarte Individuen zu einer Teilpopulation zusammengefasst werden und deren Parameteroptimierung auf eine Recheneinheit ausgelagert wird. Ein wichtiger Schritt hierbei ist ein kontrollierter Austausch zwischen zwei Populationen.

'Coarse-Grained' bezeichnet ein Verfahren, bei dem mehrere Unterpopulationen gelegentlich Individuen austauschen.

Den beiden letzteren Verfahren ist gemein, dass sie die Arbeitsweise und damit die Resultate der evolutionären Optimierung beeinflussen.

3.9 Zusammenfassung

In den Abschnitten dieses Kapitels wurden die evolutionären Algorithmen vorgestellt. Dabei wurde klar, dass die Eigenschaften die ein solcher Algorithmus besitzt sehr vielfältig ausfallen können. Dies stellt einerseits einen Vorteil dar, da ein evolutionärer Algorithmus gut an das zu bearbeitende Problem angepasst werden kann, aber andererseits wiederum einen Nachteil, da diese Anpassung die universelle Nutzung einschränken kann. Gesucht ist eine Zwischenlösung, die sowohl in einfachen als auch speziellen Problemen einen Berechnungsvorteil in Hinblick auf die Konvergenzgeschwindigkeit bringt. Es wird darüber hinaus klar, dass die Parallelisierung und der hybride Ansatz jeweils ein hohes Potential bieten, um die Rechenzeit drastisch zu senken. Ziel ist es daher, beide Prinzipien im Entwurf zu berücksichtigen und optimal einzusetzen.

4 Entwurf

In der Entwurfsphase gilt es zunächst zu klären, welche Freiheitsgrade für die Umsetzung im Rahmen der Vorgaben aber auch des vorhandenen Programms bestehen und welche möglichen Strategien bereits im Ansatz auswählbar sind oder als nicht geeignet identifiziert werden können. Darüber hinaus gilt es zu beachten, dass ein System entwickelt werden muss, welches den hybriden Ansatz verfolgt, effizient umsetzt und die Modularität als zentrales Programmierparadigma berücksichtigt und damit auch von der Problemstellung abhängige, spezielle Konfigurationen ermöglicht.

4.1 Bestehendes System

Das bestehende System ist eine Sammlung mehrerer Projekte die mehrheitlich in Visual Basic, sonst in C# geschrieben sind. Das Verwenden solcher unterschiedlicher Projekte, die wie in diesem Fall zusammenarbeiten, ist sehr einfach möglich und wird durch Microsoft Visual Studio gut unterstützt. Bei der Analyse fällt auf, dass die Software eine Hierarchie aufweist, die darauf schließen lässt, dass sie zunächst als kleine Anwendung erstellt wurde und im Laufe der Zeit mit weiteren Anforderungen gewachsen ist. So sind vielfach die grafische Oberfläche und die dahinter stehende Logik in derselben Datei zu finden was die Erweiterbarkeit stark einschränkt. Jedoch wurden bereits einige Projektteile ausgegliedert, die grundlegende Komponenten enthalten:

- Die Individuen-Objekte
- Die Basisalgorithmen
- Indikatoren zur Berechnung von Eigenschaften einer Generation
- Simulationsmodelle

Im Laufe dieser Diplomarbeit werden einige Umstrukturierungen vorgenommen, die die Erweiterbarkeit so erhöhen, dass das Einfügen der gestellten Aufgabe als weiteres Projekt ohne Probleme möglich ist und alle Anbindungen zu den Grundstrukturen und der Visualisierung auch für nachfolgende Projekte in gleicher Weise einfach durchzuführen ist.

Zur Laufzeit des Systems kann bereits aus den Beobachtungen im vorhandenen Projekt gesagt werden, dass die Simulation eines Individuums den weitaus größten Teil der Rechenzeit benötigt. Gefolgt von der Visualisierung nimmt die tatsächliche Laufzeit der evolutionären Algorithmen den geringsten Teil ein. Diese allgemeine Beobachtung ist allerdings im Rahmen der im System vorhandenen Testfunktionen verfälscht, da hier zur schnellen Überprüfung der Optimierungsstrategien keine umfangreiche Simulation, sondern nur einfache Funktionen ausgewertet werden. Dies gilt es bei der Konstruktion und anschließenden Benchmarkläufen mit diesen Testfunktionen zu berücksichtigen. Es ist also nötig, primär die Anzahl der Simulationen zu minimieren die benötigt werden

um die Paretofront der optimalen Lösungen zu erreichen, aber auch diejenigen zu minimieren, die auf eine Erhöhung der Diversität abzielen jedoch ohne qualitative Einbußen hinnehmen zu müssen.

4.2 Basisstruktur

Als Basisstruktur kommen mehrere Klassen zum Einsatz, die untereinander klar hierarchisch angeordnet sind:

Die oberste Klasse stellt der **Controller** dar: Diese zentrale Instanz regelt alle grundlegenden Einstellungen bzw. gibt diese bei der Initialisierung an die Hauptkomponenten weiter. Nach der Initialisierung wird, basierend auf der Entscheidung des Nutzers, zunächst geregelt um welche Art von Programmausführung es sich handelt. D.h. rechnet die Einheit selbstständig oder ist sie Teil eines Netzwerks als Client oder Server. Die Modi werden Single PC, Network Server und Network Client genannt. Der Controller regelt darüber hinaus die Laufzeit der Optimierung, veranlasst Simulation und Zeichnen der Individuen außerhalb des Projekts - ist also generell für die Steuerung der übrigen Klassen zuständig.

Basierend auf dem gewählten Berechnungsmodus wird der **Networkmanager** ebenfalls als Server, Client oder gar nicht initialisiert. Der Networkmanager dient indirekt auch der Kommunikation des Servers mit den Clients. Der Manager muss es den Clients ermöglichen, Statusinformationen auszutauschen und Individuen für alle Clients und den Server frei zugänglich abzulegen. Des Weiteren ist der Networkmanager zuständig für die Zuteilung der Individuen auf die Clients in der Art, dass eine möglichst geringe Gesamtrechenzeit pro Generation benötigt wird. Eine weitere wichtige Anforderung für den Networkmanager ist es, auf strukturelle Änderungen des Berechnungsnetzwerkes reagieren zu können.

Neben dem Networkmanager muss eine weitere **Network**-Klasse konstruiert werden, die die Eigenschaften der Clients als Objekt schnell und komfortabel zugänglich macht und dem Networkmanager so die Verwaltung des Netzwerks vereinfacht.

Der **Algomanager** dient im Wesentlichen dem Durchführen der allgemeinen Aufgaben im Zyklus einer evolutionären Optimierung und wird im Server- und Single PC Modus initialisiert. Neben der Dominanzprüfung und dem Zusammenstellen des neuen Genpools aus immer neu erzeugten Individuen, muss diese Klasse auch die Vorbereitungen treffen um die Algorithmen auszuführen, die für die Erzeugung dieser neuen Individuen notwendig sind. Dieser Manager steuert auch die Aufrufe der Algorithmusklasse.

Die Klasse der **Algo**rithmen dient der Erzeugung von Individuen und muss so aufgebaut sein, dass gewählt werden kann, welche Algorithmen in welchem Verhältnis bei der Erzeugung neuer Individuen zur Anwendung kommen. Es muss eine Struktur bereitgestellt werden, die es diesen Algorithmen ermöglicht, individuelle Informationen zu speichern die später in einer weiteren Iteration genutzt werden können.

Als Datenbasis der Zusammenarbeit wird der **Genpool** sowie die **neue Generation** definiert. Der Genpool soll nach jedem Evolutionszyklus die Individuen enthalten, die als Eltern für die neue Generation dienen. Eine Referenz auf diesen Genpool wird an die verschiedenen Hauptprozesse übergeben. Die neue Generation enthält die Individuen, die nach der Simulation, basierend auf den Zielfunktionen, mit dem Genpool vermischt werden.

4.3 Evolutionäre Algorithmen

Im Bereich der Anwendungen die mit evolutionären Optimierungen bearbeitet werden können, gibt es beinahe ebenso viele Algorithmen wie Aufgaben. Dies liegt zum Einen daran, dass viele Algorithmen im Laufe der Zeit verbessert werden, aber auch daran, dass diese Verbesserungen eine Spezialisierung mit sich bringen und für eine geringere Anzahl an Problemen effektiv einsetzbar sind. Um also einen möglichst universellen Algorithmus zu kreieren, muss auf hoch spezialisierte Ansätze verzichtet und somit eine durchschnittlich niedrigere Konvergenzgeschwindigkeit in Kauf genommen werden.
Eine Möglichkeit, den Vorteil einer in Grundzügen vorhandenen Spezialisierung zu nutzen und dennoch keine Flexibilität bei der Problemwahl einzubüßen, ist das parallele Nutzen verschiedener erzeugender Algorithmen. Ein mögliches Konzept dies effektiv umzusetzen wird in Kapitel 4.5.1 angesprochen. Dennoch bleibt festzuhalten, dass auch hier eine weitere Erhöhung der Konvergenzgeschwindigkeit für ein bestimmtes Problem vor allem mit stärker spezialisierten Algorithmen zu erreichen ist.

Als Basisalgorithmen aus der Klasse der Evolutionsstrategien kommen zum Einsatz:

Mutationen:

Totaler Zufall: Hier wird innerhalb der vorgegebenen Grenzen der Optimierungsparameter ein zufälliges Individuum erzeugt. Dieser Algorithmus ist vor allem zu Beginn der Berechnung effektiv um sich innerhalb des Parameterraumes zum Optimum hin zu orientieren.

Zufällige, einfache Mutation: Diese Mutation überschreibt einen Optimierungsparameter innerhalb des ihm vorgegebenen Maximal- und Minimalwertes. Hier hat der Ausgangswert keine Relevanz. Ziel dieses Algorithmus ist es, lokale Minima zu überwinden.

Ungleichverteilte Mutation: Dieser Algorithmus mutiert einen vorhandenen Optimierungsparameter innerhalb der vorgegebenen Grenzen. Kleinere Mutationen sind hier wahrscheinlicher als große. Die Variation des Optimierungsparameters berechnet sich hierbei aus der Zufallszahl x:

$$Opt_{+} = Opt * f(x)\ mit\ 0,75 < f(x) < 1,5$$
$$x < |1|\ ;\ x < 0 : f(x) = (\tfrac{1}{4}x^3 + 1)\ ;\ x > 0 : f(x) = (\tfrac{1}{2}x^3 + 1)$$

Diese Mutation bewirkt eine Veränderung in den oben angegebenen Grenzen und passt sich den Absolutwerten der Optimierungsparameter an.

Rekombinationen:

Zufällige Rekombination: Hier werden die Optimierungsparameter zweier beliebiger Individuen zufällig zusammengesetzt um eine neues Individuum zu erschaffen. Ziel ist es, erfolgreiche Kombinationen von Optimierungsparametern weiter zu verbessern.

Intermediäre Rekombination: Bei dieser Art der Rekombination werden die Optimierungsparameter der beiden Eltern zufällig gewichtet. Diese Art der Rekombination dient dem Schließen von Lücken in der aktuellen Paretofront.

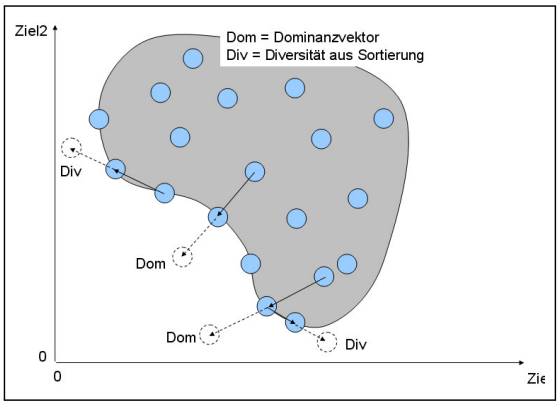

Abbildung 4.1 – Dominanzvektor und Diversität aus Sortierung

Diversität aus Sortierung: Die Diversität stellt, wie bereits erwähnt, ein Qualitätsmaß der Lösung dar. Dieser Algorithmus versucht nach einer zufälligen Sortierung der vorhandenen Lösungen die nach dieser Sortierung ersten und letzten Lösungen so zu kombinieren, dass sich die Breite der aktuellen Paretofront erhöht. Damit verfolgt dieser Algorithmus als einziger Basisalgorithmus gezielt die Diversität.

Dominanzvektor: Dominanzvektor arbeitet nach einem ähnlichen Prinzip wie der Algorithmus Diversität aus Sortierung. Es wird versucht, durch Linearkombination vorhandener Lösungen eine bessere zu erschaffen. Dabei wird in diesem Fall ein Individuum und ein von diesem Individuum dominiertes Individuum gesucht, um damit die Richtung zur Paretofront der optimalen Lösungen abschätzen zu können und mit einer gezielten Differenzbildung der Optimierungsparameter einen großen Schritt in diese Richtung voranzukommen.

Die beiden letztgenannten Algorithmen arbeiten allerdings nur dann erfolgreich, wenn es einen linearen Zusammenhang zwischen den Optimierungsparametern und dem Zielfunktionsraum gibt. Andernfalls ist der Erfolg vom Zufall abhängig. Diverse Durchläufe in Testfunktionen zeigen allerdings im linearen Fall eine beträchtliche Zunahme der Konvergenzgeschwindigkeit. Ebenfalls zu bemerken bleibt, dass 'Diversität aus Sortierung' und 'Dominanzvektor' im Rahmen dieser Diplomarbeit entwickelte, experimentelle Algorithmen darstellen und aufgrund dessen noch nicht in umfangreichen Testläufen optimiert wurden.

Diese Gruppe von Algorithmen soll als Basis für die evolutionäre Optimierung dienen. Im Sinne der Modularität müssen diese Algorithmen leicht austauschbar bzw. die Größe der Algorithmengruppe ebenso leicht anpassbar sein.

4.4 Gradientenverfahren

Aus dem Gebiet der Gradientenverfahren dient der Hook&Jeeves-Algorithmus, wie in [Syrjakow, 2005] beschrieben, als erster Ansatzpunkt. Der Algorithmus zeichnet sich durch seine Robustheit in der Konvergenz aus und eignet sich daher gut, nach einer evolutionären Optimierung eingesetzt zu werden.

Als erster Schritt wird nacheinander jeder Optimierungsparameter mit vorher festgelegter Schrittweite variiert und das neue Individuum simuliert. Nachdem alle Optimierungsparameter einmal variiert worden sind, erhält man so ein Individuum, welches sich in den meisten Optimierungsparametern vom ersten Individuum unterscheidet und insgesamt geringere Zielfunktionswerte besitzt. Die Differenz der Optimierungsparameter des ersten und des letzten Individuums ergeben einen erfolgversprechenden Offset-Wert für die nächste Variation.

Schematisch erhält man folgendes:

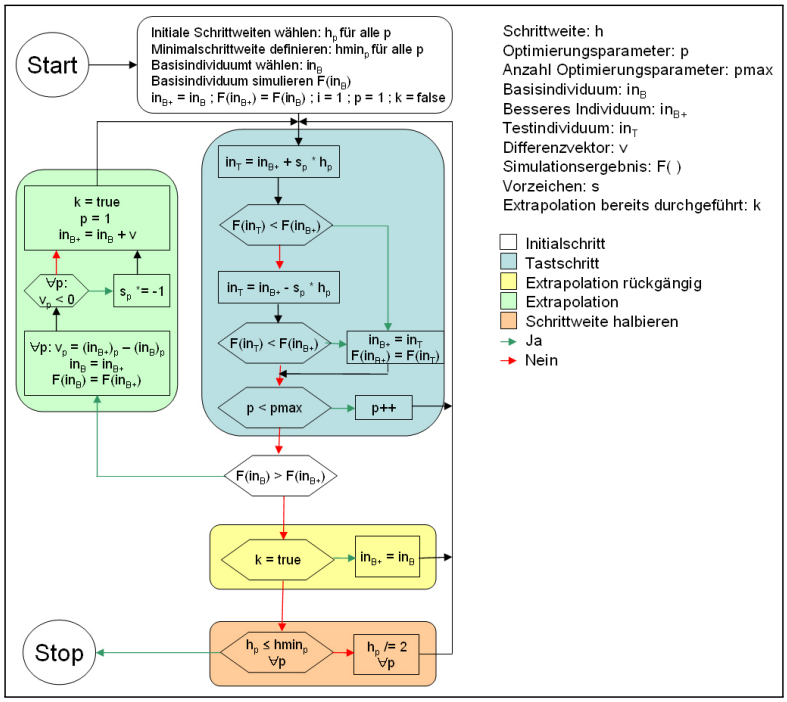

Abbildung 4.2 – Alg.: Gradientenverfahren nach Hook&Jeeves

Initialschritt:
Im initialen Schritt ist die Wahl der Schrittweite ein wichtiger Indikator für die Laufzeit des Algorithmus. Ist dieser zu klein, wird der Algorithmus mehrere Iterationen benötigen um die maximale Konvergenzgeschwindigkeit zu erreichen. Ist die Schrittweite hingegen zu groß gewählt, wird der Algorithmus erst nach mehreren Iterationen eine Schrittweite erreicht haben, die nicht über die Gültigkeitsgrenzen hinaus geht. Die Minimalschrittweite

dient als Abbruchkriterium und kann für jeden Optimierungsparameter einzeln definiert werden. Da die Schrittweite für alle Optimierungsparameter zur gleichen Zeit halbiert wird, sollte für eine optimale Gesamtlaufzeit die Initiale Schrittweiten in gleichen Faktoren von den Minimalschrittweiten abhängig definiert werden.

Tastschritt:
Bei jeder Ausführung des Tastschrittes wird jeweils ein Optimierungsparameter p um die Schrittweite hp verändert. Ist der Wert der Zielfunktion höher als zuvor, wird mit dem nächsten Optimierungsparameter fortgefahren. Sollte dies nicht der Fall sein, wird der Optimierungsparameter vom Ausgangspunkt mit der inversen Schrittweite verändert. Sollte hier ebenfalls keine Verbesserung auftreten, wird ohne Veränderung mit dem nächsten Optimierungsparameter fortgefahren. Sind alle Optimierungsparameter einmal geprüft und ggf. verändert worden, wird im nächsten Schritt geprüft, ob das nun neu erzeugte Individuum in_{B+} besser als das ursprüngliche Basisindividuum in_B ist.

Extrapolation:
Nach erfolgreichem Finden des neuen Individuums wird versucht, aus der Veränderung der Optimierungsparameter die zu dieser Verbesserung führten nicht nur den Vorzeichenvektor so anzupassen, dass ein erneuter Verbesserungsversuch möglichst sofort erfolgreich ist, sondern auch die Veränderung selbst als Offset-Vektor bereits in den neuen Testpunkt in_{B+} einfließen zu lassen.

Extrapolation Rückgängig:
Falls eine solche Extrapolation nicht erfolgreich verläuft - d.h. keine der Parameteränderungen eine Verbesserung der Zielfunktion erwirkt, wird die Extrapolation wieder rückgängig gemacht und anschließend ein weiterer Tastschritt ohne die vorherige Anwendung einer Extrapolation ausgeführt.

Schrittweite halbieren:
Sollte auch ohne Extrapolation keine Verbesserung der Zielfunktionswerte durch Parameterveränderung zu erreichen sein, ist das Minimum bereits in Schrittweite des aktuellen Punktes zu finden. Sobald diese Situation eintritt, wird die Schrittweite eines jeden Optimierungsparameters halbiert.

Es fällt auf, dass Hook&Jeeves, wie alle Gradientenverfahren, auf genau eine Zielfunktion ausgerichtet ist. Um dennoch multikriterielle Ziele zu verfolgen ist es nötig, alle Zielfunktionen in einer einzigen, neuen Zielfunktion zu vereinen. Dies erfordert eine feste Gewichtung der einzelnen Zielfunktionen. Da eine feste Gewichtung allerdings auf einer Paretofront, trotz verschiedener Ausgangspunkte, zu immer den wenigen gleichen Ergebnissen führt, in sofern es sich um mehrere Minima handelt, ist die Vielfalt durch die Wahl der Gewichtung stark beeinflusst.

Um diese Vielfalt zu wahren ist es möglich, für jedes der parallel ablaufenden Hook&Jeeves-Verfahren eine eigene Gewichtung zu definieren. Als Grundlage dienen hier die Zielfunktionswerte der Ausgangsindividuen wie in [Kalyanmoy Deb, 2001] beschrieben:

Der Wichtungsfaktor einer Zielfunktion ergibt sich wie folgt:

$$W_{\mathrm{i}} = \frac{\left(f_i^{max} - f_i(x)\right)/\left(f_i^{max} - f_i^{min}\right)}{\sum_{m=1}^{M}\left(f_m^{max} - f_m(x)\right)/\left(f_m^{max} - f_m^{min}\right)}$$

Diese Art der Bestimmung nutzt die Positionsverhältnisse der Individuen zueinander um daraus eine proportionale Verteilung der Wichtungsfaktoren zu bestimmen. D.h. die Individuen konvergieren durch diese Berechnung der Wichtungsfaktoren in folgender Weise:

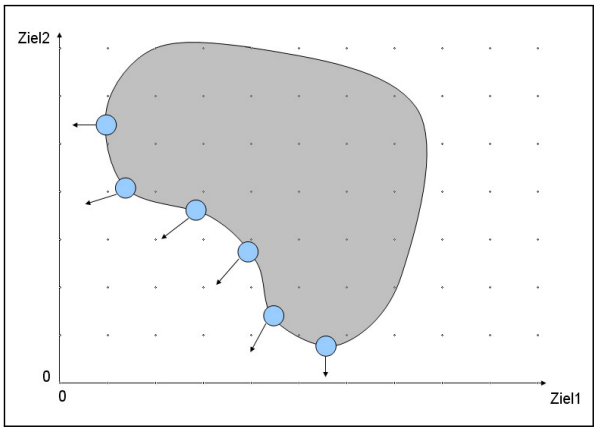

Abbildung 4.3 – Hook&Jeeves Wichtungsfaktoren

Hier kommt es jedoch sehr auf die Form der Paretofront der optimalen Lösungen an, denn die Gleichverteilung der Individuen ist lediglich durch die initiale Vergabe der Wichtungsfaktoren begünstigt, keinesfalls aber garantiert.
Jedes Individuum wird sich separat von seinem Standpunkt aus in die Richtung entwickeln, die eine Verbesserung der gewichteten und summierten Zielfunktionen in Pfeilrichtung ergibt. Falls die in diesem Beispiel eingezeichnete, graue Menge der möglichen Zielfunktionswert-Kombinationen vollständig ist, d.h. es keine Verbesserung mehr in eine Zielrichtung gibt, würden die beiden zentralen Individuen auf der Paretofront jeweils zu ihren äußeren Nachbarn wandern. Das Resultat wäre also eine ungleiche Verteilung der Individuen auf der Paretofront mit einer Häufung bei solchen Minima, die am Rand der Paretofront zu finden sind.

Die letztendlich erhaltenen Lösungen sind also sehr von der Form der Paretofront der optimalen Lösungen abhängig wobei bedacht werden muss, dass zum Zeitpunkt der Berechnung der Wichtungsfaktoren diese Form nicht bekannt ist. Ein Vorteil dieses Vorgehens ist allerdings, dass die Diversität im Allgemeinen profitiert, da die einzelnen Prozesse bestrebt sind, sich in sehr verschiedene Richtungen zu entwickeln.

Um eine bessere Verteilung auf der Paretofront zu erreichen ist es nötig, den bestehenden Algorithmus von Hook&Jeeves zu erweitern. Das Grundkonzept ist, die Wichtungsfaktoren, basierend auf dem euklidischen Abstand der Individuen im Lösungsraum, anzupassen.

Grundsätzlich müssten folgende Schritte unternommen werden:

- Sortieren der Individuen nach Position auf der Paretofront

- Abstand zu beiden Nachbarn im Lösungsraum bestimmen
- Anpassen der Wichtungsfaktoren

Diese Anpassung kann an verschiedenen Stellen des Algorithmus umgesetzt werden.

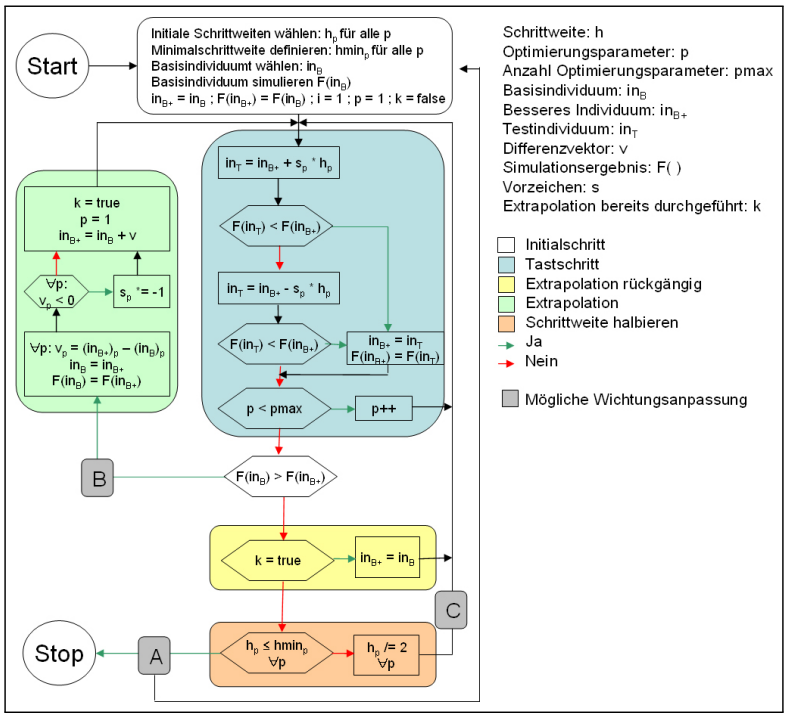

Abbildung 4.4 – Alg.: Hook&Jeeves mit Wichtungsanpassung

A:
Nachdem das endgültige Individuum gefunden wurde, werden die Wichtungsfaktoren bei Bedarf angepasst und die Berechnung noch einmal komplett durchgeführt. Dabei kann die initiale Schrittweite des Algorithmus, basierend auf dem euklidischen Abstand im Parameterraum zum Nachbarn, definiert werden. Da diese Anpassung erst durchgeführt wird nachdem die erste Berechnung beendet ist, sind die meisten Individuen bereits an ihrem, nach bisheriger Gewichtung der Zielfunktionen, endgültigen Platz und die Anpassung muss tendenziell seltener durchgeführt werden.

B:
Wann immer ein besseres Individuum gefunden wurde, werden die Wichtungsfaktoren angepasst. Diese Anpassung wird zwar oft durchgeführt, kann aber die Entwicklungsrichtung bereits früh beeinflussen wodurch tendenziell weniger Iterationen des gesamten Algorithmus und damit auch weniger Simulationen nötig sind als im Fall A. Ein Nachteil hieran ist, dass die Anpassung bereits zu einer Laufzeit des Algorithmus ausgeführt wird, in dem die einzelnen Hook&Jeeves-Prozesse potentiell die größten Entwicklungsunterschiede aufweisen und daher häufiger eine Korrektur erfolgen.

C:
Die Anpassung an C ist ähnlich zu B nur wird diese erst zu einem Entwicklungszeitpunkt durchgeführt, in dem die Paretofront der optimalen Lösungen bereits in Reichweite ist. Die Entwicklungsrichtung des Prozesses ist tendenziell zwischen den Resultaten von A und B angesiedelt. Die Korrektur der richtung in Form einer Wichtungsfaktor-Anpassung wird später ausgelöst und ist damit und ähnlich zu Möglichkeit A, was in weniger, größeren Korrektur und damit späteren Konvergenz in die endgültige Zielrichtung resultiert.

Es ist zu beachten, dass diese Modifikationen lediglich eine gleichmäßigere Ausfüllung eines jeden Minimums bewirken können. Eine komplett gleichmäßige Verteilung auf einer vom Nullpunkt aus gesehen konkaven Paretofront ist mit diesem Verfahren wahrscheinlich nicht ohne tieferen Eingriff in die Bewertungsfunktion möglich. Diesen Anpassungen ist gemein, dass sie die ursprüngliche Laufzeit des Hook&Jeeves-Algorithmus nur erhöhen können. Da diese Arbeit einen starken Fokus auf die Konvergenzgeschwindigkeit legt, kann über die Umsetzung erst nach einer Analyse des angestrebten hybriden Ansatzes entscheiden werden.

4.5 Hybrider Algorithmus

Eine hybride Form die mehrere Vorteile von Algorithmen nutzbar macht, kann an mehreren Stellen des geplanten Vorgehens verwirklicht werden.

Zunächst kann eine Kopplung der globalen und lokalen Optimierung die Vorteile der beiden Algorithmenklassen (siehe Abschnitt 3.7) vereinen. Dazu ist bei einer sequentiellen Kopplung der Übergang zwischen den beiden Optimierungen von zentraler Bedeutung und wird im Folgenden als Umschaltpunkt bezeichnet.

4.5.1 Umschaltpunkt

Nach Untersuchung der Berechnungsstruktur der evolutionären Algorithmen im Vergleich zu Hook&Jeeves wird klar, dass eine parallele Verwendung beider algorithmenklassen schwer möglich ist, da keine vergleichbaren Individuen während der Laufzeit erzeugt werden. Dies wird deutlich, wenn man die Struktur von Hook&Jeeves betrachtet:

Wird ein im Tastschritt gefundenes Individuum als besser als das Basisindividuum bewertet, so dient die gefundene Optimierungsparameter-Differenz als Offset-Wert (Extrapolation) zur weiteren Untersuchung im nächsten Tastschritt und stellt damit dem effizientesten Teil des gesamten Algorithmus dar.

Es ist also zu erkennen, dass Hook&Jeeves bei der Konvergenzgeschwindigkeit tendenziell sprunghaft zunimmt bis die Paretofront bei einem weiteren Tastschritt in Reichweite

kommt. Ist die Paretofront der optimalen Lösungen erreicht oder unterschritten, würde der Algorithmus dann oft zur Schrittweitenhalbierung tendieren.

Im Gegensatz dazu kann man an den entstehenden Individuen im Fall der evolutionären Optimierung ein gleichmäßigeres und einmaliges zu- und abnehmendes Verhalten im Bezug auf die Konvergenzgeschwindigkeit beobachten.

Aus diesem Verhalten wird deutlich, dass der Kontext in dem die Individuen entstehen, maßgeblich die Vergleichbarkeit beeinflusst und in diesem Fall unmöglich macht. Die Unvergleichbarkeit der Lösungen der beiden Algorithmenklassen macht damit klar, dass ein hybrider Algorithmus nicht ohne Weiteres die Vorteile aus parallel ablaufenden Algorithmen verschiedener Typen ziehen kann. Daraus ergibt sich die Untersuchung eines sequentiellen Ablaufes der beiden Optimierungsverfahren.

Wird die evolutionäre Optimierung vor den Gradientenverfahren ausgeführt, kann man folgendes Verhalten vermuten:

- Zu Beginn sorgt die evolutionäre Optimierung dafür, dass eine schnelle 'Abtastung' des Parameterraums frühzeitig die Entwicklungsrichtung definiert
- Durch die vielen gravierenden aber gezielten Änderungen an den Optimierungsparametern ist eine schnelle Konvergenz am Anfang zu erwarten
- Ebenfalls durch die gravierenden Änderungen ist die Überwindung lokaler Minima zu erwarten
- Ein weiterer Vorteil der bedeutenden Änderungen und des raumgreifenden Beginns ist die zu erwartende hohe Diversität
- Je mehr sich die aktuelle Paretofront der Paretofront der optimalen Lösungen nähert, desto geringer wird die Konvergenzgeschwindigkeit
- Das Ausführen eines Gradientenverfahrens für jedes von den Evolutionsstrategien erzeugte Individuum, bewirkt das zuverlässige Finden eines Minimums in der jeweiliger Nähe
- Durch die gewählte Methode zur Verarbeitung multikriterieller Ziele, kann, abhängig vom Verlauf der Paretofront der optimalen Lösungen, keine gleichmäßige Abdeckung gewährleistet werden.

Ein Ziel ist es also, vor Ende der globalen Optimierung auf die lokale Optimierung umzuschalten und so immer die maximal mögliche Konvergenzgeschwindigkeit zu erzeugen. Idealerweise ist dieser Umschaltpunkt erreicht, sobald sich die Individuen in den vermeintlichen Tälern des Lösungsraums befinden:

In diesem Beispiel der Testaufgabe Zitzler/Deb/Theile T3 ist diese Situation gut zu erkennen:

Die Punkte stellen die Werte der Individuen im Lösungsraum dar. Dabei gehören gelbe Punkte zu Individuen eines höheren Pareto-Rangs oder wurden gerade erst erzeugt und

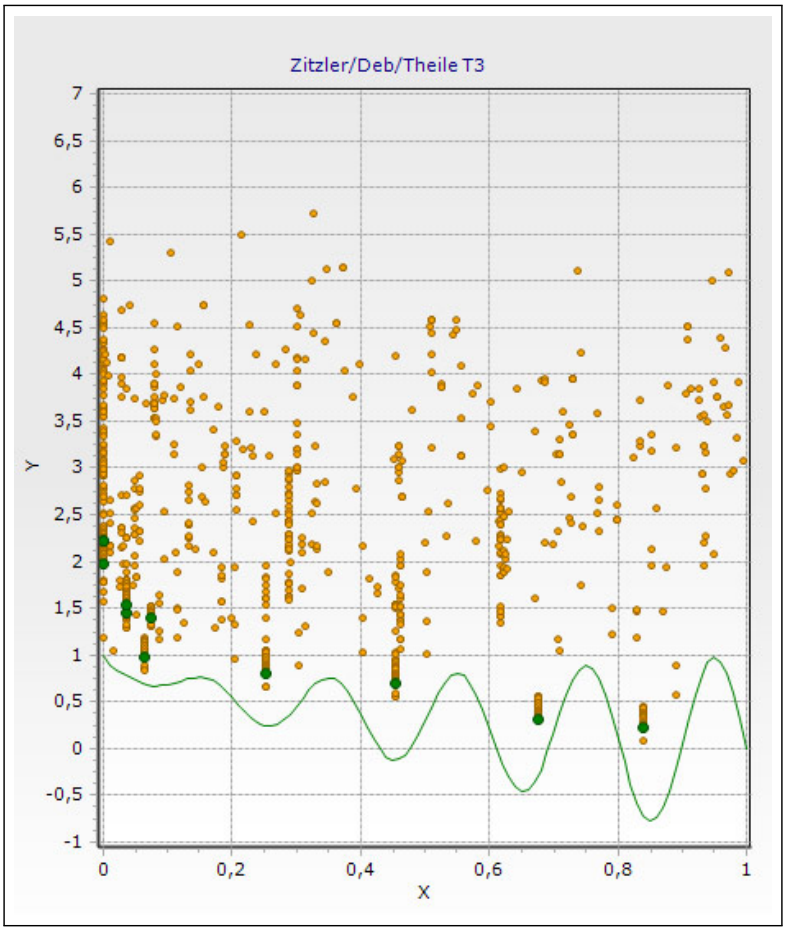

Abbildung 4.5 – Testproblem: Zitzler/Deb/Theile T3 Umschaltpunkt

der Selektionsschritt wurde noch nicht ausgeführt. Die grünen Punkte gehören zu Individuen, die von der Selektion als Eltern-Individuen ausgewählt wurden. Die grüne Linie stellt die Paretofront der optimalen Lösungen dar. Es ist gut erkennbar, dass Individuen die sich bereits in den Tälern der Lösungsmenge befinden, mit dem Hook&Jeeves-Verfahren wahrscheinlich schnell gegen das jeweils vorhandene, lokale Minimum konvergieren.

Um den gesuchten Umschaltpunkt zu definieren und eine Steuerung daraus zu entwickeln, muss zunächst geklärt werden, welche Eigenschaften eine Paretofront besitzt um daraus Indikatoren zur Klassifizierung des Umschaltpunktes definieren zu können.

Eigenschaften einer Paretofront:

- Nähe zur Paretofront der optimalen Lösungen (muss minimiert werden)
- Diversität (muss maximiert werden)
- Dichte der Front (die Individuen sollten gleichmäßig verteilt sein)

Indikatoren für diese Eigenschaften könnten sein:

- Nähe zur Paretofront der optimalen Lösungen:
  Die Nähe zur Paretofront der optimalen Lösungen kann mit Hilfe der durchschnittlichen Entfernung zum Ursprung oder eines festen Werts im Lösungsraum bestimmt werden. Der Nachteil hierbei ist, dass eine Verbreiterung der Paretofront bei einer solchen Berechnung eine rechnerisch negative Auswirkung haben kann.
- Diversität:
  Die Diversität lässt sich mit Hilfe eines Vergleichsindividuums bestimmen. So kann aus allen Individuen im Lösungsraum ein Durchschnittsindividuum gebildet werden und die Summe der jeweiligen Abstände klassifiziert die Front. Negativ hierbei wäre, dass wachsende Unregelmäßigkeiten, welche die Dichte der Front betreffen, rechnerisch nicht erfassbar sind.
- Dichte der Front:
  Die Dichte kann bestimmt werden, indem jedes Individuum den Abstand zu seinen Nachbarn mit einer quadratischen Funktion bewertet. In diesem Fall verursachen sehr nahe Nachbarn einen hohen Dichtewert wohingegen entfernt Nachbarn den Dichtewert kaum beeinflussen. Sind diese Werte bei allen Individuen ähnlich, ist die Dichte gleichmäßig. Jedoch ist eine Paretofront in der alle Individuen an der gleichen Stelle zu finden sind ebenfalls sehr gleichmäßig, besitzt aber offensichtlich nicht die erstrebte Form.

Jede der Eigenschaften kann also bestimmt werden, ist aber, wie gezeigt, von anderen Eigenschaften abhängig und beeinflusst diese wiederum. Eine weitere Fragestellung wäre, welche dieser Indikatoren in welchem Maß den Qualitätswert für die Paretofront bestimmen.

Es wird bereits hier deutlich, dass eine separate Betrachtung der Eigenschaften schwierig ist und daher eine Steuerung, basierend auf einer dieser Eigenschaften, von Grund auf fehleranfällig in besonderen Situationen sein kann. Ebenso muss die Qualität der Front auch in Hinblick auf die Eigenschaften des Problems gesehen werden: Eine gleichmäßige Verteilung auf der Paretofront wäre im Beispiel Zitzler/Deb/Theile T3 gar nicht möglich, da diese unstetig ist.

4.5.2 Entwicklungsbewertung mit Solutionvolume

Um dennoch über eine Umschaltung entscheiden zu können, muss ein Indikator geschaffen werden, der allgemeiner ansetzt. Hierbei muss in Kauf genommen werden, dass detaillierte und aussagekräftige Informationen der Paretofront nicht erzeugbar sind.

Der neue Operator soll, basierend auf dem Indikator zur Nähe der Paretofront, den Abstand zu einem festen Wert im Lösungsraum berücksichtigen. Hingegen soll sich die Vergrößerung der Diversität rechnerisch nicht als Nachteil herausstellen. Eine weitere Anforderung ist die Immunität gegen Schwankungen dieses Indikators; es muss dementsprechend ein

längerer Zeitraum der Entwicklung berücksichtigt werden und daraus eine Tendenz abgeleitet werden.

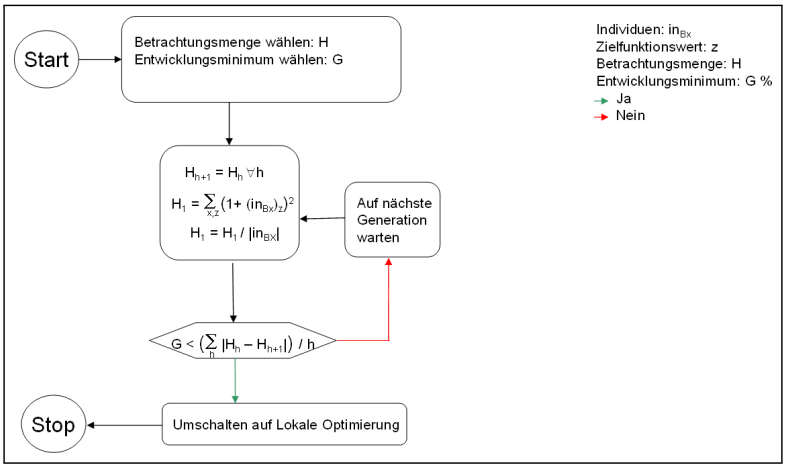

Abbildung 4.6 – Alg.: Umschaltpunkt mit Solutionvolume

Dieser Algorithmus wählt zunächst die Größe h der Betrachtungsmenge H. Diese Menge enthält Werte von h Generationen die in die Berechnung der durchschnittlichen Entwicklung einbezogen werden um so den Auswirkungen von Fluktuationen entgegenzuwirken. Die Grenze für die durchschnittliche Entwicklung darf den Wert G nicht unterschreiten.

Liegen neu simulierte Individuen vor, wird jeweils das summierte Abstandsquadrat aller Zielfunktionen jedes Individuums von einem vorher bestimmten Punkt im Lösungsraum bestimmt und summiert. Es liegen jeweils die Ergebnisse mehrerer Generationen vor. Das älteste Ergebnis in der Betrachtungsmenge wird gelöscht. Aufgrund dieses Vorgehens kann der Algorithmus erst nach h Generationen entscheiden, ob die Konvergenzgeschwindigkeit das Umschalten zur lokalen Optimierung nötig macht. Im Entscheidungsschritt wird überprüft, ob die Summe der Differenzen der einzelnen aufeinanderfolgenden Generationenwerte die minimale Entwicklungsgeschwindigkeit unterschreitet. Ist dies der Fall, so bewirkt der Algorithmus das Umschalten zur lokalen Optimierung. Die Überprüfung der Differenz der Generationenwerte wird mit einem Betrag beeinflusst um zu verhindern, dass eine zu frühe Umschaltung zur lokalen Optimierung ausgelöst wird. Diese falsche Umschaltung könnte durch eine Erhöhung der Diversität ausgelöst werden da in diesem Fall die Summe der Abstandsquadrate steigen kann. Durch die Nutzung des Betrags wird eine solche Erhöhung der Diversität nicht nur toleriert, sondern sogar als Entwicklungserfolg des evolutionären Algorithmus ausgelegt und fließt daher positiv in den durchschnittlichen Entwicklungswert ein.

Der hier vorgestellte Indikator misst also lediglich die prozentuale Änderung die in der Paretofront zwischen zwei Generationen vorkommt. Dabei ist durch die Struktur des Algorithmus definiert, dass eine Änderung tendenziell nur bessere Individuen produzieren kann und somit wird eine Änderung der Paretofront mit einer Verbesserung im Bezug auf Nähe zur Paretofront der optimalen Lösungen ODER eine Erhöhung der Diversität gleich-

gesetzt. Im Umkehrschluss kann gesagt werden, dass zu einer Laufzeit in der prozentual keine ausreichende Verbesserung des Indikatorwertes erzielt wird, der Algorithmus weder eine ausreichenden Verbesserungen in Richtung Paretofront der optimalen Lösungen, noch im Bezug auf die Diversität erzeugen kann um die Konvergenzgeschwindigkeit hoch zu halten. Das Umschalten zur lokalen Optimierung wäre nun von Vorteil.

4.5.3 Algorithmenparallelisierung mit Initiative

In den beiden vorherigen Abschnitten wurde auf den hybriden, sequentiellen Einsatz der beiden Optimierungstypen eingegangen und die daraus resultierenden Problemen und Lösungen im Bezug auf den Umschaltpunkt.

Eine weitere Möglichkeit, eine hybride Funktionalität umzusetzen, bieten die evolutionären Algorithmen selbst. So sind die in Abschnitt 4.3 angesprochenen Algorithmen zwar nicht viel mehr als einfache Modifikationen der einfachsten Algorithmenformen, aber sie sind dennoch in verschiedenen Situationen sehr verschieden erfolgreich.

Aus diesen jeweiligen Vorteilen in verschiedenen Situationen kann nun eine hybride Form geschaffen werden, die alle Vorteile der einzelnen Algorithmen vereint. Da die einzelnen evolutionären Algorithmen durchaus miteinander vergleichbar sind, ist eine parallele Ausführung sinnvoll. Der parallele Ansatz bietet zudem die Möglichkeit, nicht nur die einzelnen erzeugten Individuen vergleichend zu bewerten und innerhalb eines jeden Algorithmus darauf zu reagieren, sondern auch in die Verteilung der Ressourcen für die einzelnen Algorithmen selbst einzugreifen. Grundlegend soll also ein System geschaffen werden, welches folgende Eigenschaften und Funktionen besitzt:

- Den Algorithmen die Anzahl der zu erzeugenden Individuen zuteilen
- Die erzeugten Individuen bewerten und die Anzahl der zu erzeugenden Individuen für die nächste Generation beeinflussen
- Die Zuteilung so kontrollieren, dass aktuell nicht erfolgreiche Algorithmen nicht vollständig aus der Berechnung entfernt werden um die Möglichkeit zu wahren, den Algorithmus zu einem späteren Zeitpunkt wieder erfolgreicher einzusetzen

Im Zuge der Modularität sollte ein solches System nicht auf bestimmte Algorithmen beschränkt sein, sondern vielmehr eine Schnittstelle bieten die es erlaubt, beliebige Algorithmen einzubinden. Die so erzeugte Plattform bietet damit die Chance, auch spezielle Probleme mit speziellen Algorithmen zu berechnen.

Die zu diesem Zweck entwickelte Lösung basiert auf dem Prinzip eines Initiativewertes. Im Bereich der rundenbasierenden Spiele wird dieses Prinzip genutzt, um die Aktionsreihenfolge von Spielfiguren zu bestimmen und kann durch verschiedene Ereignisse beeinflusst werden.

Im vorliegenden Fall soll der Initiativewert allerdings als Basis der Zuteilung von Individuen auf die Algorithmen dienen und vom Erfolg der Algorithmen in der Vergangenheit

abhängen. Dieses Prinzip könnte algorithmisch und mathematisch für die hier gewünschte Anwendung so umgesetzt werden:

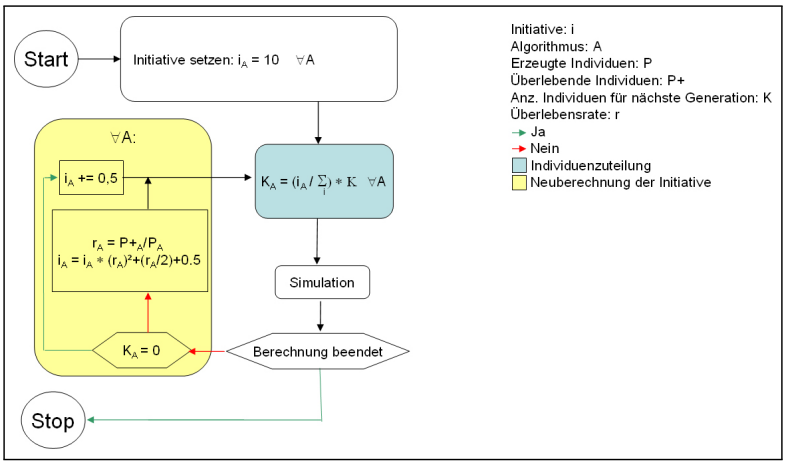

Abbildung 4.7 – Alg.: Initiative

Beim ersten Ausführen der Individuenzuteilung bewirkt die äquivalente Initiative eine gerundete Gleichverteilung der Individuen zu den erzeugenden Algorithmen.

Nach den Simulationen und solange die Berechnung nicht beendet ist, wird die Initiative pro Generation neu berechnet. Zunächst wird geprüft, ob der betreffende Algorithmus, basierend auf seinem Initiativewert, überhaupt für die letzte Generation Individuen erzeugen konnte. Ist dies nicht der Fall, wird ein Grundwert zur Initiative hinzugerechnet. Hat der Algorithmus im letzten Umlauf Individuen erzeugt, so wird die Initiative mit Hilfe der Überlebensrate r neu berechnet: Durch das Einsetzen von Testwerten wird die Funktionalität deutlich:

r_A	0	0,5	1
$i_A * (r_A)^2 + (r_A/2) + 0,5$	$i_A/2$	i_A	$2\ i_A$

Tabelle 4.1 – Beispielwerte der Initiativefunktion

Es wird also, basierend auf dem Erfolg des Algorithmus bei der Individuenauswahl der letzten Generation, die Initiative maximal verdoppelt oder minimal halbiert. Dies bewirkt, dass ein Algorithmus schnell mehr Individuen produzieren darf, sobald er erfolgreicher arbeitet als andere Algorithmen. Im Gegenzug reduziert sich die Zuteilung bei Misserfolg auf ein Mittelmaß mit derselben Geschwindigkeit. Falls kein Individuen eines Algorithmus nach der Selektion vorhanden sind, reduziert sich hingegen die Initiative asymptotisch gegen null. Um zu verhindern dass ein Algorithmus, falls er nach Zuteilung keine Individuen produziert, komplett aus der Berechnung ausscheidet, wird der dazugehörige Initiativewert pro Durchlauf leicht angehoben um kontinuierlich zu prüfen, ob der Algorithmus in einer späteren Situation erfolgreicher arbeitet.

4.6 Visualisierung und Protokollierung zur Laufzeit

Das Visualisieren stellt eine wichtige Komponente im Bezug auf Kontrolle und Entwicklung evolutionärer Algorithmen dar. Probleme können intuitiver lokalisiert werden und die Algorithmen einfacher auf ihre Funktionalität hin analysiert werden.

Im vorliegenden System ist eine derartige Visualisierung bereits vorhanden und es soll lediglich eine Anbindung geschaffen werden. Die Visualisierung beschränkt sich allerdings auf den Lösungsraum, der dort möglichen Markierung der Individuen des ersten Pareto-Rangs sowie der manuellen Einzeichnung von Nebenbedingungen und der Paretofront der optimalen Lösungen. Insgesamt stellt dies alle nötigen Hilfsmittel, auch in Form der Testfunktionen dar, um die grundlegende Funktionalität der Algorithmen zu überprüfen.

Zum Zweck der Analyse der einzelnen Vorgänge die die Individuen hervorbringen, sollte allerdings eine einfache, textbasierte Ausgabe implementiert werden mit welcher ein komplettes Berechnungsprotokoll abgespeichert werden kann.

4.7 Parallelisierung und Netzwerkkommunikation

Die zu Beginn angesprochene Beschleunigung der Berechnung durch Parallelisierung der einzelnen Aufgaben erfordert eine Kommunikation im Netzwerk die es nicht nur erlaubt, die konkreten Rechenvorgänge zu verteilen, sondern auch flexibel auf die zugrunde liegende, dynamische Struktur zu reagieren.

Als Einschränkung wird hier definiert, dass zunächst eine Parallelisierung im Client-Server-Betrieb umgesetzt wird. Das Programmieren eines verteilten Systems würde den Rahmen der Diplomarbeit sprengen, da diese Systeme im Zusammenhang mit evolutionären oder gar hybriden Optimierungen selbst ein komplexes Thema darstellen und eine eigene Arbeit fordern könnten. Dennoch bleibt festzuhalten, dass ein verteiltes System durchaus ein probates Mittel wäre, um flexibel auf Ausfälle von Hauptkomponenten zu reagieren.

Um ein Client-Server-System zu konstruieren ist es notwendig, zunächst einmal die funktionalen Anforderungen und die daraus abgeleiteten Anforderungen an die Kommunikationsebene zu definieren:

A. Funktionale Anforderungen:

A.1. Basis:

- Server erzeugt Individuen basieren auf den Algorithmen
- Server speichert die Individuen in die Arbeitstabelle einer Datenbank
- Clients lesen die Individuen
- Clients Simulieren die Individuen
- Clients speichern die Individuen in die Datenbank zurück

- Server liest die fertigen Individuen
- Server löscht die Individuen aus der Arbeitstabelle
- Server speichert die Individuen in einer Archivtabelle

A.2. Zuverlässigkeit:

- Ausfall eines oder mehrerer Clients muss erkannt werden können
- Auf den Ausfall eines oder mehrerer Clients muss reagiert werden können
- Fällt ein Client aus, muss die Zuteilung seiner Simulationen neu erfolgen
- Fällt ein Client während einer Simulation aus, muss das aktuell zu simulierende Individuum als noch nicht zugeteilt deklariert werden

A.3. Qualität:

- Die Zuteilung muss mit verschiedenen Client-Geschwindigkeiten umgehen können
- Die Zuteilung muss flexibel auf Geschwindigkeitsänderungen reagieren können
- Die neue Zuteilung muss spätestens erfolgen, sobald ein Client seine restlichen zugeteilten Simulationen beendet hat

Aus den Funktionalen Anforderungen ergeben sich nun die Anforderungen an die Kommunikation:

B. Anforderungen an die Kommunikation:

- Jedes Individuum muss einen jederzeit lesbaren Status besitzen
- Dieser Status muss von diesem Client selbst und vom Server beschrieben werden können
- Jedem Individuum muss ein Client zugeordnet werden können
- Diese Zuordnung muss vom Server revidierbar sein
- Jeder Client muss vom Server auf Funktionalität geprüft werden können
- Fälschliche Deklaration eines Clients als nicht verfügbar muss vom Client selbst revidierbar sein
- Jeder Client besitzt ein für den Server lesbares, vergleichbares Attribut um die Rechengeschwindigkeit ableiten zu können.
- Die Clients müssen den Status der Berechnung vom Server lesen können
- Ein Client muss sich selbst als Verfügbar für eine Berechnung anmelden können
- Ein Client darf nur dann an der Berechnung teilnehmen, wenn sichergestellt ist, dass die Aufgabenstellung für Server und Client gleich ist

Da theoretisch eine Parallelisierung nicht auf ein lokales Netzwerk oder ein Betriebssystem beschränkt sein soll, erfolgt die Kommunikation ausschließlich auf Basis der Daten-

bank. Somit entfallen zusätzliche Systeme die möglicherweise die Kommunikation minimal schneller machen aber an Grenzen, die durch verschiedene Betriebssysteme oder Netze aufgestellt werden, Probleme verursachen könnten. Die Auswirkungen auf die Dauer eines Kommunikationsvorgangs sind ebenfalls vernachlässigbar.

4.7.1 Datenbank

Als Datenbank wird die Open-Source-Lösung MySQL genutzt, da neben der freien Zugänglichkeit auch die Struktur des hier entwickelten Programms weder komplex noch optimierbar genug ist, um den Einsatz einer umfangreicheren, kommerziellen Variante zu rechtfertigen. Tendenziell ist für die hier vorliegende Anwendung einzig die Zugriffsgeschwindigkeit ein wichtiges Kriterium und in diesem Punkt stellt MySQL eine sehr gute Wahl dar [Horstmann, 2006]. Als Voruntersuchung wurde die Zugriffszeit auf eine MySQL-Datenbank über das Internet innerhalb eines VPN getestet. Die erhaltenen Werte von etwa einer Sekunde beim ersten und kaum erkennbaren Verzögerungen bei weiteren Zugriffen sind im Vergleich zur Simulationszeit eines Individuums von bis zu mehreren Minuten also vernachlässigbar.

Für die Zusammenarbeit zwischen Client und Server wird zunächst eine Datenbanktabelle erstellt, die Informationen über Clients und Server enthält. Diese Informationen sind für die Zuteilung und die Fehlererkennung wichtig.

Eine zweite Datenbanktabelle dient dem Transfer der Individuen zwischen Server und Clients. Hier speichert der Server initial die wichtigen Daten der Generation und welchem Client jedes einzelne Individuum zugeordnet ist. Anschließend lesen die Clients die Individuen ein, simuliert sie und speichern sie zurück in die Datenbank. Nachdem der Server diese Individuen wieder eingelesen hat, werden diese bei der Erzeugung der neuen Generation als Basis genutzt und aus der Tabelle gelöscht.

Zuletzt wird eine Datenbanktabelle benötigt, die die simulierten Individuen enthält. Sie dient als Ergebnisarchiv und reduziert damit die Zugriffszeiten auf die zweite Datenbanktabelle.

4.7.2 Scheduling

Unter Scheduling versteht man die Verteilung von Aufgaben auf verschiedene Ressourcen mit dem Ziel, die beteiligten Komponenten optimal zu nutzen und hier eine möglichst geringe Gesamtrechenzeit zu erreichen. Das Scheduling stellt also eine zentrale Komponente im Netzwerk dar, da es für die Zuverlässigkeit und Geschwindigkeit der parallelisierten Berechnung zuständig ist. Es werden folgende Arbeitsschritte durchgeführt:

Initialisierung des Schedulings:
Einmalig durch den Server:

- Wird ein Berechnungsvorgang vom Server gestartet, werden zunächst die Datenbanktabellen initialisiert
- Nun verdeutlicht der Server, dass sich die Clients für die Berechnung in die Datenbanktabelle registrieren können
- Nach einer manuellen Bestätigung, dass sich Clients eingetragen haben, beginnt das initiale Scheduling. Da noch keine Daten über die Rechenzeiten der Clients vorliegen, wird ein Standardwert für die Rechenzeit bei allen Clients angenommen. Daraus folgt zunächst eine Gleichverteilung der zu simulierenden Individuen.

Aufgabe des Clients:
Immer wenn ein passendes Individuum in der Datenbank entdeckt wurde:

- Wird ein Individuum entdeckt, welches dem Client zugeordnet ist, wird dieses eingelesen und simuliert.
- Die berechnete Dauer der Simulation wird in die Datenbank eingetragen.

Wiederholende Prüfung auf Gültigkeit des aktuellen Schedulings:
Nach der Initialisierung wird im Sekundentakt geprüft, ob durch Veränderungen oder Fehler das Scheduling erneut zu berechnen ist:

- Zu Beginn werden die Client-Daten aus der Datenbanktabelle neu ausgelesen.
- Danach wird das momentane Scheduling der noch nicht simulierten Individuen ausgelesen.
- Falls neue, fertig simulierte Individuen in der Datenbank auftauchen, werden diese gezeichnet.
- Hat ein Client innerhalb von 120% seiner angegebenen maximalen Berechnungszeit kein Ergebnis abgeliefert, wird er als ausgefallen deklariert und seine zugeteilten Individuen neu verteilt. Dies kommt zu Beginn der Berechnung häufiger vor und würde das Scheduling unbrauchbar machen falls der Client nicht die Möglichkeit hätte, seinen Fehlerstatus zu revidieren.
- Hat sich die Menge der Clients verändert, muss das Scheduling erneut angestoßen werden.
- Für den Fall dass alle Clients ausgefallen sind, muss das Scheduling eine Wartezeit anstoßen.

Neues Scheduling:
Wenn das aktuelle Scheduling als fehlerhaft gekennzeichnet wurde:

- Es wird unter Berücksichtigung der neuen Leistungsdaten der Clients hochgerechnet, wie viele Individuen einem Client für die Simulation zuzuordnen sind.
- Tritt eine Differenz zwischen Ist- und Sollwert auf, wird diese in der Zuteilung in der Datenbank korrigiert. (Dabei können auch neu hinzugekommene Clients in die Berechnung integriert werden.)

4.8 Erwartete Laufzeit

In den vergangenen Abschnitten wurde ein System skizziert, dessen Laufzeit bereits in einigen Punkten abgeschätzt werden kann. Zu diesen definierten Komponenten gehören:

- Erzeugen einer Generation
- Erzeugungsstrategie für Individuen
- Netzwerkkommunikation
- Simulation
- Erzeugen des neuen Genpools
- Prüfen auf Umschaltung zur lokalen Optimierung

Einige Voruntersuchungen haben bereits ergeben, dass die Simulationen in den vorhandenen Anwendungen den weitaus größten Teil der Rechenzeit in Anspruch nehmen. Im Hinblick auf die Skalierung der Software und die geplante Entwicklung einer Parallelisierung im Netzwerk sollten die restlichen Komponenten aber ebenfalls untersucht und gegebenenfalls optimiert werden.

- Erzeugen einer Generation:
  Hier werden die evolutionären Algorithmen eingesetzt die als aufwändigste Operation eine lineare Suche über den Genpool und/oder die Optimierungsparameter ausführen dürften.
- Erzeugungsstrategie für Individuen:
  Diese wird, global gesehen, einen entscheidenden Einfluss auf die Gesamtlaufzeit nehmen. Von der Komposition der Algorithmen wird die durchschnittliche Konvergenzgeschwindigkeit abhängen. Diese kann aber, da alle Algorithmen auf zufälliger Auswahl und der zufällig erzeugten, letzten Generation basieren, mit einer gewöhnlichen Laufzeitbeschreibung nicht erfasst werden.
- Netzwerkkommunikation:
  Die Netzwerkkommunikation wird in ihrer geplanten Konstruktion im Bezug auf die Laufzeit maßgeblich von der Anzahl der Individuen und der Anzahl der Clients abhängen.
- Simulation:
  Die Simulation ist wie oben beschrieben der aufwändigste Teil der Berechnung. Ihre Rechenzeit bleibt aber, soweit bisher untersucht, über die gesamte Berechnungsdauer jeweils konstant und kann somit als Vorfaktor eines Individuum für die Aufwandsberechnung gesehen werden.
- Erzeugen des neuen Genpools:
  Die Erzeugung eines Genpools wird mit Hilfe der Dominanzanalyse durchgeführt und stellt, soweit jetzt überblickbar, den aufwändigsten Teil in diesem Schritt dar.

Eine lineare Suche für jedes Individuum über alle restlichen Individuen mit Vergleich der Zielfunktionswerte ist Basis dieses Aufwands.

- Prüfen auf Umschaltung zur lokalen Optimierung:
  Dieser Algorithmus besitzt lediglich eine lineare Laufzeit über die Komponenten Individuum und Zielfunktionswerte.

Wie eingangs erwähnt wird die letztendliche Bestimmung der Laufzeit erst nach der Umsetzung möglich sein. Es stellt sich allerdings als wichtige Prämisse heraus, dass die Simulationszeit den entscheidenden Anteil der Laufzeit ausmacht und demzufolge die Parallelisierung im Netzwerk den größten Einfluss auf die Gesamtlaufzeit haben wird. Dennoch sollte die Anzahl der Individuen als zweiter Faktor nicht außer Acht gelassen werden - dieser Faktor wird quadratisch in den Gesamtaufwand eingehen.

4.9 Zusammenfassung

In den vorherigen Abschnitten wurde deutlich, dass der Entwurf des Systems nach modularen Gesichtspunkten aufgebaut sein muss, um eine Weiterentwicklung oder Anpassung an spezielle Aufgaben zu ermöglichen. Dabei wird die Grundstruktur hierarchisch nach Steuerungslevel gegliedert. Die oberste Ebene stellt der Controller dar, der die Initialisierung der restlichen Komponenten steuert und über die Verbindung mit den Klassen außerhalb des Projekts verfügt. Des Weiteren steuert der Controller Start und Ende der Optimierung und Initialisiert die Datenobjekte.

Der Algorithmusmanager führt alle Aktionen durch, die im Sinne der evolutionären Optimierung benötigt werden. Hierzu zählen die einzelnen Ausprägungen von Mutation, Rekombination und Selektion, aber auch die Bestimmung des Umschaltpunktes auf Basis des Solutionvolumes. Des Weiteren verwaltet der Algorithmusmanager die Individuenzuteilung im Bezug auf die parallel genutzten evolutionären Algorithmen oder die anschliessenden Gradientenverfahren.

Die zweite zentrale Komponente stellt der Networkmanager dar - dieser stellt die Verbindung mit den restlichen Rechnern im Netzwerk her und sorgt für den Datenaustausch. Weitere wichtige Funktionalitäten sind die Zuordnung der Individuen zu den Clients sowie das Berechnen dieser Zuordnung basierend auf einer dynamischen Struktur in Hinblick auf Geschwindigkeit und Verfügbarkeit der registrierten Clients. Als wichtige Komponente für den Server ist hierbei die Abbildung des Netzwerks selbst als Objekt zu sehen, da aufgrund der hier abgefragten Daten alle Kommunikation und Zuteilung bestimmt wird.

Die Parallelisierung der Rechenoperationen innerhalb eines Netzwerks stellt eine weitere Anforderung dar. Es wurde eine klassische Server-Client Struktur gewählt da diese Art der Umsetzung ohne Beeinflussung des Optimierungsergebnisses auskommt und zudem die intuitivste und einfachste Art darstellt, eine solche Berechnung zu verteilen. Als Kommunikationsgrundlage kommt eine MySQL-Datenbank zum Einsatz, die wegen ihrer schnellen Zugriffszeit und der freien Verfügbarkeit alle Kriterien erfüllt, die für diese Anwendung

definiert sind. Um die Kommunikation weiter zu vereinfachen und universeller Einzusetzen, wird diese komplett und indirekt über die Datenbank abgewickelt. Eine Erstellung eines separaten Kommunikationsnetzes ist daher nicht erforderlich und stellt demzufolge auch keine Hürde für ein unabhängiges Design bezüglich des Betriebssystems und verschiedener Netze dar.

Im Bereich der evolutionären Optimierung wurden mehrere Algorithmen als Basis der Berechnung ausgewählt - diese sind teilweise spezialisiert um Vorteile in den passenden Situationen zu erzielen. Hierbei sind besonders die Diversität aus Sortierung und der Dominanzvektor zu erwähnen die gezielt einen Geschwindigkeitszuwachs bei linearen Zusammenhängen bringen können.

Im Bereich der Gradientenverfahren wurde der Hook&Jeeves-Algorithmus ausgewählt um in Zusammenarbeit mit den evolutionären Algorithmen die optimale Konvergenzgeschwindigkeit zu erreichen. Es bleibt noch zu erwähnen, dass die Qualität der Ergebnisse des Hook&Jeeves Algorithmus durch einige Modifikationen gesteigert werden kann was zum Laufzeitvergleich allerdings nicht nötig sein wird und daher zu einen späteren Zeitpunkt umgesetzt werden kann.

Der wichtigste Schritt um die beiden Algorithmenklassen zu kombinieren, besteht aus der Analyse der in allen Schritten vorliegenden Individuen. Dabei wird klar, dass eine parallele Ausführung zumindest mit Hook&Jeeves ausgeschlossen werden muss. Der Grund hierfür liegt in der sprunghaften Entwicklung des Algorithmus in Bezug auf die Konvergenzgeschwindigkeit. Die Individuen sind daher nicht zu vergleichen mit den Individuen die die gleichmäßig agierenden evolutionären Algorithmen produzieren. Die Folge hieraus ist eine Kombination der beiden Optimierungsverfahren in sequentieller Art, wodurch die Vorteile der einzelnen Algorithmen potentiell eher der Konvergenzgeschwindigkeit zugute kommen.

Für den Übergang von der globalen, evolutionären Optimierung zur lokalen Optimierung mit Hilfe von Hook&Jeeves ist es nötig, den Umschaltpunkt zu definieren. Nach der Vorstellung und Diskussion einiger Indikatoren, deren Ergebnisse und Zusammenhänge die als Grundlage für die Umschaltung herangezogen werden, ist die Lösung ein Indikator, der sich von den im bisherigen Projekt genutzten Indikatoren unterscheidet. Der neue Indikator setzt globaler an als die vorhandenen um keine Fehler in spezielleren Problemen zu produzieren. Diese globalere Richtlinie wird mit nur spärlich verfügbaren Informationen erkauft die im Zusammenhang mit der evolutionären Optimierung zwar keine differenzierten Steuerungsmöglichkeiten zulassen, aber dennoch für die bloße Identifizierung des Umschaltpunktes ausreichen.

Die zweite Komponente im Rahmen der hybriden Nutzung stellt die Ressourcenverteilung innerhalb der evolutionären Optimierung auf deren Algorithmen dar. Prinzipiell soll ein erfolgreicher Algorithmus mehr Individuen produzieren als ein wenig erfolgreicher. Eine weitere Vorgabe hierbei ist, dass die weniger erfolgreichen Algorithmen nicht komplett aus der Berechnung ausscheiden dürfen da sie durchaus in einer späteren Situation gewinnbringend eingesetzt werden können. Das aus dem Prinzip der Initiative abgeleitete

Vorgehen setzt genau diese Vorgaben um und bewirkt eine maximal quadratische Erhöhung der Ressourcen für erfolgreiche Algorithmen und eine asymptotische Verringerung gegen null für nicht erfolgreiche Algorithmen.

5 Implementierung

Als sinnvolle Umsetzungsstrategie bei verteilten Systemen hat sich in der Vergangenheit ein zustandsbasiertes Vorgehen bewährt. Gerade im Hinblick auf Übersichtlichkeit und einfache Erweiterbarkeit ist diese Art der Systemmodellierung im Zusammenhang mit modularen Systemen eine sehr sinnvolle Möglichkeit.

In diesem Kapitel wird die Umsetzung detailliert beschrieben und auf verschiedene Probleme hingewiesen, die während Implementierungsphase aufgetreten sind.

Generell läuft die Berechnung einer Optimierung nach folgendem Schema ab:

Single PC:
So lange nicht die maximale Anzahl an Generationen erreicht wurde:

- Erzeugen der neuen Generation aus Genpool (Algomanager, Algos, Algofeedback)
- Simulation (Sim)
- Auswahl der Individuen, speichern in Genpool (Algomanager)
- Prüfen auf Umschaltung zur lokalen Optimierung (Algomanager, Solutionvolume)

Network Server:
So lange nicht die maximale Anzahl an Generationen erreicht wurde:

- Erzeugen der neuen Generation aus Genpool (Algomanager, Algos, Algofeedback)
- Netzwerkkommunikation schreiben (Networkmanager, Network)
- Simulation (Sim auf Clients)
- Netzwerkkommunikation lesen (Networkmanager, Network)
- Auswahl der Individuen, speichern in Genpool (Algomanager)
- Prüfen auf Umschaltung zur lokalen Optimierung (Algomanager, Solutionvolume)

Network Client:
So lange der Server eine weitere Generation erzeugt:

- Netzwerkkommunikation lesen (Networkmanager, Client)
- Simulation (Sim)
- Netzwerkkommunikation schreiben (Networkmanager, Client)

Als Datenbasis in der fertigen Implementierung dient der Genpool. Dieses Array aus Individuen wird in jeder Generation per Referenz an die meisten Hauptfunktionen übergeben und dient somit als 'Werkstück' welches bis zur Fertigstellung verschiedene Stationen durchläuft. Im Folgenden werden die erzeugten Klassen, deren Eigenschaften und Funktionen vorgestellt sowie die Integration in den bestehenden Kontext.

5.1 Struktur

Das vorhandene Projekt muss in einer Art restrukturiert werden, die das angesprochene Paradigma der Modularität umsetzt. Dabei ist es wichtig, die Schnittstellen zu vereinfachen und eine hierarchische Struktur aufzubauen, die ein Abstraktionslevel der Steuerung wiederspiegelt.

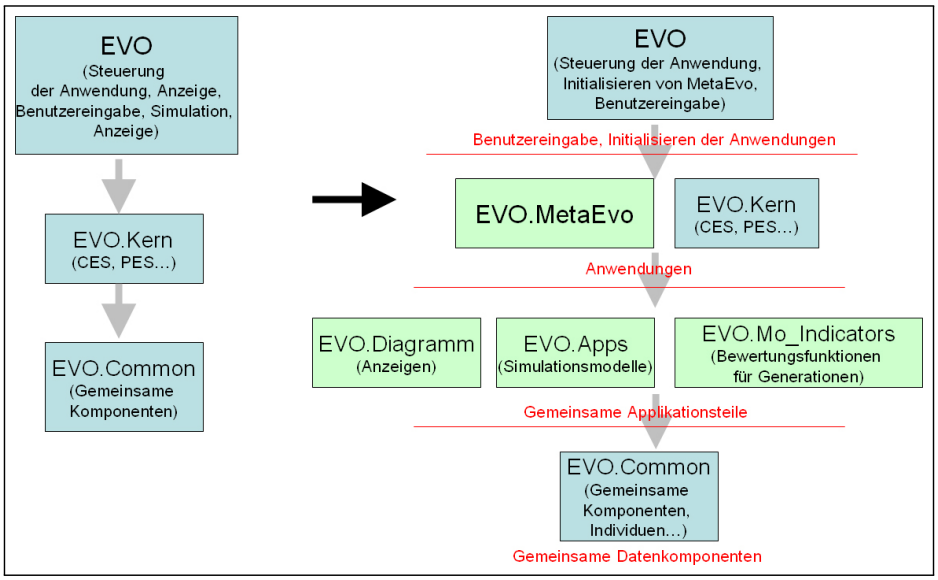

Abbildung 5.1 – Struktur von MetaEvo

5.2 Struktur: MetaEvo

Die im Rahmen dieser Diplomarbeit erzeugten Klassen sind mehrheitlich im Projekt EVO.MetaEvo zu finden. Dabei ist die Hierarchie wie rechts abgebildet umgesetzt. Diese Anordnung spiegelt die Steuerungslevel sowie die Abstraktionsebene wieder.

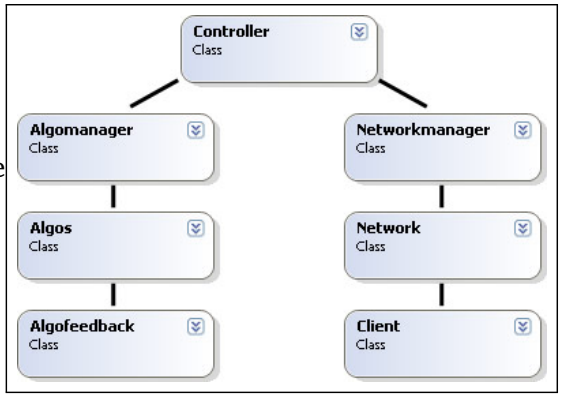

Abbildung 5.2 – Übersicht der Klassen

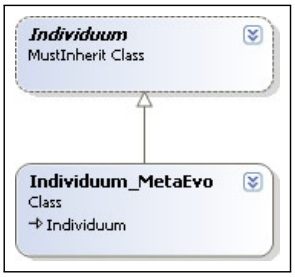

Abbildung 5.3 – MetaEvo Individuum

Das Basisdatenobjekt stellt EVO.Common.Individuum_MetaEvo dar. Dieses erbt die meisten Basiseigenschaften von EVO.Common.Individuum auf die die Funktionen wie z.B. das Zeichnen, die Simulation und die Testfunktionen zugreifen.

Als weitere Hilfskomponente existiert EVO.MO_Indicators.Solutionvolume, welches den Indikator für die Umschaltung zwischen globaler und lokaler Optimierung darstellt.

Abbildung 5.4 – Solutionvolume

Abbildung 5.5 – ApplicationLog

Das EVO.Diagramm.ApplicationLog stellt im Wesentlichen Funktionen zur Verfügung, die es ermöglicht, Daten zur erweiterten Analyse in ein Logbuch einzutragen und später als Textdatei zu speichern.

Eine weitere wichtige Komponente stellen die EVO.Common.EVO_Settings dar. Hier werden alle Informationen und Parameter gespeichert, die durch den Benutzer oder die zu berechnende Aufgabe definiert wurden. Da die Aufschlüsselung sehr umfangreich ist, wird im Folgenden jeweils nur auf EVO_Settings verwiesen.

5.3 Datenbank

Die Datenbank stellt im Rahmen der Netzwerkkommunikation die zentrale Datenablage und Informationsstelle dar. Wie bereits in den Ausführungen oben zu lesen ist, handelt es sich um eine MySQL-Datenbank. Es existieren mehrere Tabellen:

Metaevo_network:

Diese Tabelle dient dem Austausch von Informationen bezüglich der Zusammensetzung des Netzwerks und dem Status der Clients. Der Server berechnet nach dem Inhalt dieser Tabelle das Scheduling und die Clients erkennen den Berechnungsstatus der Aufgabe.

Bezeichnung	Typ	Funktion
ipName	varchar(15)	Name des Netzwerkrechners
type	varchar(20)	Server, Client
status	varchar(20)	In welchem Status sich der Rechner befindet
timestamp	timestamp	Letzte Bearbeitung eines Eintrags
speed_av	double	Durchschnittliche Rechenzeit
speed_low	double	Langsamste, bisher berechnete Rechenzeit

Tabelle 5.1 – Datenbanktabelle Metaevo_network

Metaevo_individuums:

Bezeichnung	Typ	Funktion
id	int	ID des Individuums
status	varchar(12)	raw, true, false
opt x	double	x Optimierungsparameter
pen y1	double	y1 Ergebnisse der zu minimierenden Funktion
const y2	double	Y2 Ergebnisse der Randbedingungsfunktionen
feat y3	double	Y3 Ergebnisse der Eigenschaftsfunktionen
ipName	varchar(15)	Name des zugeordneten Netzwerkrechners

Tabelle 5.2 – Datenbanktabelle Metaevo_individuums

Diese Tabelle stellt die Basis für den Datentransfer dar. Individuen werden vom Server bereitgestellt und von den Clients simuliert.

Metaevo_final_Individuums:
Die Struktur dieser Tabelle ist ähnlich zu 'Metaevo_Individuums' nur fehlt das Feld ipName. Da diese Tabelle als finaler Datenspeicher dient, ist diese Information, die die Zuordnung des Individuums zu einem Client wiederspiegelt, aber unerheblich.

Metaevo_infos:

Bezeichnung	Typ	Funktion
id	int	ID des Eintrags
wert	varchar(50)	Beliebiger Wert

Tabelle 5.3 – Datenbanktabelle Metaevo_infos

Die Tabelle metaevo_infos dient dem Informationsaustausch bezüglich der Aufgabe und als Information für die browserbasierte Berechnungsübersicht.

5.4 EVO.Common.Individuum_MetaEVO

Die Klasse EVO.Common.Individuum von der Individuum_MetaEvo erbt, stellt das Basisobjekt dar. Es vererbt neben Eigenschaften und Funktionen auch die Zuordnung zu einem Problem welches in der Basisklasse mit der Funktion
`Public Shared Sub Initialise(ByRef prob As Problem)` gesetzt wird.

Initialisierung:

```
Public Sub New(ByVal type As String, ByVal id As Integer,
ByVal numberOptparas_input As Integer)
```
Übergibt eine freie Typbezeichnung, eine eindeutige id sowie einen Satz Optimierungsparameter.

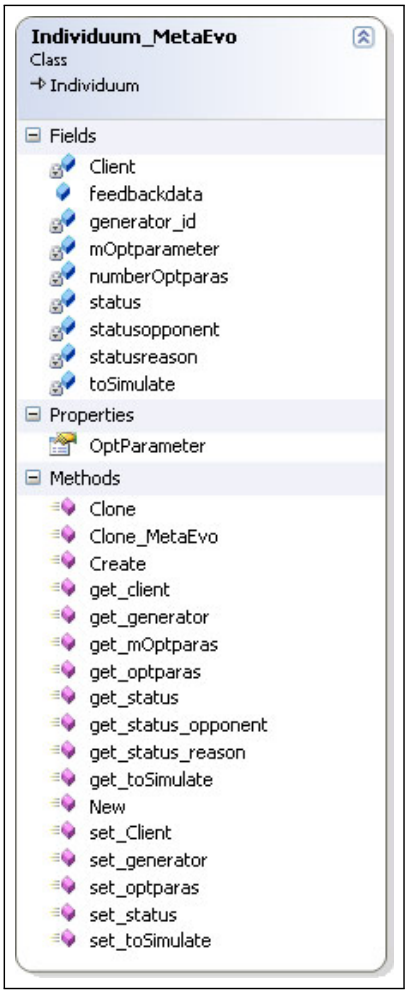

Abbildung 5.6 – Klasse Individuum_MetaEvo

Eigenschaften:

`double[] Feedbackdata`
Enthält von Algorithmen genutzte, spezifische Daten um Zusatzinformationen über mehrere Generationen weiterzugeben.

Wichtige öffentliche, geerbte Eigenschaften:

`int ID`
Die ID des Individuums.

`double[] Penalties`
Die Zielfunktionswerte des Individuums.

`double[] Constraints`
Die Randbedingungen des Individuums (negative Werte bedeuten eine Verletzung einer Randbedingung).

`double[] Features`
Eigenschaften des Individuums die nicht für die Optimierung relevant sind.

`bool Is_Feasible`
Ist wahr, falls eine Nebenbedingung verletzt wurde.

Methoden:

`public Individuum_MetaEvo Clone_MetaEvo( )`
Dupliziert ein MetaEvo-Individuum.

`public string get_client( )`
`public void set_client(string)`
Gibt den Rechnernamen des Clients zurück dem die Simulation dieses Individuums zugeteilt wurde.

`public int get_generator( )`
`public void set_generator(int)`
Gibt den Array-Index des erzeugenden Algorithmus zurück.

`public OptParameter( )get_mOptparas( )`
Gibt ein Array von Optimierungsparametern mit Zusatzinformationen zurück. Diese Optparameter-Objekte enthalten jeweils mehrere Eigenschaften:

- Minimaler Wert
- Maximaler Wert
- Bezeichnung und Einheit
- Skalierter Wert (zwischen 0 und 1)
- Startwert

`public double( )get_optparas( )`
`public void set_optparas(double())`
In den meisten Fällen genügt diese Funktion und es muss nicht get_mOptparas( ) angewendet werden. Sie gibt die reellen Werte der Optimierungsparameter in einem Doublearray zurück.

`public string get_status( )`
`public void set_status(string(opt: string2(opt: string3)))`
Gibt den Status eines Individuums zurück:

- Raw: Noch nicht simuliert
- Calculate: Wird gerade simuliert
- True: Simuliert und gültig
- False: Simuliert und ungültig

Dient in der Datenbank als Eigenschaft des Individuums. Die optionalen Eingabeparameter für set_status, string2 und string3 (jeweils mit # als Trennzeichen), stellen die Eigenschaften status_reason und status_opponent dar.

`public int get_status_opponent( )`
Gibt die ID des Individuums zurück, welches zur Selektion dieses Individuums geführt hat.

`public string get_status_reason( )`
Gibt den Grund der Selektion an:

- Dominated: Das Individuum wurde von einem anderen Individuum dominiert
- Crowding: Das Individuum wurde wegen zu hohem Aufkommen an Individuen in einem Bereich gelöscht

`public bool get_toSimulate( )`
`public void set_toSimulate(bool)`
Gibt an, ob das Individuum simuliert werden soll. (Dient im Bereich der lokalen Optimierung als Eigenschaft.)

5.5 EVO.MetaEvo.Controller

Die Klasse Controller dient der Steuerung der Optimierung auf höchster Ebene. Hier werden alle benötigten Klassen Initialisiert, die Berechnung gestartet und beendet.

Initialisierung:

```
public Controller(
ref EVO.Common.Problem prob_input,
ref EVO.Common.EVO_Settings settings_input,
ref EVO.Diagramm.Hauptdiagramm hauptdiagramm_input,
ref EVO.Common.Progress progress1,
ref EVO.Apps.Testprobleme testprobleme_input)
```

Übergibt das Problem, die Benutzereinstellungen 'settings', das hauptdiagramm, den grafischen Fortschrittsbalken 'progress' sowie das Testproblem. Eine weitere Möglichkeit der Initialisierung übergibt statt dem Testproblem das Simulationsobjekt.

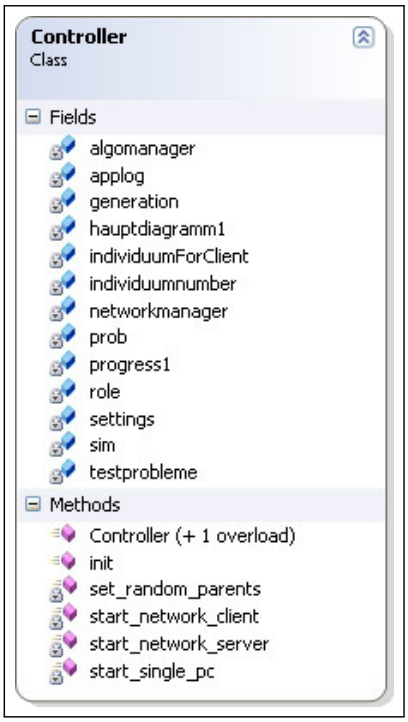

Abbildung 5.7 – Klasse MetaEvo Controller

Methoden:

```
public void init(
ref EVO.Common.Problem prob_input,
ref EVO.Common.EVO_Settings settings_input,
ref EVO.Diagramm.Hauptdiagramm hauptdiagramm_input
, ref EVO.Common.Progress progress1)
```

Startet den Berechnungsvorgang nachdem in der Initialisierung bereits die Unterscheidung zwischen Simulation und Testfunktionen getroffen wurde. Anhand von EVO_Settings wird entscheiden, welcher Berechnungsmodus (Single PC, Network Server, Network Client) gestartet wird.

```
private bool set_random_parents(
EVO.Common.Individuum_MetaEvo[]
generation_input)
```

Setzt die Optimierungsparameter jedes Individuums im Array auf zufällige Werte innerhalb der vorgegebenen Grenzen.

```
private void start_network_client()
```

Startet die Berechnung als Netzwerk-Client. Der Client wird auf dem in EVO_Settings angegebenen Server registriert und wartet auf Individuenzuteilungen in der DB um diese zu Simulieren und zurückzuspeichern. Die Funktion Endet sobald der Status des Servers in der DB 'finished' annimmt.

`private void start_network_server()`
Startet die Berechnung als Netzwerk-Server. Es werden zufällige Eltern gesetzt, danach wird wiederholt ausgeführt bis die in EVO_Settings definierte Anzahl an Generationen erreicht wurde oder die vorher in oder die lokale Optimierung die Minimalschrittweite unterschritten wurde:

- neue Individuen erzeugen (Klasse Algomanager)
- die Individuen im Netzwerk zuteilen (Klasse Networkmanager)
- Simulierten Individuen wieder einlesen (Klasse Networkmanager)
- Selektion der Individuen (Klasse Algomanager)
- Zeichnen der Individuen (TeeChart)

`private void start_single_pc()`
Hier werden die gleichen Vorgänge wie im Berechnungsmodus 'network-server' durchgeführt bis auf den Unterschied, dass die Simulation nicht ins Netzwerk ausgelagert wird.

5.6 EVO.MetaEvo.Algomanager

Der Algomanager dient als zentrale Komponente der Steuerung für die Individuenerzeugung sowie der Steuerung der Selektionsschritte. Des Weiteren werden nach der Entscheidung von EVO.MO_Indicators.Solutionvolume die nötigen Schritte für die Umschaltung zur lokalen Optimierung durchgeführt.

Initialisierung:

```
public Algomanager(
ref EVO.Common.Problem prob_input, ref EVO.Common.EVO_Settings
settings_input, int individuumnumber_input,
ref EVO.Diagramm.ApplicationLog applog_input,
ref EVO.Diagramm.Hauptdiagramm hauptdiagramm_input)
```

Ausser den bekannten Übergabeparametern wird individuumnumber übergeben - der Startindex für die IDs der neu zu erzeugenden Individuen.

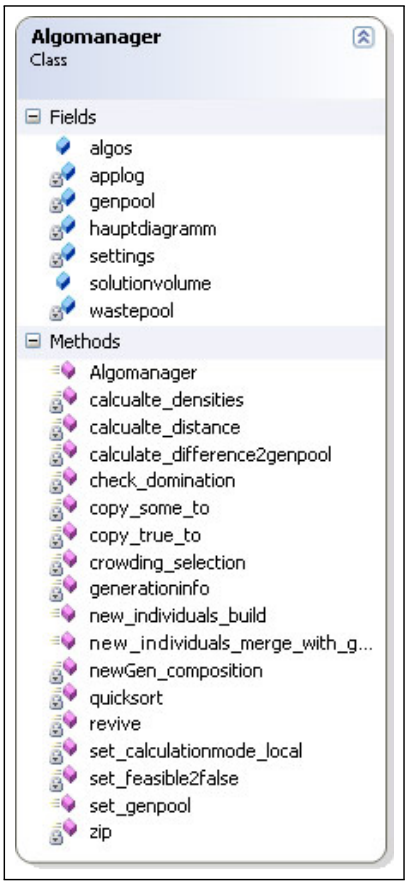

Abbildung 5.8 – Klasse MetaEvo Algomanager

Eigenschaften:

```
algos[] Algos
```

Ermöglicht den Zugriff auf spezifische Algorithmus-Daten für diverse ApplicationLog-Einträge und Statusmeldungen der Berechnung.

```
EVO.MO_Indicators.Solutionvolume solutionvolume1
```

Ermöglicht die Abfrage des letzten Solutionvolumes durch den Controller.

Methoden:

```
private double[] calcualte_densities(ref
EVO.Common.Individuum_MetaEvo[] work_input)
```

Berechnet die Dichte der Individuen im Lösungsraum.

```
private double calcualte_distance(
double[] input1, double[] input2)
```

Berechnet den euklidischen Abstand zweier Individuen im Lösungsraum.

```
private int calculate_difference2genpool(ref
EVO.Common.Individuum_MetaEvo[] genpool_input,
 ref EVO.Common.Individuum_MetaEvo[] input2)
```

Berechnet die Differenz der Anzahl der gültigen Individuen von input2 zur Genpool-Grösse.

```
private void check_domination(
ref EVO.Common.Individuum_MetaEvo[] genpool_input
, ref EVO.Common.Individuum_MetaEvo[] input2)
```
Prüft die Individuen aus genpool_input und input2 auf gegenseitige Dominanz. Falls genpool_input und input2 gleich sind, wird innerhalb der Eingabe geprüft. Beim Auffinden einer Dominanz wird das dominierte Individuum in den Zustand 'false' gesetzt, statusreason auf 'dominated' gesetzt sowie das dominante Individuum vermerkt.

```
private void copy_some_to(ref EVO.Common.Individuum_MetaEvo[]
newgen_input, ref EVO.Common.Individuum_MetaEvo[] newgen_output)
```
Kopiert alle Individuen aus newgen_input in newgen_output.

```
private void copy_true_to(ref EVO.Common.Individuum_MetaEvo[]
genpool_input, ref EVO.Common.Individuum_MetaEvo[] genpool_output)
```
Kopiert Individuen aus genpool_input mit dem Status 'true' in genpool_output.

```
private void crowding_selection(int killindividuums_input,
ref EVO.Common.Individuum_MetaEvo[] genpool_input,
ref EVO.Common.Individuum_MetaEvo[] input2)
```
Setzt den Status von Individuen basieren auf der Bevölkerungsdichte im Lösungsraum auf 'false' und statusreason auf 'crowding'. Die Anzahl von killindividuums_input bestimmt die Menge. Es wird unabhängig von der Zugehörigkeit zu einem Eingabearray entscheiden welche Individuen ausscheiden.

```
private string generationinfo(ref EVO.Common.Individuum_MetaEvo[]
generation)
```
Liest Informationen über das vorliegende Individuenarray. Es werden der Erzeugeralgorithmus, die Optimierungsparameter, die Randbedingungen sowie die Zielfunktionswerte als string zurückgegeben.

```
public void new_individuals_build(
ref EVO.Common.Individuum_MetaEvo[] new_generation_input)
```
Ruft die Klasse Algos auf und veranlasst die Erzeugung neuer Individuen im Array mit Hilfe des hinterlegten Genpools.

```
public void new_individuals_merge_with_genpool(
ref EVO.Common.Individuum_MetaEvo[] new_generation_input)
```
Diese zentrale Methode führt die Selektion mit Hilfe der meisten hier beschriebenen Methoden aus. Als Vergleichsbasis dient auch hier der hinterlegte Genpool.

```
private void newGen_composition(
ref EVO.Common.Individuum_MetaEvo[] new_generation_input)
```
Diese Komponente verteilt, basieren auf den Ergebnissen der letzten Generationen, die Anzahl der zu erzeugenden Individuen auf die einzelnen Algorithmen.

```
private void quicksort(ref EVO.Common.Individuum_MetaEvo[] input,
int kriterium, int low_input, int high_input)
```
Quicksort wird eingesetzt, um einen Laufzeitvorteil beim Dominanzcheck zu erreichen indem die MetaEvo-Arrays vorher sortiert werden. (Siehe Kapitel **??**

```
private void revive(int numberawake_input,
ref EVO.Common.Individuum_MetaEvo[] genpool_input,
ref EVO.Common.Individuum_MetaEvo[] input2)
```
Die Methode revive setzt, die Anzahl angegeben durch numberawake_input, Individuen mit dem Status false wieder auf true. Dabei werden die beiden Eingabe-arrays zusammengefasst und so die Individuen völlig gleich behandelt.

```
private void set_calculationmode_local(ref EVO.Common.Individuum_MetaEvo[]
 genpool_input, ref EVO.Common.Individuum_MetaEvo[] new_generation_input)
```
Steuert Teile der Umstellung auf die lokale Optimierung. Dies bedeutet das Vorbereiten der Individuen-Arrays sowie das Setzen des neuen Berechnungsalgorithmus.

```
private void set_feasible2false(ref EVO.Common.Individuum_MetaEvo[] input
, ref EVO.Common.Individuum_MetaEvo[] input2)
```
Setzt den Status von Individuen auf 'false' falls eine Nebenbedingung verletzt wurde.

```
public void set_genpool(ref EVO.Common.Individuum_MetaEvo[] genpool_input)
```
Diese Methode hinterlegt den Genpool als Basis für die Selektion un die Generierung neuer Individuen.

```
private void zip(ref EVO.Common.Individuum_MetaEvo[] genpool_input,
ref EVO.Common.Individuum_MetaEvo[] input2)
```
Diese Methode wird, nach der Anwendung der Selektionsprozesse, auf die beiden Arrays aus genpool_input und input2 angewendet. Es werden die verbleibenden, als Eltern für die neue Generation ausgewählten Individuen in den genpool_input verschoben.

5.7 EVO.MetaEvo.Algos

Diese Klasse hat die Aufgabe, neue Individuen zu produzieren. Dies geschieht auf Basis der von Evo.MetaEvo.Algomanager.newGen_composition berechneten Verteilung sowie dem hinterlegten Genpool und Wastepool. Die Klasse kommt bei globaler und lokaler Optimierung zur Anwendung da sie alle Algorithmen enthält.

Initialisierung:

```
public Algos(ref EVO.Common.EVO_Settings settings_input,
int individuum_id_input, ref EVO.Diagramm.ApplicationLog applog_input)
```

Auch an Algos wird der Startindex der Individuen-ID weitergegeben und nach jedem neu erzeugten Individuum erhöht.

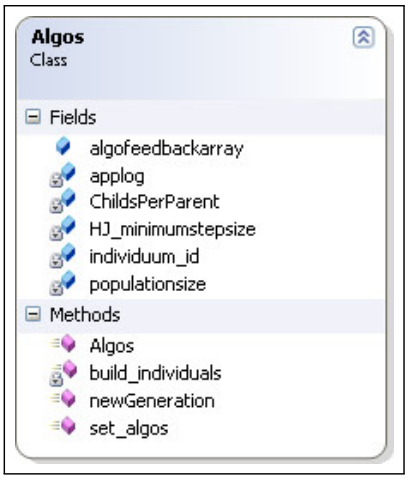

Abbildung 5.9 – Klasse MetaEvo Algos

Eigenschaften:

```
Algofeedback[] algofeedbackarray
```

Beinhaltet die von Evo.MetaEvo.Algomanager berechnete Verteilung der zu erzeugenden Individuen auf die Algorithmen. Der Index der Einträge im Arrays wird als algo_id genutzt.

Methoden:

```
private void build_individuals(
ref EVO.Common.Individuum_MetaEvo[]
genpool_input,
ref EVO.Common.Individuum_MetaEvo[]
new_generation_input,
ref EVO.Common.Individuum_MetaEvo[]
wastepool_input, int algo_id, int startindex_input)
```

Erzeugt eine vorgegebene Anzahl an Indviduen durch den Algorithmus mit der passenden algo_id. Die neu erzeugten Individuen werden im Array new_generation_input, ab dem gegebenen startindex, gespeichert. Für die meisten Algorithmen wird dabei der Genpool als Berechnungsgrundlage benötigt, für andere zusätzlich der Wastepool.

```
public void newGeneration(ref EVO.Common.Individuum_MetaEvo[]
genpool_input, ref EVO.Common.Individuum_MetaEvo[] new_generation_input,
ref EVO.Common.Individuum_MetaEvo[] wastepool_input)
```

Diese Methode stellt die Verbindung nach aussen dar und nutzt die build_individuals-Methode. Es wird anhand der zu erzeugenden Individuen der Startindex eines jeden Algorithmus bestimmt und die Erzeugung jeweils gestartet.

`public void set_algos(string algos2use_input)`
Diese Methode dient der Vorbereitung der Klasse und definiert durch einen einfachen String die Algorithmen die genutzt werden. Auf Basis dieser Algorithmen wird das Algofeedbackarray erzeugt in dem die Zusatzinformationen zum jeweiligen Algorithmus gespeichert werden.

5.8 EVO.MetaEvo.Algofeedback

Diese Klasse stellt im Wesentlichen eine Speicherstruktur dar. Die Initialisierungsmethode setzt daher lediglich einige Standardwerte.

Initialisierung:

```
public Algofeedback(string name_input,
int number_individuals_for_nextGen_input)
```

In der Initialisierung wird zunächst für alle Algorithmen der Initiative-Wert auf 10 gesetze.

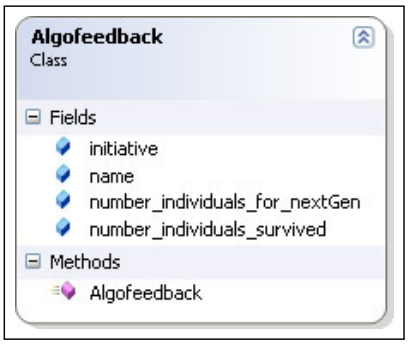

Abbildung 5.10 – Klasse MetaEvo Algofeedback

Eigenschaften:

```
double initiative
```

4.5.3 Dieser Wert dient, im Vergleich mit den Werten aller Algorithmen, der Bestimmung der Anzahl der zu erzeugenden Individuen für die nächste Generation.

```
string name
```

Hier wird der Name des Algorithmus gespeichert

```
int number_individuals_for_nextGen
```

Gibt die Anzahl der zu erzeugenden Individuen in der nächsten Generation an.

```
int number_individuals_survived
```

Anzahl der nach der Selektion übriggebliebenen, neu erzeugten Individuen: Dient der Berechnung des Umschaltpunktes zur lokalen Optimierung sowie als Grundlage für die Anpassung des Initiative-Wertes für die Berechnung der Individuen-Verteilung der nächste Generation.

5.9 EVO.MetaEvo.Networkmanager

Der Networkmanager dient als Schnittstelle zwischen Server und Client dem Austausch aller Daten die zur Bearbeitung der Aufgabe im Netzwerk benötigt werden. Die Informationen können in ihrer Zugehörigkeit zwischen der Client-Tabelle und der Individuen-Tabelle getrennt werden.

Initialisierung:

```
public Networkmanager(ref EVO.Common.Individuum_MetaEvo individuum_input,
ref EVO.Common.EVO_Settings settings_input, ref EVO.Common.Problem
prob_input, ref EVO.Diagramm.ApplicationLog applog_input)
```

Der Networkmanager benötigt ein Individuum um nach dessen Datenmuster die Datenbanktabellen zu erstellen.

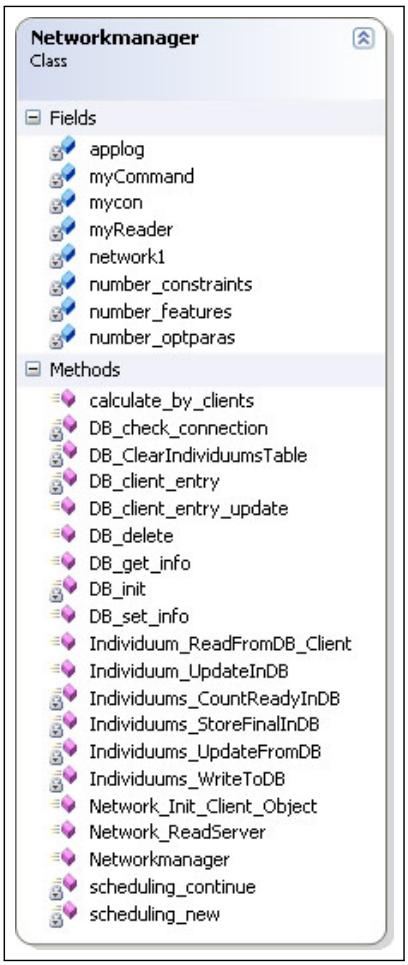

Abbildung 5.11 – Klasse MetaEvo Networkmanager

Methoden:

```
public bool calculate_by_clients(ref EVO.Common
.Individuum_MetaEvo[] generation_input, ref EVO
.Diagramm.Hauptdiagramm hauptdiagramm_input)
```

Diese Methode stellt die Hauptmethode dar. Von aussen werden die neuen Individuen übergeben sowie das Hauptdiagramm in das sie nach der Simulation eingezeichnet werden. Das Hauptdiagramm wird hier übergeben um unmittelbar nach der Simulation eines Individuums dieses zu zeichnen. Diese Klasse nutzt, vergleichbar mit dem Aufbau des Algomanagers, die meisten der hier vorhandenen, privaten Methoden.

```
private bool DB_check_connection()
```

Diese Methode prüft die vom benutzer eingegebene Verbindung zur Datenbank und gibt das Ergebnis als Bool zurück.

```
private void DB_ClearIndividuumsTable()
```

Löscht den Inhalt der Individuen-Tabelle in der Datenbank um sie so für die Speicherung neuer Individuen vorzubereiten.

```
private void DB_client_entry()
```

Diese Methode trägt den ausführenden PC als zur Verfügung stehenden Client in die Datenbank ein.

```
public void DB_client_entry_update()
```
Prüft, ob ein Client-Entry vorliegt und aktualisiert diesen Eintrag. Ist kein Eintrag vorhanden, wird DB_client_entry() aufgerufen.

```
public void DB_delete(ref EVO.Common.EVO_Settings settings_input)
```
Löscht die Datenbank. Dieser Befehl wird bisher nicht verwendet - nach Ausführung des Programms stehen also alle Individuen solange zur Verfügung bis das Programm erneut gestartet wird oder die Datenbank manuell gelöscht wird.

```
private void DB_init(ref EVO.Common.EVO_Settings settings_input)
```
Erzeugt die Tabelle für die Clients sowie die beiden Tabellen für die Individuen in der Datenbank.

```
public void Individuum_ReadFromDB_Client(ref EVO.Common.Individuum_MetaEvo
 individuum_input)
```
Liest das nächste Individuum von der Datenbank ein. Es muss dabei dem ausführenden PC zugeordnet sein. Die einzig wichtigen Daten sind dabei die Optimierungsparameter als Grundlage der Simulation und der Berechnung für die Einhaltung der Randbedingungen, sowie die ID des Individuums.

```
public void Individuum_UpdateInDB(ref EVO.Common.Individuum_MetaEvo
individuum_input, string fields2update, string status_input)
```
Mit dieser Funktion können mehrere Parameter eines Individuums in der Datenbank geändert werden. Wichtig ist dabei der string 'fields2update' welcher mit 'status', 'opt', 'feat', 'const' und 'ipname' jeweils die Felder selektieren kann, die verändert werden sollen.

```
private int Individuums_CountReadyInDB()
```
Zählt die bereits simulierten und zurückgespeicherten Individuen in der Datenbank.

```
private void Individuums_StoreFinalInDB(
ref EVO.Common.Individuum_MetaEvo[] generation_input)
```
Speichert fertig simulierte Individuen in der 'metaevo_final_individuums' Tabelle um sie zu archivieren.

```
private void Individuums_UpdateFromDB(ref EVO.Common.Individuum_MetaEvo[]
generation_input)
```
Alle lokal in 'generation_input' vorhandenen Individuen werden durch die in der Datenbank vorhandenen Individuen mit gleicher ID überschrieben.

```
private void Individuums_WriteToDB(ref EVO.Common.Individuum_MetaEvo[]
generation_input)
```
Die in generation_input vorhandenen Individuen werden in die Datenbanktabelle metaevo_individuums geschrieben.

`public Client Network_Init_Client_Object(string ipName_input)`
Erstellt ein Objekt vom Typ Client zur einfachen Verwaltung von Statusinformationen des Servers selbst.

`public string[] Network_ReadServer()`
Liest den Status und dessen Aktivierungszeit des Server-Eintrags in der Datenbank.

`private void scheduling_continue(ref EVO.Common.Individuum_MetaEvo[] generation_input)`
Muss innerhalb eines laufenden Berechnungsvorgangs ein Scheduling (**??**) neu berechnet werden gilt es folgendes zu tun:

- Prüfen ob sich die Zusammensetzung der Clients in der Datenbank geändert hat
- Falls keine Clients mehr vorhanden, warten
- Defekte Clients identifizieren und deren zugeordnete Individuen wieder freigeben
- Zuordnungen der Individuen zu den Clients neu bestimmen
- Benötigte Änderungen in der Datenbank umsetzen

`private void scheduling_new(ref EVO.Common.Individuum_MetaEvo[] generation_input)`
Beim Erstellen eines neuen Schedulings wird zunächste die aktuelle Client-Liste aus der Datenbank geladen und die eventuell vorhandenen Individuenzuordnungen auf Client-Seite gelöscht. Danach werden die Individuen gleichmässig verteilt.

5.10 EVO.MetaEvo.Network

Die Klasse Network dient als Abbild der Client-Tabelle in der Datenbank dem Server dazu, die Clients zu verwalten und somit die Lese- und Schreibzugriffe in der Datenbank zu minimieren.

Initialisierung:

```
public Network(ref MySqlConnection mycon_input)
```
Benötigt lediglich die MySQL-Verbindung

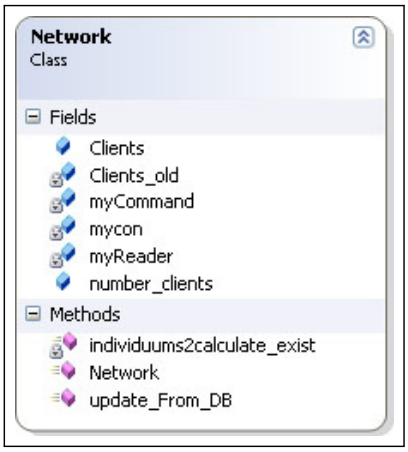

Abbildung 5.12 – Klasse MetaEvo Network

Eigenschaften:

```
Client[] Clients
```
Zusammenfassung der einzelnen Client-Daten in einem Array

```
int number_clients
```
Gibt die Anzahl der Clients an

Methoden:

```
public bool update_From_DB()
```
Liest die Client-Tabelle neu ein und entscheided aufgrund von Veränderungen ob ein neues Scheduling berechnet werden muss. Dies kann aufgrund forlgender Ereignisse geschehen:

- Anzahl der Clients ändert sich
- maximale Rechenzeit eines Clients ist um 50% überschritten
- Ein Client hat keine Individuen mehr zu berechnen aber es liegen noch 'raw'-Individuen vor

5.11 EVO.MetaEvo.Client

Diese Klasse stellt im Wesentlichen, ähnlich wie EVO.MetaEvo.Algos, einen Datenspeicher dar. Sie wird genutzt, um einen Client aus der Datenbank-Tabelle der Clients im Programm darzustellen und so komfortabler zu verwalten.

Initialisierung:

```
public Client(ref MySqlConnection mycon_input, string ipName_input,
string status_input, DateTime timestamp_input, double speed_av_input,
double speed_low_input, int numberindividuums_input)
```

Die Initialisierung erfolgt entweder mit diesen Parametern oder mit keinem für die Selbstverwaltung der Clients.

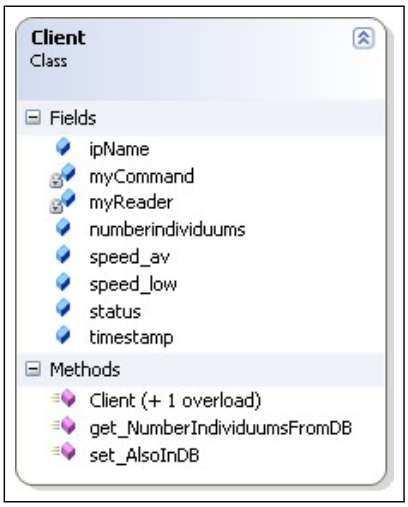

Abbildung 5.13 – Klasse MetaEvo Client

Eigenschaften:

`string ipName`
Beinhaltet den Netzwerknamen eines Clients.

`int numberindividuums`
Gibt die Anzahl der zu simulierenden Individuen an die dem Client zugeordnet sind.

`double speed_av`
Gibt die durchschnittliche Simulationszeit in Millisekunden an.

`double speed_low`
Gibt die bisher maximale Simulationszeit in Millisekunden an. Dies wird genutzt um den Ausfall eines Clients zu detektieren.

`string status`

- Client Status: {ready, claculating, error}
- Server Status: {init Genpool, generate Individuums, waiting for client-calculation, select Individuums, finished}

Gibt den Berechnungsstatus an. Beispielsweise ist für Clients so zu erkennen, ob eine Simulationsaufgabe vom Server als abgeschlossen gekennzeichnet wurde.

`DateTime timestamp`
Definiert die Zeit zu der die letzte Änderung des Client-Eintrags in der Datenbank durchgeführt wurde.

Methoden:

`public void get_NumberIndividuumsFromDB()`
Gibt die Anzahl der Individuen aus der Datenbank zurück die dem Client zugeordnet sind.

`public void set_AlsoInDB(string status_input, double speed_av_input, double speed_low_input)`
Ändert eine Eigenschaft eines Clients im Objekt, aber auch in der Datenbank. Um für eine der Variablen keine Änderung durchzuführen, wird der Status leer gelassen oder für die Geschwindigkeitsangaben auf -1 gesetzt.

5.12 Hilfsprogramme und Funktionen

5.12.1 ApplicationLOG

Die Komponente ApplicationLog kann von einer Eigenschaft der Evo_Settings freigeschaltet werden und besitzt im Wesentlichen nur drei Funktionen:

Initialisierung:

```
Public Sub New(ByRef settings_input As EVO.Common.EVO_Settings)
```

Die Settings geben an, ob Applicationlog genutzt wird.

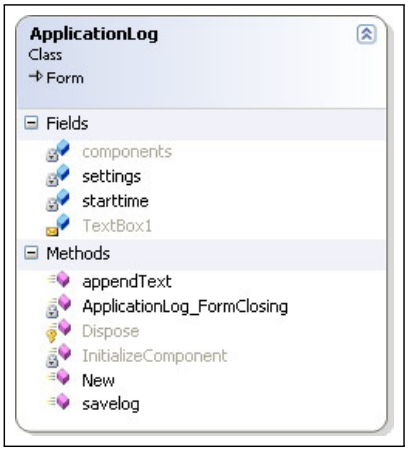

Abbildung 5.14 – Klasse ApplicationLog

Methoden:

```
public void appendText( )
```

Fügt eine Textzeile dem Logbuch hinzu

```
public void appendResult( )
```

Fügt den Inhalt eines Arrays als Text an das Logbuch an

```
public void savelog( )
```

Speichert das Logbuch in einer beliebigen Textdatei

5.12.2 Solutionvolume

Diese Klasse dient als Indikator zum Finden des Umschaltpunktes. Eine weitere Funktionalität macht es möglich, diese Entscheidung auch gleich zu treffen.

Initialisierung:

`public Solutionvolume(int historylength_input, double minimumchange_input, ref EVO.Diagramm.ApplicationLog applog_input)` Historylength definiert die Anzahl der Werte die zum Vergleich herangezogen werden. Diese Werte bestehen jeweils aus der prozentualen Veränderung des Solutionvolumes zwischen zwei Generationen. Minimumchange definiert dabei die untere Grenze der prozentualen Veränderung die summiert zwischen allen betrachteten, aufeinanderfolgenden Generationen erzeugt worden ist.

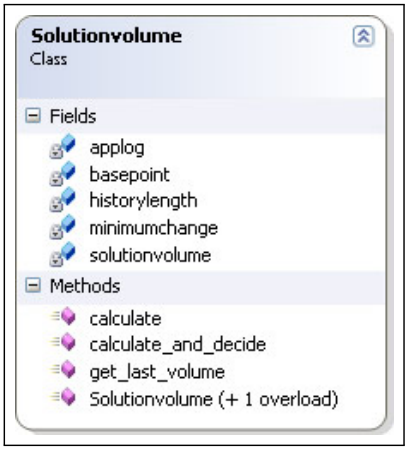

Abbildung 5.15 – Klasse Solutionvolume

`public Solutionvolume(int historylength_input)`

Soll nur die Methode 'calculate' genutzt werden, genügt diese Initialisierung.

Methoden:

`public double[] calculate(ref EVO.Common.Individuum_MetaEvo[] generation)`
Berechnet das neuste Solutionvolume und speichert es in solutionvolume[0]. Alle anderen, vorherigen Ergebnisse werden im Array um eine Position nach hinten verschoben bzw. der letzte Eintrag gelöscht. Das ganze Array wird danach zurückgegeben.

`public bool calculate_and_decide(ref EVO.Common.Individuum_MetaEvo[] generation)`
Basierend auf den Ergebnissen der calculate-Methode wird durch Vergleich des Solutionvolumes zwischen mehreren Generationen (Anzahl durch Historylength bestimmt) die durchschnittliche Entwicklung berechnet. Liegt diese unterhalb des Minimumchange-Wertes, wird 'true' zurückgegeben um die Umschaltung zur lokalen Optimierung auszulösen.
`public double get_last_volume()`
Gibt den zuletzt berechneten Solutionvolume-Wert zurück

5.12.3 Hauptdiagramm(TeeChart)

Die Klasse Hauptdiagramm stellt zwar eine bereits vorhandene Klasse dar, wird hier aber dennoch beschrieben da die komplette Zeichenfunktionalität von ihr abhängt. Auf die Initialisierung, die bereits außerhalb von MetaEvo stattfindet, wird hier nicht näher eingegangen.

Methoden:

```
Public Sub ZeichneIndividuum(ByVal ind As Common.Individuum, ByVal runde As
 Integer, ByVal pop As Integer, ByVal gen As Integer, ByVal nachf As Integer
, ByVal Farbe As System.Drawing.Color, Optional ByVal ColEach As Boolean =
 False)
```

Zeichnen eines Individuums:
'Runde' definiert eine Wiederholung der kompletten Berechnung, 'pop' eine parallele Ausführung mehrerer Generationen. Beides ist in MetaEvo nicht vorgesehen und wird daher als Konstante übergeben. Die id des individuums wird als 'nachf' übergeben und eine Farbe aus System.Drawing.Color übergeben.

```
Public Sub ZeichneSekPopulation(ByVal pop()As Common.Individuum)
```

Zeichnet eine Reihe von Individuen mit gleicher Farbe. Diese Funktion wird zum Zeichnen der nach der Selektion verbleibenden Eltern-Individuen genutzt.

```
Public Sub LöscheLetzteGeneration(ByVal pop As Integer)
```

Löscht die zuletzt gezeichnete Generation der Population 'pop' aus dem Bild.

5.13 Benutzereingaben

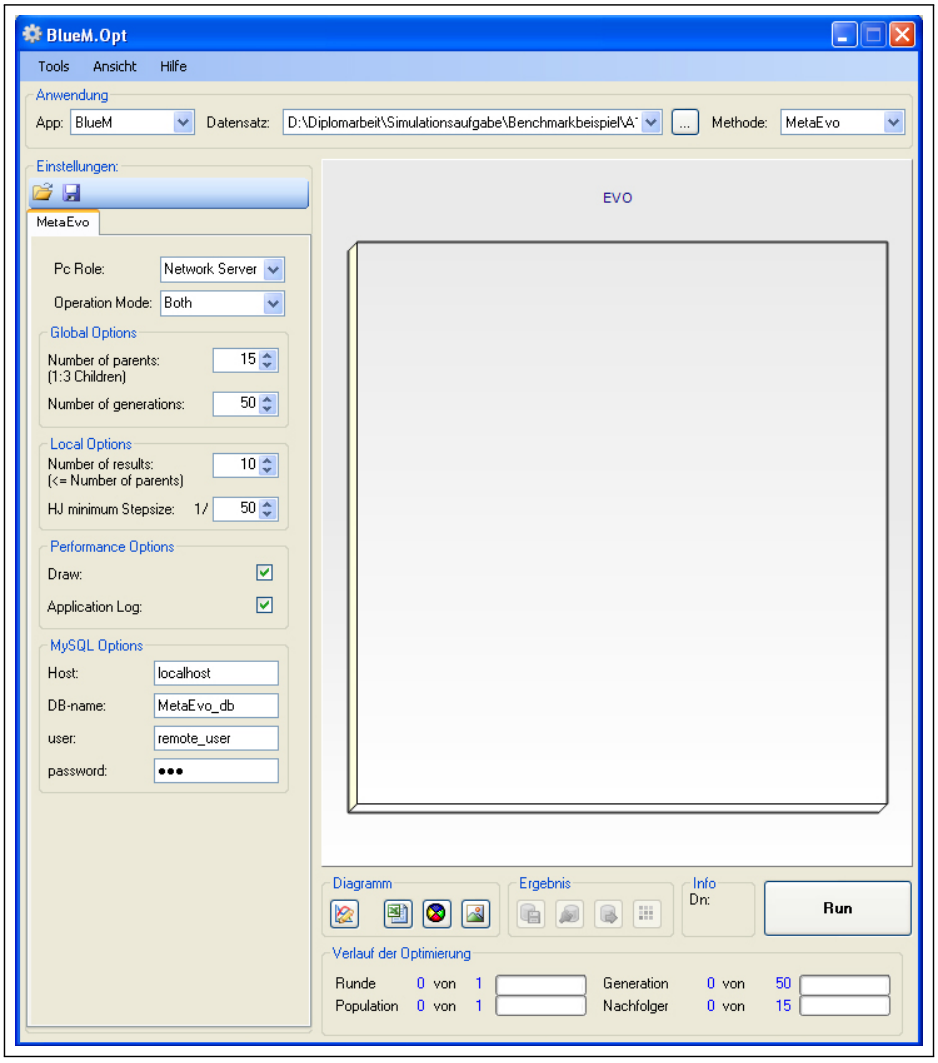

Abbildung 5.16 – Benutzereingabe

Die Benutzereingabe ist dank des modularen Aufbaus gut für eine Erweiterung geeignet. Nach Wahl der Applikation, des Datensatzes sowie der Methode werden die nicht benötigten Eingabeschuppen für die Einstellungen ausgeblendet. Es muss lediglich eine weitere Eingabeschuppe für MetaEvo eingefügt werden.

Unter 'PC Role' kann neben 'Network Server' auch 'Network Client' und 'Single PC' gewählt werden. Im Feld 'Operation Mode' kann neben 'Both' auch 'Local Optimizer' und 'Global Optimizer' gewählt werden. Alle Benutzereingaben werden anschließend in EVO_Settings gespeichert und sind so für alle Programmteile zentral zugänglich.

5.13.1 Schedulingviewer

Während der Umsetzung und Testläufe im Bereich der Netzwerkkommunikation wurde ein kleines Werkzeug zur Sichtprüfung der Individuenzuteilung im Netzwerk erzeugt. Da dieses Werkzeug für alle beteiligten Clients einsehbar sein sollte, wurde auf Basis von PHP und HTML eine dynamische Tabelle mit Zuordnungsgrafiken erzeugt, die alle wesentlichen Informationen der aktuellen Berechnung beinhaltet.

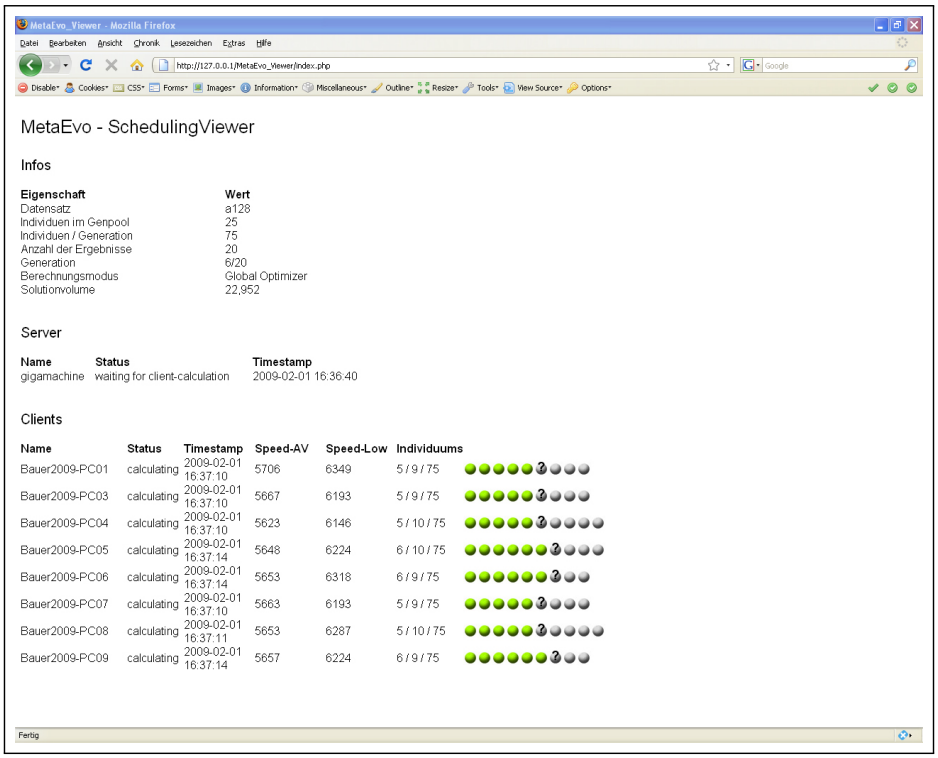

Abbildung 5.17 – Schedulingviewer

6 Auswertung

In diesem Kapitel wird der entwickelte Ansatz aufgrund von Messergebnissen überprüft. Des Weiteren wird eine Laufzeitanalyse des entstandenen Systems durchgeführt um ein Skalierungsverhalten der Software zu bestimmen.

6.1 Optimierungsaufgaben

Um die Funktionalität des Systems zu prüfen, werden jeweils mit dem Testbeispiel Zitzler/Deb/Theile T3 sowie dem in Arbeitsblatt ATV-A 128 beschriebenen Kanalnetz drei Disziplinen exemplarisch berechnet.

- Evolutionäre Algorithmen mit fester Individuenzuordnung
  (evolutionär ohne Initiative)
- Evolutionäre Algorithmen mit dynamischer Individuenzuordnung basierend auf Initiative (evolutionär)
- Hybrider Ansatz mit Evolutionären Algorithmen, Umschaltpunkt und lokaler Optimierung mit Hook&Jeeves (hybrid)

6.1.1 Optimierungsaufgabe Zitzler/Deb/Theile T3

Die im bestehenden System fest integrierte Testfunktion beschreibt eine mehrteilige und damit nicht stetige Paretofront. Die wichtigste Eigenschaft einer Testfunktion gegenüber einer realen Aufgabenstellung ist, dass zur 'Simulation' kein Modell zur Anwendung kommt, sondern wie in diesem Fall zwei einfache, mathematisch festgelegte Zielfunktionen. Zum Zweck der schnellen Berechnung einer Optimierung im Rahmen des Testens von Algorithmensystemen eignen sich solche Testfunktionen daher sehr gut (50 'Simulationen' pro Sekunde mit Intel Core 2 Duo E8400, 3GHz). Um komplexe evolutionäre Optimierungsstrategien in angemessenem Rahmen zu testen, besitzen diese Funktionen jedoch 15 Optimierungsparameter. Diese Tatsache stellt für die Anwendung von Hook&Jeeves eine wichtige Eigenschaft dar, da diese Anzahl für die Laufzeit die Eigenschaft eines linearen Vorfaktors besitzt.

Für dieses Beispiel wurden zehn Testläufe in den oben genannten Disziplinen durchgeführt für die jeweils galt:

- Größe des Genpools: 25 Individuen
- Anzahl der erzeugten Individuen pro Generation: 75 Individuen (Nach dem von Rechenberg [Rechenberg, 1994] empfohlenen Verhältnis von Genpool zu neuer Generation von 1:3)

- Anzahl der Generationen: 50
- verwendete Algorithmen: Zufällige Einfache Mutation, Ungleichverteilte Mutation, Zufällige Rekombination, Intermediäre Rekombination, Diversität aus Sortierung, Totaler Zufall, Dominanzvektor
- Zufällig erzeugter, erster Genpool
- Hybrid: Reduzierung auf 15 lokale Optimierungen

6.1.2 Optimierungsaufgabe Arbeitsblatt ATV-A 128

Dieses Beispiel einer realen Aufgabenstellung aus dem Arbeitsblatt ATV-A 128 mit dem Titel 'Richtlinien für die Bemessung und Gestaltung von Regenentlastungsanlagen in Mischwasserkanälen' stellt eine allgemeine Benchmarkfunktion dar. Das Arbeitsblatt gilt als Standardvorlage für die Bemessung von Entlastungsanlagen in Mischwasserkanalnetzen (d.h. die Volumen und Drosselabflüsse der Regenüberläufe und Regenüberlaufbecken). Im Rahmen dieses Arbeitsblattes ist ein Beispiel-Kanalnetz implementiert, welches mit der Simulationssoftware BlueM die Mischwasserabflüsse (d.h. die Abflüsse zur Kläranlage und solche, die über die Entlastungsanlagen in die Gewässer entlastet werden) simuliert. Es wird grundsätzlich angestrebt, möglichst wenig Mischwasser an den Bauwerken zu entlasten (da dieses dann anstatt zur Kläranlage in Flüsse bzw. Bäche eingeleitet wird). Ziel ist es, die entlastete Wassermenge der Drosselabflüsse an verschiedenen Bauwerken gleichzeitig zu minimieren. [1] Die Optimierungsaufgabe besteht lediglich aus drei Optimierungsparametern, benötigt für die Simulation allerdings sechs Sekunden pro Parametersatz (Intel Core 2 Duo E8400, 3GHz).

Für dieses Beispiel wurde jeweils nur ein Testlauf in den oben genannten Disziplinen durchgeführt:

- Größe des Genpools: 60 Individuen
- Anzahl der erzeugten Individuen pro Generation: 180 Individuen
- Anzahl der Generationen: 30
- verwendete Algorithmen: Zufällige Einfache Mutation, Ungleichverteilte Mutation, Zufällige Rekombination, Intermediäre Rekombination, Diversität aus Sortierung, Totaler Zufall, Dominanzvektor
- Zufällig erzeugter, erster Genpool
- Solutionvolume mit Offsetwert der Zielfunktionen (5.5 ; 41.5)
- Hybrid: Reduzierung auf 20 lokale Optimierungen

[1] Beschreibung nach IHWB

6.2 Funktionalität

Zur Überprüfung der Funktionalität einzelner Komponenten und Verhaltensweisen dienen die oben beschriebenen Optimierungsaufgaben in verschiedenen Konstellationen als Vergleichsbasis. Es sei darauf hingewiesen, dass die gezeigten Eigenschaften, wie bei zufallsbasierten Anwendungen üblich, keinen Anspruch auf Allgemeingültigkeit ausdrücken können.

6.2.1 Evolutionäre Algorithmen mit und ohne Initiativewerten

Als sehr typisch erwiesen sich die Ergebnisse des Berechnungsprotokolls 2009117_1565 des Testbeispiels Zitzler/Deb/Theile T3: Es ist zu erkennen, dass, basierend auf der vorliegenden Berechnungssituation der Aufgabe, verschiedene Algorithmen mehrheitlich am schnellen Fortschritt der Berechnung beteiligt sind. Die evolutionären Algorithmen ohne Initiative erreichen nach durchschnittlich 20,3 Generationen einen relativ stabilen Wert des Solutionvolumes, die evolutionären Algorithmen mit Initiative erreichen diesen Punkt nach durchschnittlich 18,8 Generationen. Der Unterschied beträgt hier also 1,5 Generationen. Deutlich wird die Funktionalität der Initiative und die Interpretation der Messwerte durch zwei Grafiken:

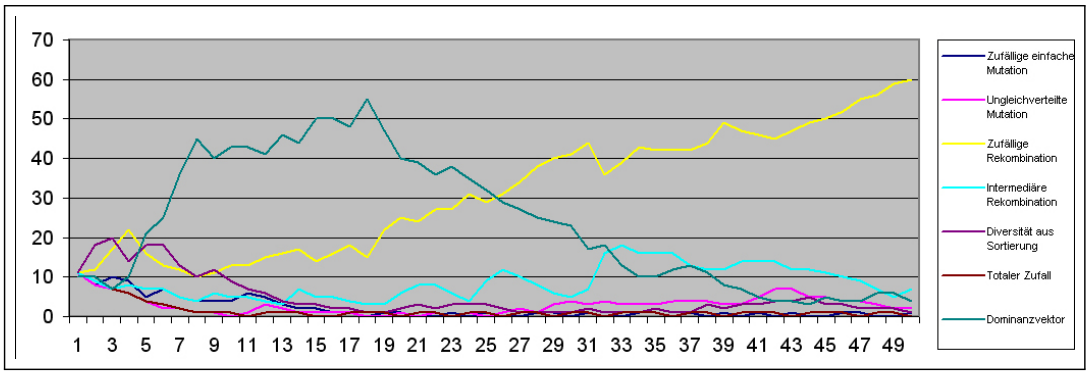

Abbildung 6.1 – Evo mit Initiativewerten: Algorithmenzuteilung Zitzler/Deb/Theile T3

Aus den Grafiken ist zu erkennen, dass zu Beginn der Berechnung Algorithmus 7 (Dominanzvektor 4.3) sehr oft eingesetzt wird. Dies kann dadurch erklärt werden, dass in der vorliegenden Situation eine tatsächliche Paretofront noch nicht gut ausgebildet ist und der Abstand der dominierten Individuen zu den dominanten Individuen im Genpool relativ groß sein kann. Werden zwei solche Individuen mit großer Distanz für die Durchführung dieses Algorithmus gewählt, ist ein resultierendes Individuum, insofern es gültig ist, einen großen Schritt näher an der Paretofront platziert und wird selten noch innerhalb dieser

Generation durch andere Individuen dominiert. Somit ist die Überlebensrate der Individuen dieses Algorithmus zu diesem Zeitpunkt der Berechnung relativ hoch und resultiert in einer vermehrten Zuteilung von Individuenplätzen in der folgenen Generation. Es bleibt aber zu bemerken, dass wie bereits beschrieben, dieser Algorithmus auf lineare Zusammenhänge zwischen Parameter- und Lösungsraum angewiesen ist und hier offensichtlich in großem Maß profitiert.

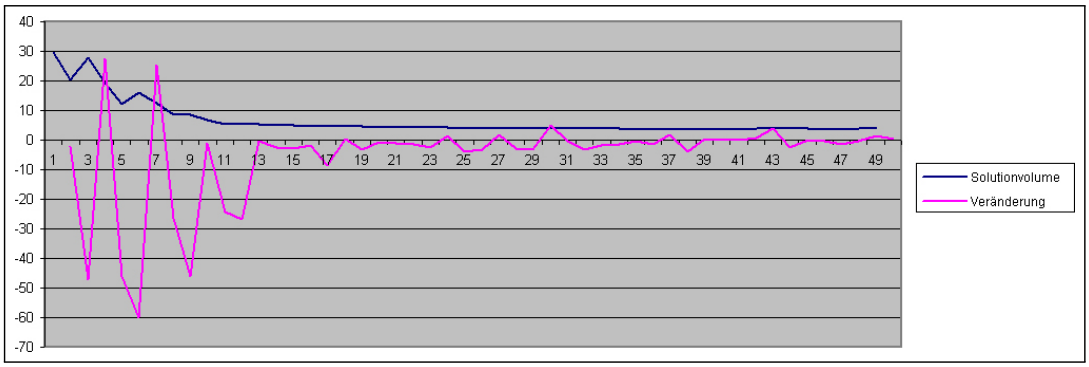

Abbildung 6.2 – Auswertung: Evo mit Initiativewerten 2

Nähert sich die Menge der Individuen im Genpool weiter der Paretofront, sind die Fortschritte die 'Dominanzvektor' erzeugen kann tendenziell gering und die Zuteilung ändert sich. Es zeigt sich, dass Algorithmus 3 (Zufällige Rekombination) mit der Nähe zur Paretofront der optimalen Lösungen immer erfolgreicher arbeitet. Hierzu zeigt Grafik **??** das Verhalten des Genpools. Das Solutionvolume (Blau) nimmt, mit kleinen Sprüngen, kontinuierlich ab. Die prozentualen Veränderungen, die dabei in der visualisierten Form der Paretofront vorkommen, werden von der zweiten Kurve (Rosa) dargestellt. Sind die Veränderungen hoch, ändert sich also die Abstandssumme der Individuen im Genpool (in diesem Fall vom Ursprung im Lösungsraum) stark und kann auch in einer rechnerischen Verschlechterung resultieren. Diese Verschlechterungen sind allerdings, bedingt durch die Formel für das Solutionvolume und dem Vorgehen der Selektion, Verbesserungen in der Diversität wie bereits in 4.5.2 beschrieben.

Ein weiterer Test zeigte erwartungsgemäß, dass die Wahl von gleichen Parametern für alle Individuen des Genpools, Algorithmus 6 (Totaler Zufall) zwar kurzfristig, aber in hohem Maß eingesetzt wurde. Durch die Wahl von zufälligen Startparametern für alle Elternindividuen wurde diese Entwicklung vorweggenommen.

Im Arbeitsblatt ATV-A 128 kann keine Erhöhung der Konvergenzgeschwindigkeit beobachtet werden. Eine Erklärung könnte das Erreichen des stabilen Solutionvolumes bereits nach 4 Generationen sein und eine ähnliche Beschleunigung wie in Zitzler/Deb/Theile T3 würde sich hier nur in einem Bruchteil einer Generation bemerkbar machen.

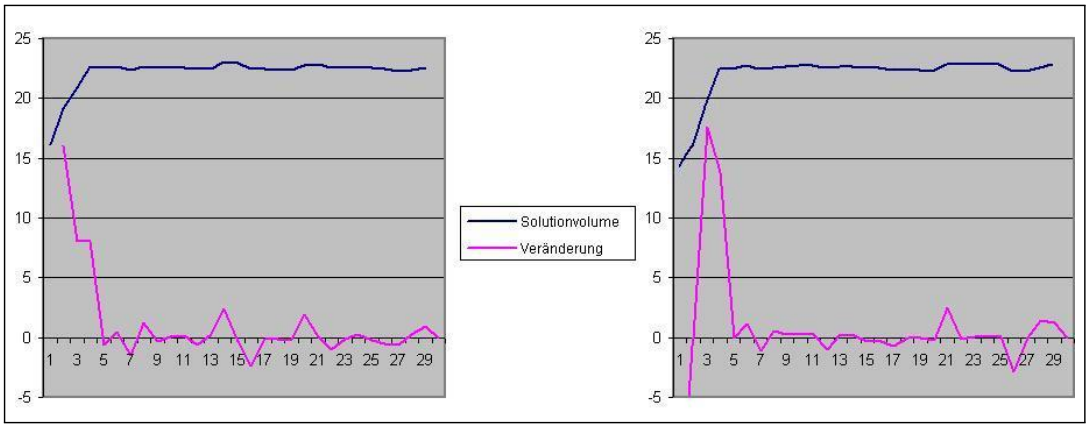

Abbildung 6.3 – Evo mit Initiativewerten: Algorithmenverteilung Arbeitsblatt ATV-A 128

Nach mehrfachen Testläufen mit verschiedenen Problemstellungen kann gesagt werden, dass das Prinzip der Initiative die Konvergenzgeschwindigkeit der evolutionären Algorithmen erhöht.

6.2.2 Hybrider Ansatz kontra evolutionäre Algorithmen

Der nun folgende Vergleich stellt zwei Strategien gegenüber: Zum Einen die rein auf evolutionären Algorithmen beruhenden Optimierungsstrategie und zum Anderen eine Strategie, die nach dem Finden eines Umschaltpunktes von den global operierenden, evolutionären Algorithmen zur lokalen Optimierung mit Hook&Jeeves wechselt. Es ist nötig, einen Umschaltpunkt im Verlauf einer evolutionären, globalen Optimierung zu bestimmen, die Daten des Genpools abzuspeichern, und an dieser Stelle unabhängig voneinander mit der evolutionären Optimierung und der lokalen Optimierung fortzufahren.

Abbruchkriterien

Um die beiden Ansätze letztendlich zu vergleichen, gilt es den Status der Lösung zu bestimmen:

Im Fall der lokalen Optimierung ist dies ohne Weiteres möglich, da der hier verwendete Hook&Jeeves-Algorithmus eine definierte Ausgangsschrittweite nutzt und diese bei Bedarf halbiert. Ein Abbruchkriterium basierend auf dieser Schrittweite der Optimierungsparameter ist somit einfach zu bestimmen und könnte im realen Anwendungsfall sogar als Gestaltungsparameter gelten. Ein weiterer Vorteil dieser klaren Definierbarkeit ist die Möglichkeit, die Modellgenauigkeit vollständig auszunutzen.

Bei der rein evolutionären Optimierung gestaltet sich dies weitaus schwieriger. Mathematisch wäre ein Abbruchkriterium erst erreicht, wenn eine optimale Verteilung der Indivi-

duen auf der Paretofront der optimalen Lösungen anzutreffen wäre. Dies ist allerdings in der Praxis kaum nachzuweisen, da die Paretofront der optimalen Lösungen nicht bekannt ist. So gilt es den Punkt zu identifizieren, ab dem die Paretofront nur fluktuiert und sich weniger als ein Mindestmaß verbessert. Da diese Betrachtung allerdings nur möglich ist, nachdem eine Reihe von unnötigen Lösungen berechnet wurden und darüber hinaus als höchst subjektiv eingeordnet werden kann, ist ein solches Kriterium nicht zu bevorzugen.(Im vorliegenden Testbeispiel sind zur Bestimmung des Abbruchkriteriums für evolutionäre Algorithmen oft mehr als zehn Generationen nötig.)

Berechnungsgeschwindigkeit

Vergleicht man nun die beiden Prozesse basierend auf der Anzahl der Simulationen bis zum Abbruchkriterium wird deutlich, dass sich die Rechenzeiten stark unterscheiden. Während im Beispiel Zitzler/Deb/Theile T3 die Optimierung mit evolutionären Algorithmen etwa gleich viele Schritte vor und nach dem Umschaltpunkt benötigt um anschaulich die Paretofront der optimalen Lösungen zu erreichen, ist die Anzahl der Simulationen für die lokale Optimierung ungleich höher. Bei genauerer Betrachtung wird jedoch deutlich, dass die Lösungsqualität der lokalen Optimierung höher ist.

Die evolutionären Algorithmen profitieren im Allgemeinen vom Vorhandensein mehrerer Individuen als potentielle Basis der neuen Generation, da im Gesamtzusammenhang die Diversität durch 'Crowding Selection' (eine auf Bevölkerungsdichte basierendes Selektionsverfahren) und der sprunghaften Fortschritt einiger weniger Individuen für die ganze Generation positive Auswirkungen hat. Im Gegensatz dazu sind die lokalen Optimierungen unabhängig voneinander und ihre Anzahl stellt einen linearen Skalierungsfaktor für die Rechenzeit dar. In der Konsequenz bedeutet dies, dass vor der lokalen Optimierung eine Reduzierung der Individuen im Genpool auf die tatsächlich gewünschte Anzahl von Ergebnissen sehr wichtig für die Gesamtrechenzeit ist.

Ein weiterer Punkt im Zusammenhang mit der Verwendung von Hook&Jeeves ist, dass im ungünstigsten Fall bis zum Erreichen der Paretofront der optimalen Lösungen viele Iterationen des Algorithmus durchlaufen werden müssen. Es liegt daher im Interesse der Laufzeit eines solchen Prozesses, die initiale Schrittweite des Algorithmus nicht zu klein zu wählen. Im Idealfall wäre diese Schrittweite etwa auf die Entfernung zur Paretofront zu setzen.

Qualität des Ergebnisses

Die Qualität des Ergebnisses richtet sich zunächst nach der Nähe zur Paretofront der optimalen Lösungen und kann so mit Hilfe des Beispiels Zitzler/Deb T3 sehr gut verglichen werden.

Zunächst kann gesagt werden, dass die lokale Optimierung mit Hook&Jeeves sehr zuverlässig jedes Minimum, welches einem Individuum nahe steht, findet. Dies kann dadurch erklärt werden, dass die Halbierung der Schrittweite für die Optimierungsparameter rechnerisch ein Absuchen des kompletten Parameterraums darstellt. Daher muss durch dieses Absuchen das Minimum gefunden werden.

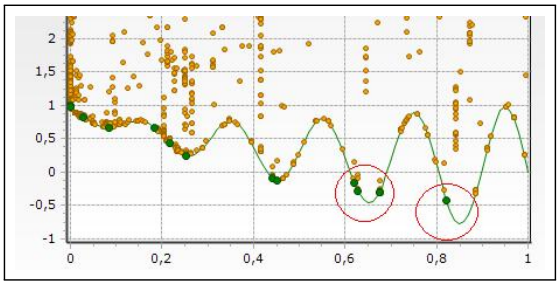

Abbildung 6.4 – Ungenaue Lösungen bei evolutionäre Algorithmen

Für die hier verwendete evolutionäre Optimierung stellt diese Messlatte ein unerreichbares Ziel dar, da diese Aufgrund ihrer Berechnungsstruktur nicht von selbst terminieren. Abhilfe könnte mit einer Anpassung von Mutationsschrittweiten und Rekombinationen basierend auf Nähe im Zielfunktionsraum geschaffen werden. Die starren Eigenschaften sind aber insofern wichtig, als dass Beispielsweise eine Anpassung der Schrittweiten der evolutionären Algorithmen die Robustheit dieser Algorithmen an sich beeinträchtigen kann. In einem solchen Fall ist nicht zu unterscheiden, ob ein gefundenes Minimum von globaler oder lokaler Natur ist und damit eine Schrittweitenanpassung rechtfertigt oder nicht.

Es bleibt allerdings zu bemerken, dass die evolutionären Algorithmen im Allgemeinen ohne weiteres Zutun eine gleichmäßigere Abdeckung der Paretofront produzieren da 'Crowding Selection' ein Teil der Selektionsstrategie darstellt. Für die lokale Optimierung mit Hook&Jeeves bleibt zu bedenken, dass, wie in Abschnitt 4.4 beschrieben, eine dynamische Anpassung der Wichtungsfaktoren eine wesentliche Verbesserung der Abdeckung mit sich bringen kann.

Im Beispiel Arbeitsblatt ATV-A 128 ist dieses Ergebnis ebenfalls deutlich: Die nach 30 Generationen rein evolutionär gefundenen Lösungen werden teilweise von den Lösungen der hybriden Optimierung mit insgesamt niedrigerer Laufzeit dominiert.

Nach 9 Generationen evolutionärer Optimierung wird der Umschaltpunkt ausgelöst und eine Reduzierung auf 20 Hook&Jeeves Prozesse startet. Die Anzahl der Simulationen, die durch diese lokalen Optimierungen ausgelöst werden, beträgt 1005. Bei einer Größe der ursprünglichen Generation von 180 Individuen entspricht dieser Werte etwa 5,58 Generationen. Würde man dieses Ergebnis auf 40 weitere Individuen ausdehnen, müsste nach 9 + (3x 5,58) = 25,74 Generationen eine Lösung vorliegen, die deutlich die rein evolutionär gefundene Lösung nach 30 Generationen übertrifft.
Dazu ist allerdings zwingend die Anpassung der Wichtungsfaktoren der Zielfunktionen in den einzelnen Hook&Jeeves-Prozessen nötig um eine ähnlich gute Abdeckung der Paretofront, wie sie die evolutionäre Optimierung produziert, zu erreichen.

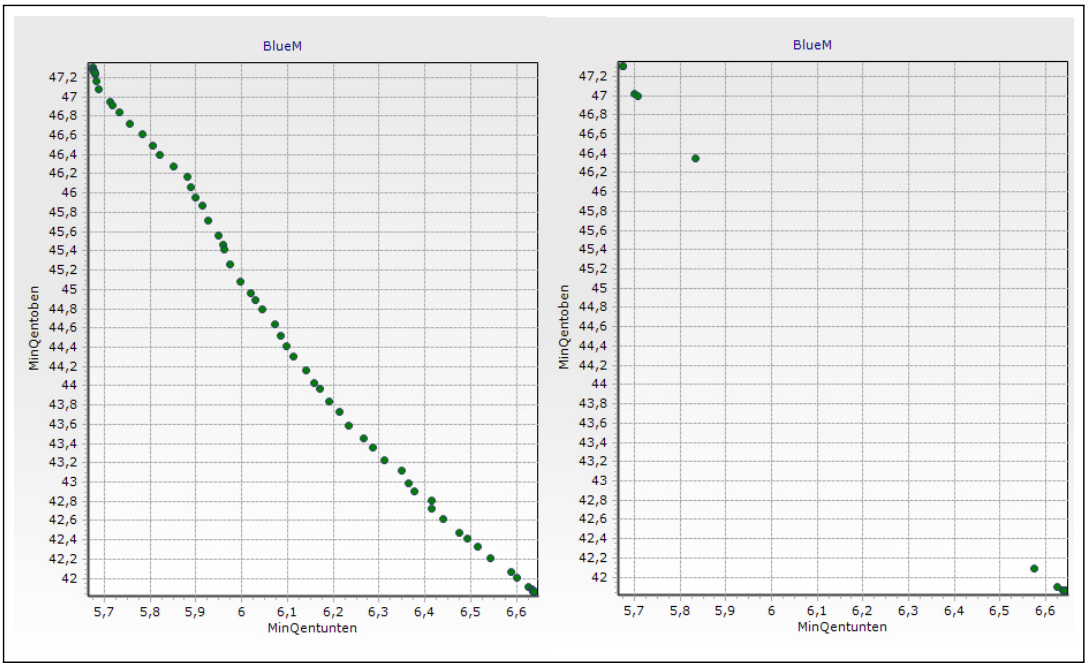

Abbildung 6.5 – Arbeitsblatt ATV-A 128: Links: Rein evolutionäre Optimierung, Rechts: Hybride Optimierung

Es kann also gesagt werden, dass die hybride Optimierungsstrategie bessere Lösungen in kürzerer Zeit findet als die rein evolutionäre Optimierungsstrategie. Dabei bleibt dennoch zu bemerken, dass die evolutionäre Optimierung bis zum Umschaltpunkt bessere Lösungen produziert als dies eine Optimierung mit ausschließlicher Anwendung von Hook&Jeeves könnte.

Fazit

Die hier vorgestellten Ansätze unterscheiden sich gravierend hinsichtlich ihrer Anwendungsbereiche und es kristallisieren sich zwei Klassen von gesuchten Lösungsmengen heraus:

Die evolutionäre Optimierung bis zum Finden des Umschaltpunktes dient im Wesentlichen dem schnellen Auffinden vieler grober Lösungen und berechnet diese effektiv. Es ist also die schnell erreichbare Vielfalt, die im Vergleich zur hybriden Optimierungsstrategie mit nachgeschalteter lokaler Optimierung, eine kürzere Ausführungszeit benötigt. Ist die Qualität der Lösungen zweitrangig, da durch nicht mathematisch erfassbare Auswahlkriterien eine anschließende Selektion durchgeführt wird, spielt die Vielfalt der Lösungen die wichtigste Rolle und eine rein evolutionäre Optimierungsstrategie ist das Mittel der Wahl.

Die hybride Strategie besitzt einige Vorteile gegenüber der rein evolutionären Strategie und spricht ein anderes Anwendungsfeld an. So findet die hybride Optimierungsstrategie sehr gute Lösungen in einer vorher frei definierbaren Skalierung von Parameterschritten. Die Berechnung einer solchen Lösung benötigt im Vergleich zur rein evolutionären Optimierung zwar mehr Rechenzeit, ist aber qualitativ hochwertiger. Eine Reduktion der vom globalen Algorithmus erzeugten Ausgangsmenge an Individuen auf die Anzahl der gewünschten Lösungen macht den Aufwand noch besser beherrschbar. Nicht zuletzt ist die klar definierbare und garantiert erreichbare Abbruchbedingung ein nicht zu unterschätzender Vorteil in einer realen Anwendung.

6.3 Laufzeiten

Die Laufzeituntersuchung der Software kann in verschiedene Komponenten unterteilt werden:

- Erzeugen einer Generation
- Erzeugungsstrategie für Individuen
- Netzwerkkommunikation
- Simulation
- Erzeugen des neuen Genpools
- Prüfen auf Umschaltung zur lokalen Optimierung

Im Folgenden benutzte Parameter zur Laufzeitangabe:

IN_+	Anzahl neuer Individuen
IN_g	Anzahl der Individuen im Genpool
Op	Anzahl der Optimierungsparameter
Zf	Anzahl der Zielfunktionen
Clients	Anzahl aktive und funktionierende Netzwerkclients

Tabelle 6.1 – Parameter zur Laufzeitbestimmung

6.3.1 Erzeugen der Individuen

Die Laufzeit beim Erzeugen einer Generation ist zunächst abhängig von deren Größe. Da jedes neu erzeugte Individuum möglichst eine neue Zusammenstellung von Optimierungsparametern aufweisen soll, ist die Laufzeit ebenfalls abhängig von der Anzahl der Optimierungsparameter. Beide Parameter nehmen linear Einfluss, sodass man von einem Aufwand von

$$f = O(IN_+ * Op * Algorithmusfaktor)$$

sprechen kann. Die Wahl des Algorithmus nimmt unterschiedlichen Einfluss auf die Laufzeit. So sind Rekombinationsalgorithmen generell auf mindestens zwei weitere Individuen angewiesen die maximal mit linearer Suche aus dem Genpool nach verwendbaren Individuen suchen. Da die Individuen im Genpool bereits nach wenigen Generationen die Elite der gültigen Individuen (keine Nebenbedingungen verletzend) darstellen, kann dieser Faktor bei den bisher implementierten Algorithmen vernachlässigt werden und als feste Obergrenze angesehen werden für die gilt:

$$\lceil Algorithmusfaktor \rceil = (IN_g)^2$$

6.3.2 Erzeugungsstrategie für Individuen

Die Erzeugungsstrategie hängt maßgeblich vom Problem und den gewählten Algorithmen sowie zum grossen Teil von Zufällen ab. Diese Eigenschaften der evolutionären Algorithmen macht es unmöglich eine zuverlässige Laufzeitangabe zu definieren. Allerdings kann gesagt werden, dass die Generationengröße, die Anzahl neuer Individuen pro Genpool-Individuum sowie der Anzahl der lokalen Optimierungsprozesse die nach der globalen Optimierung verbleiben den entscheidenden Anteil der Laufzeit ausmachen.

6.3.3 Netzwerkkommunikation

Die Netzwerkkommunikation selbst geht mit konstantem Aufwand in die Laufzeit ein, da die Aktualisierung der Individuen aus dem Netzwerk zeitgesteuert ist. Die zur Netzwerkkommunikation gehörende Zuordnung der Individuen ist allerdings von der Anzahl der Clients und der Anzahl der Individuen abhängig. In der Zuordnung kommt vierfach eine doppelte Schleife über Individuen und Clients vor.

$$f = O(4 * IN_+ * Clients * (Op + Zf))$$

6.3.4 Simulation

Die Simulation als vorhandener Teil der Anwendung geht im vorliegenden Benchmarkbeispiel als weitaus größter Teil in die Laufzeit ein, ließe sich aber mit einem theoretisch riesigen Netzwerk wiederlegen. Da dies im Normalfall jedoch nicht auftritt, kann dieser überwiegende Einfluss der Simulation als genereller Fall angesehen werden. In welcher

Form die Anzahl der Optimierungsparameter im Zusammenhang mit der Anzahl der Zielfunktionen die Berechnungsdauer beeinflusst ist allerdings nicht klar.[2]

$$f = O((Op\ ?\ Zf) * Simulationszeitraum * x)$$

Für die Simulation ist es üblich, eine ganze Generation simulieren zu lassen. Hier spielt für die Gesamtdauer die Anzahl der zu simulierenden Individuen genauso eine Rolle wie die Anzahl der aktiven Clients im Netzwerk.

$$x \in \mathbb{N}_+ \text{ wobei } x = \left\lceil \frac{IN_+}{Clients} \right\rceil$$

Verfälscht wird diese Wahrnehmung durch die eingangs erwähnten Testfunktionen, da die 'Simulation' im Vergleich zu realen Aufgabenstellungen eine wesentlich geringeren Ausführungszeit besitzen. (Im Benchmarkbeispiel ist von einer reinen Simulationszeit pro Individuum von mehreren Sekunden auszugehen wohingegen die Testbeispiele etwa um den Faktor 500 weniger Rechenzeit benötigen - anhängig vom konkreten Testbeispiel.)

6.3.5 Erzeugen des neuen Genpools

Die Erzeugung des Genpools stellt algorithmisch den komplexesten Teil der Aufgabe dar. Sie setze sich zusammen aus:

1. Dominanzprüfung zwischen Genpool und neuen Individuen
2. Quicksort der neuen Individuen nach einer vorher festgelegten Zielfunktion
3. Dominanzprüfung innerhalb der neuen Individuen
4. Mengenanpassung des neuen Genpools
5. Erzeugen des neuen Genpools
6. Quicksort des Genpools

Zu Beginn liegen Genpool und neue Individuen unsortiert vor. Die Dominanzprüfung zwischen Genpool und diesen neuen Individuen beschränkt sich zunächst nur auf den sequentiellen Vergleich der Individuen im Genpool mit allen neuen Individuen, da die Individuen des Genpools sich im Normalfall nicht dominieren. Dieser Vergleich besitzt also den Aufwand:

$$(1.)\ f = O(IN_+ * IN_g * Zf)$$

[2] Im Benchmarkbeispiel A128 handelt es sich um eine Zeitreihe deren Beginn und Ende im Datensatz innerhalb der Grenzen der Verfügbarkeit von Daten beliebig variiert werden können.

Es bleibt zu bemerken, dass durch frühe Abbruchkriterien beim Vergleich der Vorfaktor stark verringert wurde. Nach dieser Prüfung werden die neuen Individuen mit Quicksort nach einem beliebigen Zielfunktionswert sortiert. Bekannt ist, dass dies den Aufwand

$$(2.)\ f = O(IN_+ * log(IN_+))$$

besitzt. Durch diese Funktion kann die anschließende Dominanzprüfung der neuen Individuen untereinander mit dem Vorfaktor $\frac{1}{2}$ versehen werden, was rechnerisch bei steigender Individuenanzahl einen Laufzeitvorteil mit sich bringt.

$$(3.)\ f = O\tfrac{1}{2}(IN_+^2 * Zf)$$

Die Mengenanpassung und Erzeugung des neuen Genpools sind von linearem Aufwand, basierend auf der Menge der Individuen.

$$(4.,5.)\ f = O(IN_+ + IN_g)$$

Zum Abschluss wird der Genpool noch einmal nach dem vorher gewählten Kriterium sortiert um den Algorithmen eine besser strukturierte Menge von Individuen zu übergeben. Algorithmen wie 'Diversität aus Sortierung' nutzen diese Eigenschaft.

$$(6.)\ f = O(IN_g * log(IN_g))$$

Die allgemeine Vorgehensweise nach [Rechenberg, 1994] empfiehlt ein Größenverhältnis von 1:3 zwischen Genpool und neu erzeugten Individuen. Geht man noch allgemeiner davon aus, dass $IN_g \leq IN_+$ gilt, lässt sich der gesamte Aufwand abschätzen mit:

$$f_{ges} = O(IN_+^2 * Zf)$$

Der wichtigste Faktor bezüglich der Laufzeit für die Erstellung eines neuen Genpools ist also die Anzahl der neuen Individuen.

6.3.6 Visualisierung einer Generation

Die Visualisierung der Individuen kann zu Beginn unterteilt werden in das Zeichnen und das Erzeugen der Ausgabe, die Applicationlog bereitstellt. Beides spielt bei den Testbeispielen eine wesentlich größere Rolle als im Benchmarkbeispiel ist aber sonst eher von untergeordneter Wichtigkeit. Es gilt generell:

$$f = O((IN_g + IN_+) * Zf)$$

6.3.7 Prüfen auf Umschaltung zur lokalen Optimierung

Die Prüfung basiert, wie schon in 4.5.1 erwähnt, auf der Summe der Abstandsquadrate der Individuen zu einem Offestwert im Lösungsraum. Daher gilt intuitiv:

$$f = O(IN_g * Zf)$$

6.4 Grenzen der Software

Die im Rahmen dieser Diplomarbeit erzeugte Software und die umgesetzten Modularisierung sollen ein Höchstmaß an Flexibilität bieten. Es ist möglich, die erzeugenden Algorithmen vollständig und sehr einfach auszutauschen. Die Algorithmen die den neuen Genpool initial erstellen sind ebenfalls einfach austauschbar.

Die Grenzen dieser Software sind in der Basisstruktur zu suchen. So ist es lediglich möglich, beliebig lange Arrays vom Basistyp Individuum_MetaEvo an die einzelnen Funktionen zu übergeben. Der Basistyp Individuum_MetaEvo erlaubt allerdings die Konfiguration einer inhaltlich anpassbaren Feedbackinformation welche ein Double-Array beliebiger Größe darstellt. Dieses Array ist an das Individuum gekoppelt und so können jederzeit Zusatzinformationen gespeichert werden.

Ebenfalls eine Barriere stellt die Netzwerkfunktionalität dar - der Umfang der Änderungen im bestehenden Programm wären sehr umfangreich geworden um dieses Feature allgemein zur Verfügung zu stellen. Jedoch ist die zugehörige Klasse selbst als Modul zu verwenden.

Die verwendete Programmiersprache C# und C++ erschweren einen Einsatz und die Leistungsfähigkeit auf nicht-Windows-Systemen.

Die eher allgemeinen Grenzen der Software aufgrund der Verwendung von evolutionären Algorithmen stellen sich wie folgt dar:
Es kann nicht garantiert werden, dass alle Minima gefunden werden und es kann nicht definiert werden, in welchem Zeitraum eine Berechnung die optimalen Ergebniswerte erreichen wird.

6.4.1 Zusammenfassung

Mit dem Ergebnis, dass die Simulation den weitaus größten Rechenaufwand darstellt, entscheidet die Anzahl an gewünschten Lösungen über die zu wählende Strategie. Sind wenige Lösungen in hoher Qualität gefordert, ist die Anwendung einer hybriden Strategie mit den hier genutzten Algorithmen die bessere Wahl. Ist jedoch die schnelle Vielfalt der Lösungen entscheidend, sollte die rein evolutionäre Strategie bevorzugt werden.

Generell kann gesagt werden, dass die kritischste Komponente im Rahmen der Berechnung einer Aufgabe mit evolutionären Algorithmen die Anzahl der Individuen darstellt. Sie ist die einzige Komponente die quadratisch in die Aufwandsberechnung der erzeugenden und selektierenden Logik einfließt. Dennoch ist dieses Ergebnis im Rahmen der Benchmarkaufgabe sowie der sonstigen realen Aufgaben nicht relevant, da hier vielmehr die Simulationszeit sowie die Erzeugungsstrategie eine Rolle spielen. Die Simulationszeit kann durch konsequente Reduktion der zu simulierenden Werte auf die letztendlich gewünschten Eigenschaften in sehr großem Maß auf die Rechenzeit Einfluss nehmen und muss nicht zwangsläufig durch Reduktion der Individuen und damit der Lösungsvielfalt verwirklicht werden. In ähnlicher Weise stellt sich die Erzeugungsstrategie dar - hier ist es entscheidend, eine gute Algorithmuskombination für die Aufgabe zu finden, die einen schnellen und kontinuierlichen Fortschritt bringt. Eine Möglichkeit, diese spezifisch beste Algorithmuskombination zu finden wurde im Rahmen dieser Arbeit durch eine dynamische Zuteilung von Algorithmusresourcen, basierend auf deren jeweiligen Erfolg, mit dem Initiative-Prinzip umgesetzt. Nicht zuletzt ist die Netzwerkparallelisierung ein wichtiger Schritt zur effektiveren Nutzung des hier erweiterten Systems.

7 Ausblick

Die hier umgesetzte Lösung eines parallelen, multi-kriteriellen, hybriden Optimierungsalgorithmus zeigt deutlich die Vorteile der lokalen Optimierung als zweiten Teil der durch die globale Optimierung begonnenen Berechnung. Gleichzeitig existieren jedoch einige Möglichkeiten diese Optimierungsstrategie und die Netzwerkfunktionalität weiter zu entwickeln:

- Anpassung der Wichtungsfaktoren für die parallel ablaufenden Hook&Jeeves-Prozesse der lokalen Optimierung: Verbessert die Abdeckung der Paretofront im Ergebnis der Optimierung (4.4).
- Umsetzung von Multithreadding auf den simulierenden Clients: Erhöht die Simulationsgeschwindigkeit und reduziert so die Anzahl der benötigten Rechner.
- Nutzen von Erzeuger-Informationen zur lokalen Anpassung von Parametern globaler Optimierungsalgorithmen: Reagieren auf (lokale) Minima durch z.B. Reduzierung der Schrittweite. Hierdurch kann die Qualität einer Lösung die nur durch evolutionäre Algorithmen erzeugt wurde gesteigert werden oder die Übergabe an die lokale Optimierung verbessert werden.
- Dedizierte Rechenclients als Weiterentwicklung des hier genutzten Clients ohne grafische Oberfläche und im Idealfall vollständig automatisiert. Diese würden selbstständig nach Berechnungen suchen, die entsprechenden Probleme aus einer Datenbank laden und Simulationsaufgaben übernehmen. Ziel ist das einfachere und schnellere Berechnen von Optimierungsaufgaben.

Im Bezug auf die Steuerung von Kanalnetzen kann gesagt werden, dass die hier entwickelte Plattform zusammen mit spezialisierten Algorithmen eine intelligente Steuerung ermöglicht die das Potential hat, die Effektivität und nicht zuletzt die Sicherheit in der Realität enorm zu steigern. Dabei wird klar, dass die Anwendung keinesfalls auf Kanalnetze beschränkt ist, diese jedoch zum jetzigen Entwicklungsstand und der zur Verfügung stehenden Rechenleistung ein sehr gutes Umfeld bieten, Echtzeit-Anwendungen zu entwickeln und die Forschung auf diesem Gebiet voranzutreiben.

Literaturverzeichnis

[Adamy, 2007] Adamy, J. 2007. *Fuzzy-Logik, Neuronale Netze und Evolutionäre Algorithmen*. Shaker Verlag.

[Carlos A. Coello Coello & Veldhuizen, 2007] Carlos A. Coello Coello, Gary B. Lamont, & Veldhuizen, David A. Van. 2007. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer.

[Erick Cantú-Paz, 1997] Erick Cantú-Paz, David E. Goldberg. 1997. Predicting Speedups of Ideal Bounding Cases of Parallel Genetic Algorithms. *ICGA*.

[Frankhauser, 2004] Frankhauser, R. 2004. REBEKA II - Software zur Unterstützung der Massnahmenplanung. *gwa(11)*.

[Goldberg & Richardson, 1987] Goldberg, D. E., & Richardson, J. 1987. Genetic algorithm with sharing for multimodal function optimization. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*.

[Goldberg & Richardson, 1989] Goldberg, D. E., & Richardson, J. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company.

[Holst, 2003] Holst, Johnny N. 2003. USING WHOLE BUILDING SIMULATION MODELS AND OPTIMIZING PROCEDURES TO OPTIMIZE BUILDING ENVELOPE DESIGN WITH RESPECT TO ENERGY CONSUMPTION AND INDOOR ENVIRONMENT. *Eighth International IBPSA Conference*.

[Horstmann, 2006] Horstmann, Jutta. 2006. Freie Datenbanken im Unternehmenseinsatz. *Heise*.

[Jasper A. Vrugt, 2007] Jasper A. Vrugt, Bruce A. Robinson. 2007. Improved evolutionary optimization from genetically adaptive multimethod search. *PNAS*.

[Kalyanmoy Deb, 2001] Kalyanmoy Deb, Tushar Goel. 2001. A Hybrid Multi-Objective Evolutionary Approach to Engineering Shape Design.

[Kelley, 2002] Kelley, C.T. 2002. A Brief Introduction to Implicit Filtering.

[Meirlaen, 2002a] Meirlaen, J. 2002a. Immission Based Real-Time Control of the Integrated Urban Wastewater System. *Faculteit Landbouwkundige en Toegepaste Biologische Wetenschappen. Universität Gent. Dissertation.*

[Meirlaen, 2002b] Meirlaen, J. und P. A. Vanrolleghem. 2002b. Model Reduction through Boundary Relocation to Facilitate Real-Time Control Optimisation in the Integrated Urban Wastewater System. *Water Science and Technology 45 (4-5): 373-381*.

[Muschalla, 2006] Muschalla, Dirk. 2006. Evolutionäre multikriterielle Optimierung komplexer wasserwirtschaftlicher Systeme. *Technischen Universität Darmstadt, Fachbereich Bauingenieurwesen und Geodäsie, Dissertation*.

[Nelder & Mead, 1965] Nelder, J. A., & Mead, R. A. 1965. A Simplex Method for Function Minimization. *Computer Journal*.

[Ostrowski, 1998] Ostrowski, M. W., R. Mehler und A. Leichtfuß. 1998. Dokumentation des Schmutzfrachtmodells SMUSI Version 4.0. *Institut für Wasserbau und Wasserwirtschaft,Technische Universität Darmstadt*.

[Papageorgiou & von Stryk, 2009] Papageorgiou, M., & von Stryk, O. 2009. *Optimierung: Statische, dynamische, stochastische Verfahren*. Springer.

[Rauch, 2000] Rauch, W., V. Krejci und W. Gujer. 2000. REBEKA - Ein Simulationsprogramm zur Abschätzung der Beeinträchtigung der Fliessgewässer durch Abwassereinleitungen aus der Siedlungsentwässerung bei Regenwetter. *EAWAG*.

[Rechenberg, 1994] Rechenberg, I. 1994. *Evolutionsstrategie '94*. Frommann-Holzboog.

[Schütze, 1998] Schütze, M. 1998. Integrated Simulation and Optimum Control of the Urban Wastewater System. *Department of Civil Engineering. Imperial College of Science, Technology and Medicine. Dissertation.*

[Staub, 1997] Staub, Thomas. 1997. Optimierung mit genetischen Algorithmen Theoretische Grundlagen und Vergleiche. *Fachhochschule Trier, Fachbereich Angewandte Informatik, Seminararbeit*.

[Syrjakow, 2005] Syrjakow, Michael. 2005. Simulationstechnik. *Universität Karlsruhe, Institut für Rechnerentwurf und Fehlertoleranz, Vorlesungsskript*.

[Wikipedia, 2009a] Wikipedia. 2009a. *Bergsteigeralgorithmus*. http://de.wikipedia.org/wiki/Bergsteigeralgorithmus.

[Wikipedia, 2009b] Wikipedia. 2009b. *Downhill-Simplex-Verfahren*. http://de.wikipedia.org/wiki/Downhill-Simplex-Verfahren.

[Wikipedia, 2009c] Wikipedia. 2009c. *Intervallhalbierungsverfahren*. http://de.wikipedia.org/wiki/Intervallhalbierungsverfahren.

[Wikipedia, 2009d] Wikipedia. 2009d. *Sekantenverfahren*. http://de.wikipedia.org/wiki/Sekantenverfahren.

[WOLFGANG RAUCH, 1999] WOLFGANG RAUCH, POUL HARREMOEÈS. 1999. GENETIC ALGORITHMS IN REAL TIME CONTROL APPLIED TO MINIMIZE TRANSIENT POLLUTION FROM URBAN WASTEWATER SYSTEMS. *Pergamon / Elsevier Science Ltd.*