Metaheuristiken für die mehrkriterielle Optimierung

Endbericht der Projektgruppe 447

Universität Dortmund

31. Januar 2005

Teilnehmer:
Nicola Beume, Matthias Brinkmann, Torsten Lickfeld, Michael Janas, Ralf Kosse, Jens Kossek, Hendrik Noot, Miriam Padberg, Daniel Saltmann, Benedikt Schultebraucks, Kay Thielmann, Igor Tipura

Betreuer:
Thomas Bartz-Beielstein, Jörn Mehnen, Tim Richard, Karlheinz Schmitt

Folgende Tools wurden entwickelt, um das Zusammenspiel der Module zu verbessern oder überhaupt erst zu ermöglichen.

0.0.1 Design

0.0.1.1 Motivation

Mit Hilfe des Design-Moduls kann der Anwender einen Satz von vielen Algorithmen-Aufrufen mit systematisch variierten Parametern veranlassen. Dies ist hilfreich, weil für die Erforschung des Verhaltens randomisierter Heuristiken viele Experimente notwendig sind, da allgemeine theoretische Ergebnisse schwierig zu erlangen sind. Insbesondere bei den natur-inspirierten Verfahren, die oft viele Parameter enthalten, ist es schwierig, Algorithmen zu vergleichen, denn die Qualität der Optimierung hängt – neben dem Problem, auf das sie angewendet werden – auch stark von der gewählten Parameterbelegung ab. Die Problematik, ein geeignetes Qualitätsmaß für eine von einem Algorithmus produzierte Lösungsmenge zu finden, soll durch die Metriken gelöst werden (siehe **??**). Hier soll nun durch das Design der Experimente ein faires Szenario geschaffen werden, damit der tatsächlich beste Algorithmus auch die besten Ergebnisse liefern kann. Dazu muss man zunächst herausfinden, wie die beste Parametereinstellung für einen Algorithmus sind. Will man zwei Algorithmen vergleichen, muss man die Testfunktion bzw. das zu optimierende Problem und die zur Verfügung stehenden Ressourcen (z. B. Anzahl der Zielfunktionsauswertungen) geeignet wählen. Unter den Ergebnissen der via Parametereinstellung variierten Algorithmen kann die günstigste Einstellung ermittelt werden. Zusätzlich können durch eingehende Analyse Zusammenhänge zwischen den Parametern aufgedeckt werden.

0.0.1.2 Syntax für die Benutzung

Das Design-Modul ermöglicht dem Anwender, Algorithmen mit verschiedenen Parametereinstellungen aufzurufen. Hierzu steht eine Syntax zur Verfügung, mit der Parameterbelegungen als Mengen oder in Form von Intervallen angegeben werden können. Bei der Intervallschreibweise kann zwischen der Angabe eines Abstandes der Elemente und der Anzahl der Elemente gewählt werden.
Diese Syntax ist Teil des Aufrufs des Design-Moduls und so kann durch einen Aufruf ein ganzer Satz von Experimenten angestoßen werden.
Bei jedem Aufruf schreibt der benutzte Algorithmus seine Ergebnisse in eine Datei, die vom Design-Modul mit einem Namen versehen wird, der seine Parameter-Belegung wiedergibt. Durch Auslesen dieses Namens kann dann das Analyse-Modul die Parameter-Belegung nachvollziehen (siehe Kapitel **??**).
Beispiel-Aufruf:

```
java design.Design --t testfkt.FonsecaF1 --a OnePlusOnePG
-g s 100 1000 -archive i 10 100 step 10 -deviation i 0.1 1
count 5
```

Dieser Aufruf würde dann den Algorithmus `OnePlusOnePG` auf der Testfunktion `FonsecaF1` aufrufen und die Parameter des Algorithmus entsprechend variieren, so dass `g` die Werte 100 und 1000, `archive` die Werte von 10 bis 100 in Zehnerschritten und `deviation` insgesamt 5 Werte zwischen 0.1 und 1 annimmt.

0.0.1.3 Anwendung im Projekt

Das Design-Modul wurde im zweiten PG-Semester genutzt, um das Verhalten von Algorithmen bei bestimmten Parametereinstellungen zu analysieren und so allgemeine Erkenntnisse zu gewinnen, die vielleicht die Verbesserung von Verfahren ermöglichen. Erweiterungen zu diesem Tool waren geplant, wurden aber nicht mehr, oder nur indirekt, umgesetzt.

- Planung der Experimente
  Für das Factorial-Design wurde diese Klasse nicht mehr erweitert sondern eine zweite eigene Klasse geschrieben.

- Constraints
  Da nicht alle Parameterkombinationen, die der Benutzer wünscht, auch von den Algorithmen realisiert werden können, sollte ein Constraint-System eingeführt werden um zu überprüfen, ob die Parametereinstellung für das entsprechende Verfahren gültig ist. Eine einfaches Beispiel für eine Bedingung bei einer (μ, κ, λ)-ES wäre: $\mu \leq \lambda$, falls $\kappa = 1$.
  Die Notwendigkeit diese Erweiterung umzusetzen hat sich nicht ergeben, da es sich bei dem oben genannten Beispiel bis jetzt um das einzige handelt. Verletzungen solcher Constraints liegen dabei wohl auch meistens in Bereichen, deren Untersuchung nicht interessant ist, da dort keine guten Werte vermutet werden.

- Batch-Modus
  Da die Experimente unter Umständen sehr rechenintensiv sind, war geplant, das Design-Modul auch im Batch-Modus ausführbar zu machen.
  Diese Funktionalität musste nicht von der PG selbst umgesetzt werden, da am Lehrstuhl 11 bereits ein Batchsystem existiert, dem man solche Aufgaben übertragen kann. Dieses Batch-System wurde für die von uns ausgeführten Experimente ausführlichst genutzt.

0.0.2 Die Tool-Klassen

Viele Algorithmen nutzen gleiche Mechanismen. So wird z.B. jeder Algorithmus eine Dateiausgabe benötigen oder die bereits bewerteten Individuen bzgl. Pareto-Dominanz untersuchen.
Um hier unnötige Fehlerquellen auszuschließen und doppelte Arbeit zu vermeiden, haben wir bereits zu Beginn unseres Projekts ein eigenes Paket namens `tools` angelegt, in dem wir vermeintlich wiederverwertbare Teile für die Algorithmen, sowie allgemeine Hilfsklassen gesammelt haben. Dadurch ist eine Sammlung entstanden deren Funktionen im Folgenden näher erklärt werden sollen.

0.0.2.1 Allgemeine Hilfsklassen

Individual Fast sämtliche der von uns implementierten Algorithmen arbeiten in irgendeiner Weise mit Populationen von Individuen. Diese bestehen meistens aus einer Liste von Objektvariablen, den dazugehörigen Fitnesswerten und eventuellen Zusatzdaten, wie z.B. dem Alter des Individuums in Form der Generation oder einer Liste von Strategievariablen.

Diese Daten wurden von uns gekapselt und mit einfachen Zugriffsfunktionen versehen, welche die komfortable Einbindung in die Algorithmen erleichtert. Zusätzlich arbeiten viele weitere Hilfsklassen, wie z.B. das `ParetoArchive`, auch mit dieser Darstellung eines Individuums, so dass auf diese Weise eine einheitliche Kommunikationsschnittstelle zwischen den Klassen gewährleistet ist.

ParetoArchive Die Klasse `ParetoArchive` bietet diverse statische Methoden, um unser implementiertes Pareto-Archive zu aktualisieren bzw. zu verwalten. Algorithmen zur mehrkriteriellen Optimierung arbeiten oft mit Pareto-Archiven und müssen diese häufig um zusätzliche Individuen erweitern oder überprüfen, ob die Menge weiterhin nur aus pareto-optimalen Elementen besteht.
Sämtliche in dieser Klasse implementierten Methoden arbeiten auf Listen von Individuen, welche die Form der vorher vorgestellten Individual-Klasse haben müssen.

ParetoDomination Die zwei statischen Methoden, die mit dieser Klasse zur Verfügung gestellt werden, dienen der Überprüfung der Pareto-Dominanz.
Die erste Methode überprüft, ob das zweite übergebene Individuum von dem ersten dominiert wird und gibt das Ergebnis als Boolean-Wert zurück, während die zweite Methode die genaue Art der Paretodominanz als Integer zurückgibt.
Als Rueckgabe gibt es folgende Möglichkeiten:

- Integer 4: Wert, der ausgegeben wird, wenn die Individuen nicht miteinander vergleichbar sind.
- Integer 3 : Wert, der ausgegeben wird, wenn beide Individuen gleiche f(x)-Werte haben.
- Integer 2 : Wert, der ausgegeben wird, wenn Ind1 Ind2 dominiert, d.h. an mindestens einem Funktionswert einen besseren Wert hat.
- Integer 1: Wert, der ausgegeben wird, wenn Ind2 Ind1 dominiert.

RandomPG Die Erweiterung der Klasse `java.util.Random` durch die Klasse `RandomPG`, dient hauptsächlich dem Komfort des Programmierers. Der Zugriff auf zufällige gleichverteilte Double-Werte wurde über eine zusätzlich Methode erweitert. Hierbei können jetzt direkt gewünschte Schranken angegeben werden.
Um während der Experimente den Zufall besser kontrollieren zu können, und auch zufällig auftretende Fehler beim Debugging leichter reproduzieren zu können, wurde zusätzlich ein leichterer Zugriff auf den aktuellen Random-Seed ermöglicht.

0.0.2.2 I/O-Tools

Im Speziellen gab es bei der Entwicklung unseres Projekts viele Stellen, an denen sich mit der Ein- und Ausgabe beschäftigt werden musste. Um hier einheitlich und damit untereinander kompatibel zu bleiben, wurden die hierfür benötigen Klassen ins Paket `tools.io` ausgelagert und von allen Algorithmen gemeinschaftlich genutzt.

AlgoOutput.java Durch die hier zur Verfügung gestellten statischen Methoden können die Ausgabedateien für die Fitnesswerte und die Objektvariablen der Individuen erstellt werden. Diese Dateien enthalten die Daten in einem gut weiterverarbeitbaren

Format und können mit Hilfe einer weiteren Methode mit einem vom Datum abhängigen Dateinamen versehen werden.

ArgsInputBeautifier.java Die Aufrufe der Algorithmen werden häufig mit sehr langen Parameterlisten gestartet. Um nun einen schnellen und einfachen Zugriff auf diese Parameterliste zu haben, wandelt die Methode `beautify()` ein String-Array mit Parametern in eine stringbasierte Hashmap (`java.util.Properties`) um, so dass jeder Parameterwert einfach über seinen Namen abrufbar ist. Weiterhin kann über diese Klasse ein direktes Typecasting auf die als String übergebenen Werte erfolgen. Bei Bedarf können die genutzten Parameter in Form einer Textdatei gespeichert werden, die den Algorithmen wieder als Parameterdatei übergeben werden kann. Somit können häufig genutzte Einstellungen ohne großen Aufwand gesichert und wiederverwendet werden.
Anstatt der Klasse `Properties` kann hier auch die im `tools.io` Paket mitgelieferte Klasse `PropertiesPG` genutzt werden, die die Klasse `Properties` um einige Funktionalitäten erweitert. Bei den Zugriffen auf die Parameter können via `PropertiesPG` auch Default-Werte eingestellt und überprüft werden, ob die Angabe dieses Parameters zwingend notwendig ist.

FileParser.java Mit dieser Klasse können Dateien in dem Format, das auch von der Klasse `AlgoOutput` genutzt wird, wieder eingelesen werden. Die Datei wird in Form einer Liste von ArrayListen zurückgegeben, wobei jede ArrayList eine Zeile der Datei darstellt.

FilterAndDelete.java Diese Klasse hat sich im wesentlichen aus zwei Gründen entwickelt. Da wir schon während der Entwicklungsphase einige Experimente gestartet haben, hat sich an einigen Stellen ein inkompatibles Zahlenformat gezeigt. Um solche Daten trotzdem hinterher nutzen zu können, mussten sie gefiltert und gegebenfalls konvertiert werden. Unser Projekt arbeitet ausschließlich mit einem Punkt als Dezimaltrennzeichen. Die Tausenderstellen werden nicht durch ein Trennzeichen markiert. Also sollten in den zu bearbeitenden Dateien keine Kommata vorkommen.
Außerdem werden beim Temperierbohrungsproblem (siehe **??**) teilweise ungültige Individuen erzeugt, deren Punkte z. B. außerhalb der Gussform liegen, oder die durch das zu gießende Werkstück gehen. Diese Bohrungen könnten das Ergebniss verfälschen und sollten somit ausgefiltert werden. Leider war es nicht möglich die Ergebnisse aus dem Temperierbohrungsproblem bezüglich aller 12 Fitnesswerte zu interpretieren, da die einzige Metrik, die von uns für mehr als 2-dimensionale Fitnesswerte implementiert wurde, leider eine zu hohe Laufzeit hat.
Da diese Klasse teilweise Daten ohne Rückfrage und Sicherheitskopie löscht, ist es wichtig, vorher eine eigene Sicherheitskopie anzulegen.
Aus den genannten Problemen ergaben sich folgende Funktionen:

- `-d`
  Angabe des zu bearbeitenden Verzeichnisses.
- `-del_Comma`

Entfernt sämtlich Kommata! Das ist bei Zahlen im Format 1,234.567 sinnvoll.

- `-del_Invalid`
  Entfernt ungültige Individuen vom Temperierbohrungsproblem und ist somit für sämtliche anderen Probleme nicht interessant. Ungültig heißt in diesem Fall, dass in der dritten Spalte ein Wert größer als eins oder in den Spalten acht bis zwölf ein Wert größer als null steht.
- `-convert point` oder `-convert comma`
  Es werden alle Kommata zu Punkten oder bei Verwendung des Parameters `point` alle Punkte zu Kommata konvertiert (evtl. sollte zuvor erst `del_Comma` ausgeführt werden).
- `-select 1,2,3,6`
  Aus den Fitnesswerten werden die angegebenen Spalten gelöscht. Es gibt keine Überprüfung, ob die Spalten existieren. Die erste Spalte hat die Nummer eins.
- `-undo`
  Kopiert alle `.old`-Dateien zurück zu ihrem ursprünglichen Dateinamen. Natürlich ist das nur ein echtes „Undo“ wenn in den `*.old` noch die unveränderten Daten stehen.

SortFile.java `SortFile` sortiert ein Array von Dateien (`fileArray`) nach den Werten des per `sortingParameter` angegebenen Parameternamens. Der angegebene Parameter (z. B. g=100, g=200, g=300, ...), der im Dateinamen jeweils vorkommen muss, wird im Dateinamen gefolgt von seinem Wert angegeben. Diese Werte beeinflussen nun die Sortierung. Kommen mehrere Dateinamen mit denselben Werten dieses Parameters vor, so wird deren Sortierung untereinander nicht beeinflusst. Der zugrundeliegende Sortieralgorithmus ist also stabil.
Diese Funktion wird hauptsächlich bei der Visualisierung durch Plotter eingesetzt, da dort die zufällige Reihenfolge der Dateien auf der Festplatte für eine Visualisierung nicht geeignet ist. Eine Sortierung ist hier notwendig, um die entstehenden Grafiken gut lesbar zu machen.

Help.java Diese Klasse ruft nur die Methode `toString()` der angegebenen Klasse auf und gibt dessen Rückgabe auf der Konsole aus. Auf diese Weise können zu fast jeder Klasse Hilfen abgerufen werden.
Dazu muss die Klasse beim Aufruf mit Paketnamen angegeben werden, also mit Punkt-Trennung zwischen Paketen, z. B.: `java tools/io/Help algo.MueRhoLES`.
Die Hilfen enthalten z. B. bei den Algorithmen die korrekten Aufrufe sowie Beispielparameter in sinnvollen Bereichen.