jMole

A Genetic Approach to the Australia-Problem

Jochen Christ  
Daniel Kimmig  
Benjamin Kühnlenz

Inhalt

[Einleitung 3](#_Toc174676405)

[Das Australia-Problem 3](#_Toc174676406)

[Der Algorithmus 3](#_Toc174676407)

[Der Algorithmus im Überblick 3](#_Toc174676408)

[Die Codierung 3](#_Toc174676409)

[Der Strafkosten-Ansatz 3](#_Toc174676410)

[Das Erzeugen der Startpopulation 3](#_Toc174676411)

[Die Selektion 3](#_Toc174676412)

[Die Rekombination 3](#_Toc174676413)

[Die Mutation 3](#_Toc174676414)

[Optimierung der Ergebnisqualität 3](#_Toc174676415)

[Performance 3](#_Toc174676416)

[Die GUI 3](#_Toc174676417)

[Die GUI 3](#_Toc174676418)

[Ergebniswerte 3](#_Toc174676419)

[Ergebnisse Holmberg 3](#_Toc174676420)

[Ergebnisse Boccia 3](#_Toc174676421)

[Verhalten bei Änderungen 3](#_Toc174676422)

[Änderung der Strafkosten 3](#_Toc174676423)

[Änderung der Populationsgröße 3](#_Toc174676424)

[Änderung der Generationenanzahl 3](#_Toc174676425)

[Änderung der Selektionsmethode 3](#_Toc174676426)

[Lizenz 3](#_Toc174676427)

[LGPL 3](#_Toc174676428)

# Einleitung

Jochen

jMole ist eine tolle Sache. Wir können hier jetzt tolle sachen hin schreiben. Wenn wir wollen zumindest.

Ein neuer Absatz steht hier.

Hier steht dann auch ein bisschen was dazu woher der Name jMole kommt und was denn das Australia-Problem ist...

# Das Australia-Problem

Beschreibung des Problems.

Der Algorithmus

Im Folgenden wird der Algorithmus und die Details vorgestellt.

# Der Algorithmus im Überblick

Die Genetische Algorithmus

# Die Codierung

Nachdem im vorherigen Abschnitt der Algorithmus vorgestellt wurde, steht im folgenden Abschnitt die Codierung im Vordergrund. Eine Codierung dient dazu, Informationen zu formulieren. Ein Beispiel aus der Biologie ist die Codierung von Informationen in den Genen eines Lebewesens, die zur Bildung von spezifischen Proteinen herangezogen werden.

Auch beim Australia-Problem existieren Informationen, die mittels einer geeigneten Codierung formuliert werden müssen, damit sie maschinenlesbar werden und entsprechend verarbeitet werden können. Hierbei sticht ganz besonders die Information heraus, welcher Kunde von welchem Lager versorgt werden soll. Durch die Auswahl einer möglichen Technik zur Codierung ergeben sich Vor- und Nachteile für beispielsweise die Verständlichkeit und Komplexität, aber auch für den Aufwand zur Implementierung von Operatoren für bspw. die Rekombination. Die folgende Abbildung zeigt ein Beispiel für die gewählte Codierung, um die Zuordnung von Kunden zu Lagern ausdrücken zu können.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5** | **2** | **4** | **1** | **2** | **5** | **3** | **7** | **7** | **2** | **…** |  |  |  |  |  | **m** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | … |  |  |  |  |  | n |

dHierbei wird ein Feld (Array) veranschaulicht, dass dem Index n die verschiedenen Werte m zuordnet. Der Index steht bei der gewählten Codierung für die Kunden und ergibt sich aus den Werten der jeweiligen Probleminstanz. Der Wert an der n-ten Stelle entspricht dem Lager, zu dem der Kunde n zugeordnet wird. Auf diese relativ simple Art und Weise kann Konsistenz sichergestellt werden, d.h., dass ein Kunde immer zu einem Lager zugeordnet ist.

# Der Strafkosten-Ansatz

as

# Das Erzeugen der Startpopulation

benni

Greedy, A bit Greety and Random

# Die Selektion

jc

Eltern auswählen oder Kinder aussortieren…

# Die Rekombination

jc

lalala

# Die Mutation

jc

simple mutation, nearNeighbor

# Optimierung der Ergebnisqualität

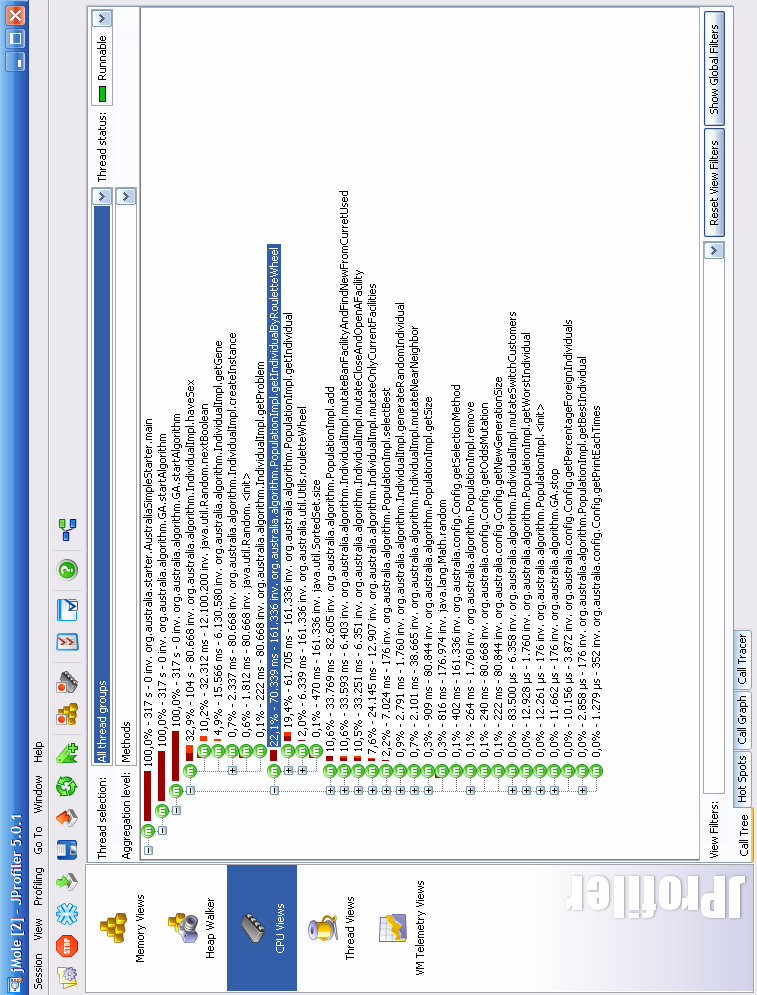
jc

Verwendung der Datenbank und Einspielen bekannter Optima.

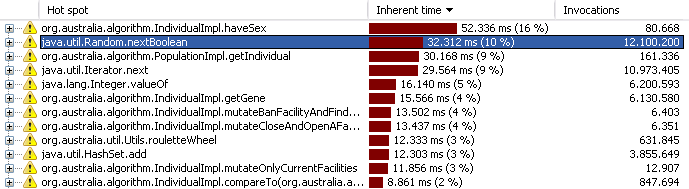
# Performance

Im Bereich Software Engineering ist Performance ein zentraler Begriff und kann daher in vielen Anforderungsspezifikationen wieder-gefunden werden. Darin wird die Performanz eines Systems oft mit quantifizierbaren Messgrößen wie Zeitverbrauch oder einem bestimmten Durchsatz beschrieben. Diesen Zielvorstellungen steht die Notwendigkeit gegenüber, die Leistungsfähigkeit eines Systems zu analysieren, wodurch detailierte Informationen über bspw. mögliche Flaschenhälse gewonnen werden können. Anhand dieser Informationen kann man Veränderungen am System vornehmen, die evtl. zu einer Leistungssteigerung führen. Ist dies nicht ausreichend, können die Ergebnisse der Leistungsmessung als Grundlage für einen neuen konzeptionellen Entwurf des Systems dienen.

Gerade bei Algorithmen, wie dem vorliegenden Algorithmus zur Lösung des Australia-Problems, die keine direkte Abhängigkeit zu Drittsystemen wie bspw. Datenbanken haben, und nicht durch sich wiederholende I/O-Prozesse in ihrer Leistungsfähigkeit beschränkt sind, ist es ratsam eine Leistungsmessung durchzuführen. Der Grund dafür liegt in der Tatsache, dass bis auf die Kerneinheiten des Rechners (Bsp.: CPU) keine Abhängigkeit zur Leistungsfähigkeit der Umgebung besteht und dadurch ein großes Potential für Optimierungen in der Konzeption des Algorithmus gegeben ist. Um aber sinnvolle Entscheidungen treffen zu können bedarf es einem soliden Werkzeug zur Leistungsmessung, da ansonsten die Gefahr besteht, dass die gewonnen Informationen falsch sind bzw. die Leistung nur in geringem Maße beeinflussen. Für Java-Applikationen hat sich solch ein Werkzeug, der sog. JProfiler, bereits etabliert und findet breite Anwendung in der Praxis. Die folgende Abbildung zeigt einen Auschnitt aus der Benutzerschnittstelle von JProfiler bei der Leistungsmessung von jMole.



In der Abbildung wird veranschaulicht, wie mittels JProfiler das Laufzeitverhalten von jMole analysiert werden kann. Von besonderem Interesse sind dabei die Funktionen, die entweder am längsten dauern oder am häufigsten aufgerufen werden. Daraus lassen sich am deutlichsten Bereiche erkennen, die sich positiv auf die Leistung auswirken. JProfiler unterstützt dies durch eine Auflistung dieser sogenannten „Hot spots“, also kritischen Bereichen der Applikation. Die folgende Abbildung zeigt ein Beispiel aus der Benutzerschnittstelle von JProfiler zur Auflistung von Hot-spots aus jMole.



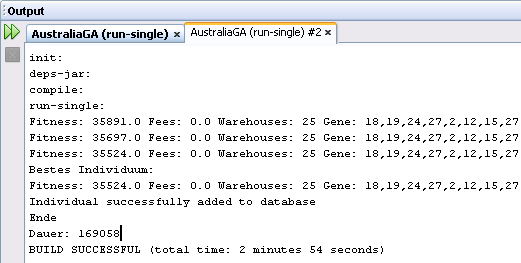
Durch diese Information konnte jMole schrittweise optimiert werden, da sich Fragen über das Konzept und die Struktur des Algorithmus im Groben, sowie über Details der Implementierung einzelner Methoden nach jedem Durchlauf aufs Neue gestellt haben.

Hierfür lassen sich einige Beispiele aus der Entwicklung von jMole nennen. Der zentrale Strafkosten-Ansatz hat sich ergeben, als durch JProfiler herausgefunden wurde, dass die Funktionen zur Garantie der Zulässigkeit der Individuen einen sehr schlechten Einfluss auf die Leistung hatten. Darüber hinaus haben sich Experimente ergeben bezüglich der Nutzung unterschiedlicher Collection-Klassen (Bsp.: TreeSet) aus dem Java Development Kit (JDK) um gewisse Zugriffsmodalitäten erfüllen zu können. Allerdings ist den verschiedenen Gedankengängen zur Optimierung an gewissen Stellen eine Grenze gesetzt. Beispielsweise finden sich weit oben in der Rangliste der häufigst aufgerufenen bzw. der zeitlich längsten Funktionen einige Utility-Methoden des JDKs, auf die der Algorithmus letztlich angewiesen ist und nicht „weg-optimiert“ werden können.

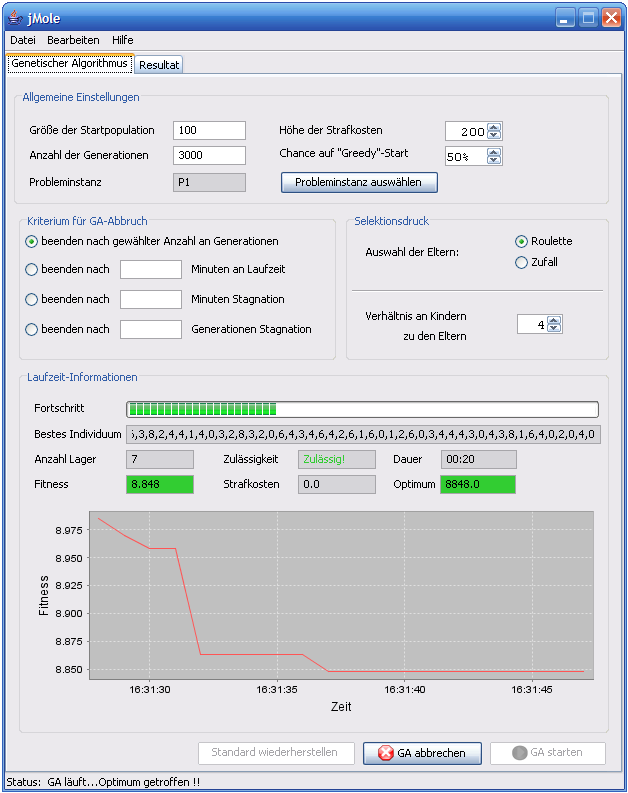
Die GUI

# Die GUI

Für die Benutzung von jMole stehen grundsätzlich zwei Möglichkeiten zur Verfügung. Die erste Möglichkeit ist der Aufruf von einer der vielen Starter-Klassen über die Kommandozeile. Dabei erfolgt periodisch nach einer konfigurierten Anzahl an Generationen eine Ausgabe, bei der die Fitness des aktuell besten Individuums der Population, die Höhe der Strafkosten und damit auch die Zulässigkeit, die Anzahl der verwendeten Lager und der detailierte Gen-String sowie Informationen über die Fixkosten der Lager für diese Zuordnung ausgegeben. Die folgende Abbildung zeigt ein Beispiel für die Nutzung von jMole über die Kommandozeile.



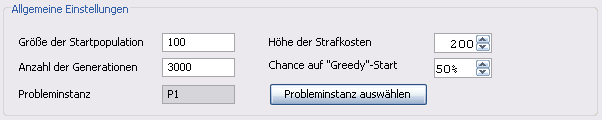
Da die Starter-Klassen einige Abhängigkeiten zur Implementierung des Algorithmus haben, wurde hierfür eine Entwicklungsumgebung benutzt, die die aufwändige Konfiguration des Classpath übernimmt. Eine zweite Möglichkeit zur Nutzung von jMole besteht in der Swing basierten Benutzerschnittstelle, die in der folgenden Abbildung gezeigt und danach genauer erläutert wird.



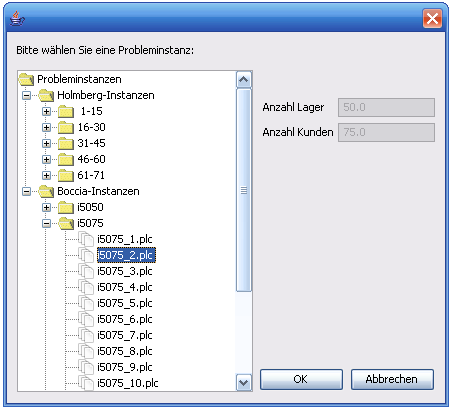
Die Abbildung zeigt die Benutzerschnittstelle von jMole, die sich dem Benutzer während dem Durchlaufen des Algorithmus bietet. Dabei gliedert sich die Ansicht in drei Bereiche, erstens die Konfiguration von allgemeinen Einstellungen, zweitens die Auswahl von Abbruchkriterium und Selektionsmethoden und drittens die Anzeige von Informationen zur Laufzeit des Algorithmus wie ein Fortschrittsbalken und ein Diagramm zur Visualisierung der Fitness. Die Benutzung dieser drei Bereiche wird im Folgenden genauer dargestellt.

##### Konfiguration der allgemeinen Einstellungen

Für die allgemeinen Einstellungen sind die gründsätzlichen variablen Werte für jMole von Bedeutung. Dies sind die Größe der Startpopulation, die Anzahl zu erzeugender Generationen, die Probleminstanz, die Höhe der Strafkosten für unzulässige Individuen und die Chance auf eine nicht zufällige, sondern rangfolgebasierte Generierung der Startpopulation. Dies spiegelt sich im folgenden Ausschnitt aus der Benutzerschnittstelle von jMole wieder.



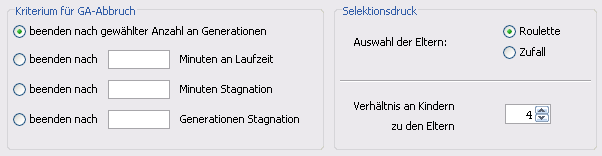
Für die Auswahl der Probleminstanz steht ein eigener Dialog zur Verfügung, um die sonst fehlerträchtige Eingabe zu erleichtern. Dieser erscheint nach Betätigung des Buttons „Probleminstanz auswählen“ und ist modal zum Hauptfenster. Die folgende Abbildung zeigt dieses Dialogfenster.



Um bei der großen Menge an Probleminstanzen nicht den Überblick zu verlieren, wurden sowohl die Holmberg- als auch die Boccia-Probleminstanzen in eine Baumstruktur geordnet, sodass man sich schneller zu einer spezifischen Instanz vorarbeiten kann. Um den Typ der Instanz einordnen zu können wird die Anzahl an Lager und Kunden nach Auswahl der Instanz in den Textfeldern auf der rechten Seite angezeigt. Abschließend existieren noch Buttons zum Bestätigen der Auswahl sowie zum Abbruch des Dialogs, die den Benutzer zum Hauptfenster zurückführen.

##### Auswahl von Abbruchkriterium und Selektionsmethoden

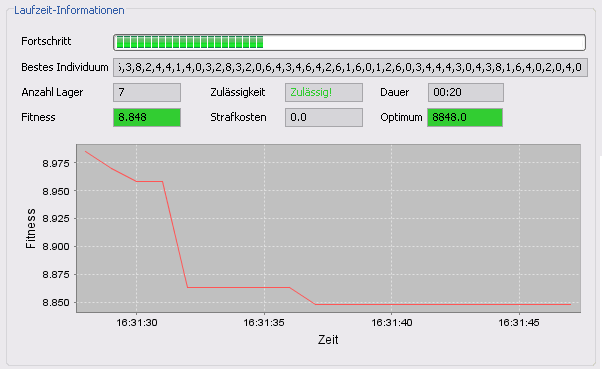
Neben den allgemeinen Einstellungen müssen noch erweiterte Einstellungen wie das Abbruchkriterium für den Algorithmus und die zu verwendenden Selektionsmethoden ausgewählt werden. Es stehen vier Abbruchkriterien zur Auswahl, wobei sich zwei davon auf die Laufzeit beziehen und die anderen zwei auf das Verhalten des Algorithmus bei der Stagnation der Fitness-Werte. Dies wird im folgenden Ausschnitt aus der Benutzerschnittstelle aufgezeigt.



Der Standard-Wert für das Abbruchkriterium ist die gewählte Anzahl an Generationen. Darüber hinaus kann auch eine fixe Laufzeit in Minuten gewählt werden. Bei dieser Einstellung bricht der Algorithmus nach dieser Laufzeit ab ohne das beste Individuum oder den derzeitigen Stand der Population bzw. die Anzahl an erzeugten Generationen zu beachten. Darüber hinaus existieren noch Abbruchkriterien, die hilfreich sind, um den Algorithmus über lange Zeit auszuführen. Dies wird über die Technik der Betrachtung der Stagnation der Fitness-Werte erreicht, d.h., dass der Algorithmus ohne fixe Grenze laufen kann, bis keine besseren Individuen gefunden werden.

##### Anzeige von Informationen zur Laufzeit des Algorithmus

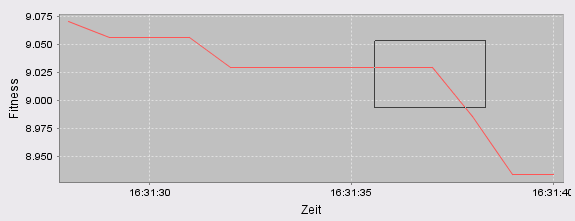
Nach erfolgreicher Konfiguration kann der Algorithmus über den Button „GA starten“ gestartet werden, sodass sich dem Benutzer folgende Informationen offenbaren.



Die Informationen zur Laufzeit beschreiben einerseits den Gen-String des besten Individuums, die Anzahl an benutzten Lagern, die Fitness, die Strafkosten und die Zulässigkeit des Individuums. Bei Holmberg-Probleminstanzen wird zusätzlich das erreichbare Optimum angezeigt und mit dem Feld für die Fitness grün eingefärbt, falls das Optimum getroffen wurde.

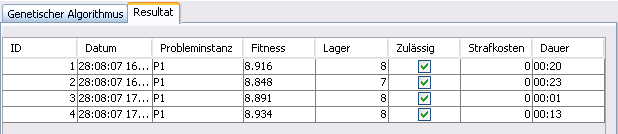
Andererseits ergeben sich Laufzeit-Informationen über den Fortschritt des Algorithmus und über die Dauer, d.h. wie lange der Algorithmus bereits an dieser Probleminstanz mit den gegebenen Einstellungen rechnet. Der Fortschrittsbalken ist nicht determiniert für den Fall, dass ein Kriterium bezüglich der Stagnation gewählt wurde, da hierbei ein Fortschritt nicht eindeutig berechnet werden kann. Die Eigenschaft „Nicht determiniert“ bedeutet in diesem Zusammenhang, dass der Balken sich von links nach rechts und wieder zurück bewegt und damit die Nicht-Berechenbarkeit besser zum Ausdruck kommt.

Als drittes Element dieses Bereichs wird ein Diagramm laufend gezeichnet, das die Entwicklung der Fitness des derzeit besten Individuums im Verhältnis zur Dauer der Berechnung aufzeigt. Durch Klicken auf das Diagramm und gleichzeitigem Festhalten der linken Maustaste kann man sich in das Diagramm hineinzoomen, um einen besseren Überblick zu erhalten.



Durch Betätigen der rechten Maustaste öffnet sich ein Kontexmenü, anhand dessen man ebenfalls Zoom-Vorgänge vornehmen kann. Das Gedrückt-Halten der rechten Maustaste und ein Zug nach links und das Loslassen der rechten Maustaste bringt das Diagramm wieder in seine Ausgangstellung.

Wenn der Algorithmus abgearbeitet wurde, wird das Resultat und einige zusätzliche Informationen wie das Datum des Durchlaufs in eine Tabelle geschrieben. Diese Tabelle erreicht man durch Wechseln auf den Tab „Resultat“, auf dem man einen Überblick über die unterschiedlichen Durchläufe erhalten kann. Diese Auflistung geht nach Beenden der Benutzerschnittstelle verloren. Eine automatische Exportfunktion ist für die nächste Version von jMole vorgesehen. Manuell kann man auch Einträge markieren und über Copy+Paste in ein anderes Programm wie Excel überführen.



##### Starten der Benutzerschnittstelle von jMole

Diesem Dokument liegt ein Datenträger bei, der den Quellcode für sämtliche Teilfunktionen und damit auch für die Benutzerschnittstelle enthält. Anhand dieses Quellcodes ist es möglich eine lauffähige Version zu kompilieren.

Dazu liegen noch einige vorgefertigte Distributionsformen bei. Einerseits ein JAR-Archiv das durch Doppelklick ausgeführt werden kann oder über den Befehl

java –jar jMole.jar

nach Navigieren in den jeweiligen Ordner von „jMole.jar“ ausgeführt werden kann.

Für das Betriebssystem Windows liegt außerdem eine ausführbare Datei „jMole.exe“ bei, die einfach durch Doppelklick ausgeführt werden kann.

Für tiefergreifende Fragestellungen und Supportanfragen steht das jMole-Team zur Verfügung.

Ergebniswerte

# Ergebnisse Holmberg

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instance** | **jMole** | **Computational Optima** | **Deviation (absolute)** | **Deviaton (relative)** |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |

# Ergebnisse Boccia

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instance** | **jMole** | **Computational Optima** | **Deviation (absolute)** | **Deviaton (relative)** |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |
| **p1** | 8848 | 8848 | 0 | 0.0 % |
| **p2** | 1234 | 1234 | 4 | 10.2 % |

Verhalten bei Änderungen

# Änderung der Strafkosten

jc

p43

# Änderung der Populationsgröße

a

# Änderung der Generationenanzahl

jc

a

# Änderung der Selektionsmethode

In diesem Abschnitt wird die Entwicklung der Fitness bei Veränderung der Selektionsmethode betrachtet. Unter Selektionsmethode versteht man dabei u.a. die Auswahl der Eltern für eine Rekombination. Aber auch die Größe der neu zu erzeugenden Individuen wird darunter gefasst. Hierfür werden die bereits in den vorherigen Untersuchungen beschriebenen Startparameter verwendet, um eine einheitliche Betrachtungsweise zu ermöglichen. Außerdem werden fünf Stichproben pro Veränderung der Selektionsmethode genommen, damit Ausreißer besser ausgeglichen werden können. Die einzigen Variablen in dieser Betrachtung sind ersten die Auswahl der Eltern („Roulette“, „Random“) und das Verhältnis an Kindern zu den Eltern (1-5). Die folgende Tabelle gibt einen Überblick über die erzielten Werte bei Veränderung der Selektion der Eltern.



Die Verteilung der Fitness-Werte aus der Tabelle wird im folgenden Diagramm veranschaulicht.



Bei vier der Stichproben war die Selektionsmethode „Roulette“ eindeutig besser, da jeweils niedrigere Fitness-Werte und damit niederigere Kosten erzielt wurden. Um die Werte zu glätten wurden bei der weiteren Berechnung für die fünf Stichproben der beiden Selektionsmethoden jeweils der durchschnittliche Mittlerwert und der Median berechnet. Die schlechtere Lösung wurde als Nullpunkt gewählt um die Differenz zur besseren Lösung besser aufzeigen zu können. Die absoluten Differenzen werden in der folgenden Abbildung aufgezeigt.



Dasselbe Verfahren wird im Folgenden auf die Selektionsmethode „Verhältnis an Kindern zu Eltern“ angewendet. Für die jeweiligen Faktoren eins bis fünf wurde der Algorithmus ausgeführt und jeweils fünf Stichproben genommen. Die Ergebnisse werden in der folgenden Tabelle aufgezeigt.



Der Verlauf der Fitness-Werte für die fünf verschiedenen Faktoren wird in der folgenden Abbildung veranschaulicht.



Anhand der einzelnen Datenreihen lässt sich die Aussage treffen, dass mit einem steigenden Verhältnis an Kindern zu Eltern eine niedrigerer Fitness-Wert und damit niedrigere Kosten für die Kunden-Lager-Zuordnung einhergehen. Wie im vorherigen Beispiels werden die Datenreihen anhand eines durchschnittlichen Mittelwertes und eines Medians bereinigt, um den Trend eindeutiger darstellen zu können.

Dieser Trend wird im folgenden Schaubild verdeutlicht, der diese beiden Werte der fünf Datenreihen darstellt.



Zusammenfassend lässt sich anhand der Stichproben sagen, dass eine „Roulette“ basierte Selektion der Eltern und eine größere Anzahl an zu erzeugenden Kindern für bessere Ergebnisse sorgen.

Lizenz

# LGPL

LGPG