

jMole

A Genetic Approach to the
Australia-Problem

Jochen Christ
Daniel Kimmig
Benjamin Kühnlenz

Inhalt

Einleitung.....	3
Das Australia-Problem.....	4
Der Algorithmus.....	5
Der Algorithmus im Überblick	6
Die Codierung	7
Der Strafkosten-Ansatz.....	8
Das Erzeugen der Startpopulation	9
Die Selektion.....	11
Die Rekombination	12
Die Mutation.....	13
Optimierung der Ergebnisqualität.....	15
Performance	16
Die GUI.....	19
Die GUI.....	20
Ergebniswerte.....	27
Ergebnisse Holmberg.....	28
Ergebnisse Boccia	30
Verhalten bei Änderungen	32
Änderung der Strafkosten	34
Änderung der Populationsgröße	35
Änderung der Generationenanzahl.....	36
Änderung der Selektionsmethode	37
Lizenz	39

Einleitung



jMole ist ein Algorithmus zur Lösung des Australia-Problems.

Das Australia-Problem heißt eigentlich *Single Source Capacitated Facility Location Problem*. Da der Name zu lang und zu kompliziert ist, sprechen wir entsprechend des Beispiels aus der Vorlesung vom **Australia-Problem**.

Dazu wird ein genetischer Algorithmus¹ verwendet. Da besonders an unserer Lösung ist die Verwendung des **Strafkostenansatzes**, der hochperformant sehr gute Lösungen findet.

Zur einfachen und schnellen Bedienung haben wir eine ansprechende **Oberfläche (GUI)** entwickelt.

Der Algorithmus wird auf den folgenden Seiten vorgestellt. Unsere gefundenen Optima sind ab Seite 28 dargestellt.

¹ Um genau zu sein, müsste man von einer Evolutionsstrategie sprechen, da wir eine ganzzahlige Codierung einsetzen. Aber diese Unterscheidung wird auch in der Literatur nicht so eng gesehen.

Das Australia-Problem



Das Australia-Problem beschreibt ein Transportproblem, welches verschiedene Lagerhäuser sowie verschiedene Kunden beinhaltet. Die Restriktionen sind:

- ein Kunde wird von genau einem Lagerhaus beliefert
- alle Kunden müssen beliefert werden

Stellt man sich Australien als Kontinent vor, welcher von der restlichen Welt abgeschnitten wäre. So gäbe es eine begrenzte Anzahl an Lagerhäusern sowie eine begrenzte Anzahl an Kunden.

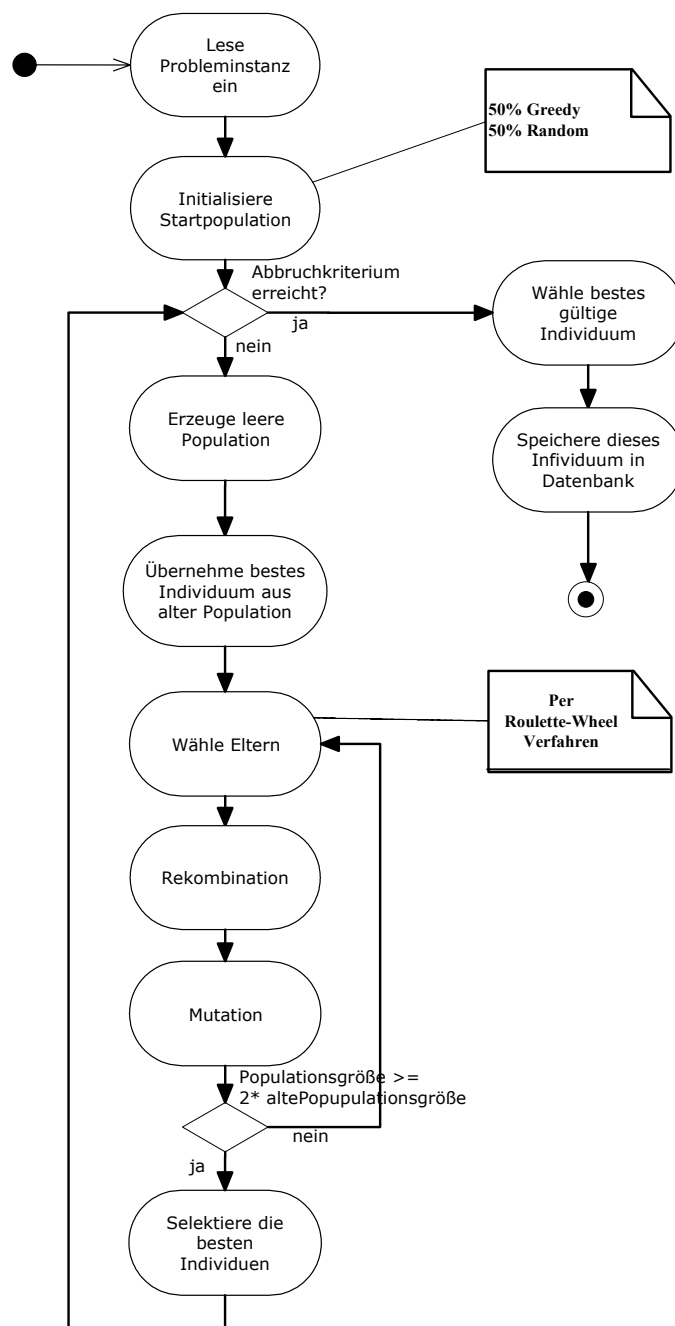
Grundsätzlich könnte jedes Lagerhaus in Australien jeden x-beliebigen Kunden beliefern, jedoch zu verschiedenen Kosten, da die Entfernungen variieren.

Die Aufgabe des Algorithmus ist es nun eine eventuell suboptimale Verteilung der Lagerhäuser zu den einzelnen Kunden zu finden sodass die Bedingungen erfüllt sind.

Der Algorithmus

Im Folgenden wird der Algorithmus und die Details vorgestellt.

Der Algorithmus im Überblick



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.

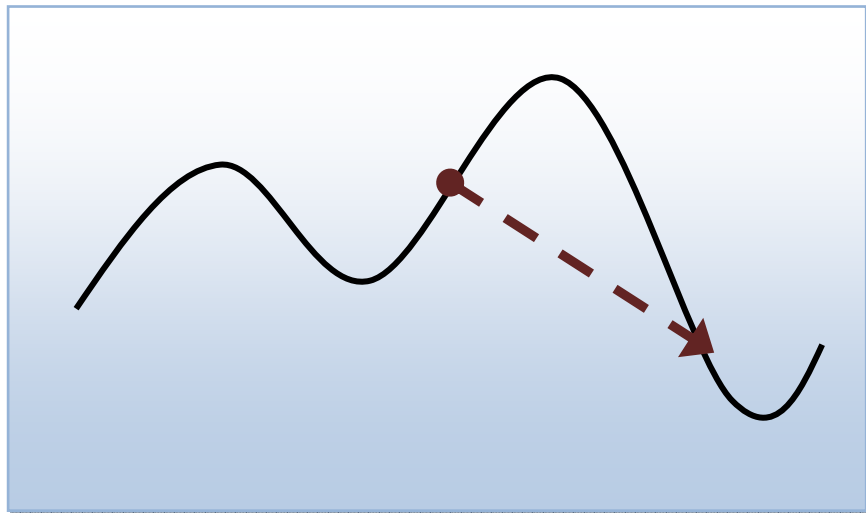
5	2	4	1	2	5	3	7	7	2	...						m
1	2	3	4	5	6	7	8	9	10	...						n

d

Hierbei 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 Problem Instanz. 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

Der Name jMole leitet sich aus unserem Strafkostenansatz ab. Wie ein **Maulwurf** versucht sich unser Algorithmus durch einen Berg zu graben, um bessere Lösungen zu finden.



Der entscheidende Vorteil, der sich dadurch ergibt, ist die Tatsache, dass so zum großen Teil vermieden werden kann, in einem lokalen Optimum stecken zu bleiben.

Gleichzeitig muss aber auch versucht werden, eine gültige Lösung zu finden. Dies erreichen wir dadurch, dass größere Entfernungen von einer gültigen Lösung weg stärker bestraft werden als umgekehrt. Unser Maulwurf wird auf diese Weise schnell zu einer gültigen Lösung gebracht.

Darüber hinaus liegt die Performance dieses Ansatzes weit über der Vorgehensweise nur gültige Lösung zu akzeptieren, da „Reparaturen“ und das „Wegschmeißen“ ungültiger Lösungen wegfällt.

Das Erzeugen der Startpopulation

Der Benutzer kann auf die Aufstellung der Startpopulation mit zwei Parametern Einfluss nehmen.

Die Parameter sind:

- Größe der Startpopulation
- Chance auf „Greedy“-Start

General Settings dialog box showing parameters for the start population and other settings.

Allgemeine Einstellungen	
Größe der Startpopulation	100
Anzahl der Generationen	100
Chance auf "Greedy"-Start	50%
Höhe der Strafkosten	10,5
Durchläufe	1
Problem Instanz	
<button>Auswählen</button>	

Größe der Startpopulation

Die Größe der Startpopulation gibt an wie viele zulässige Individuen erzeugt werden sollen. In einer Schleife werden solange verschiedene Individuen erzeugt bis die Anzahl der zulässigen Individuen der Größe der Startpopulation entspricht.


Chance auf „Greedy“-Start

Diese Kenngröße gibt an wie viele Individuen in der Startpopulation zufällig und wie viele Individuen nach einem bestimmten Algorithmus berechnet werden.

Funktionsweise des Algorithmus:

Als erstes stellt der Algorithmus eine interne Tabelle auf. Diese beinhaltet für jeden Kunden alle zuliefernden Lagerhäuser, aufsteigend nach den Transportkosten.

Kunde	Lagerhäuser				
1	12	24	2	5	...
2	25	40	1	18	...
...



Zu Beginn des Algorithmus darf ein zufällig gewählter Kunde das günstigste Lagerhaus auswählen. Bei diesem Lagerhaus wird der benötigte Bedarf des Kunden abgezogen. Anschließend darf der nächste zufällig ausgewählte Kunde dessen günstigstes Lagerhaus wählen. Besitzt dieses Lagerhaus nicht mehr ausreichend Kapazität wird das nächst beste Lagerhaus dieses Kunden aus der sortierten Tabelle ausgewählt.

Auf diese Weise werden so viele Individuen erzeugt, wie im Parameter angegeben wurden. Die restlichen Individuen werden zufällig erzeugt.

Wie sich dieser Parameter auf die Ergebnisqualität auswirkt wird in einem späteren Kapitel behandelt.

Die Selektion

Die Selektion, also der Faktor der bestimmt, welche Individuen überleben, kann über zwei Parameter gesteuert werden:

1. Auswahl der Eltern

Der Selektionsdruck kann bereits bei der Auswahl der Eltern erfolgen.

Eine Möglichkeit ist das rangbasierte Roulette-Wheel-Verfahren. Hier werden den Individuen mit einer besseren Fitness eine höhere Wahrscheinlichkeit zugeschrieben, als schlechteren.



2. Größe der Kinderpopulationen

Es können in der neuen Generation beliebig viele Kinder erzeugt werden. Da die Größe der Populationen über die Generationen hinweg gleich bleiben soll, können nur so viele Kinder überleben, wie es Individuen in der Elterngeneration gibt.

Je mehr Kinder erzeugt werden, desto größer ist der Selektionsdruck, da prozentual weniger Kinder überleben können.



Die Rekombination

Nach der Selektion der Eltern – ob per Zufall oder nach dem Roulette-Wheel-Verfahren, erfolgt das Zeugen eines Kindes.

Dies ist objektorientiert klar verständlich:

```
Individual baby = mum.haveSex(dad);
```

Während der Rekombination werden die Gene der Eltern zufällig vermischt. Wir haben uns entgegen der üblichen Spaltung der Gene in der Mitte für eine horizontale Rekombination entschieden:

Mum:	1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Dad:	4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 6, 5, 7, 8, 9, 1, 2, 3

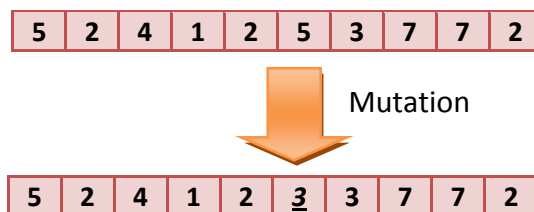
Baby:	1, 5, 6, 4, 8, 9, 0, 1, 9, 0, 1, 6, 3, 4, 8, 9, 7, 2, 9

Die Entscheidung, ob ein Lagerhaus von der Mutter oder vom Vater gewählt wird, überlassen wir jeweils dem Zufall.

Aus dem neuen Gen-String wird daraufhin eine Baby-Instanz erstellt.

Die Mutation

Wie auch in der Natur erfährt das Erbgut unserer neugeborenen Individuen mit einer bestimmten Wahrscheinlichkeit eine Veränderung im Erbgut.



Im einfachsten Falle wird das zugeordnete Lager eines zufälligen Kunden durch ein anderes (zufälliges) ersetzt. Im Beispiel wird den Kunde 6 nun das Lager 3 statt Lager 5 zugeordnet.

Hat das Individuum eine höhere Fitness hat es auch eine höhere Überlebenschance und führt zu einer Verbesserung der Lösung.

Diese Mutationsoperationen können nun zur Lösung des Problems noch weiter optimiert und kombiniert werden.

Folgende Operationen stehen in unserem Algorithmus zur Verfügung:

mutate()

Zufällige Mutation, wie oben beschrieben.

mutateOnlyCurrentFacilities();

Zufällige Mutation, aber das neue Lager wird nur aus den bereits verwendeten Lagern gewählt.

mutateNearNeighbor();

Eine Variante: das neue Lager soll sinnvollerweise in der Nähe des Kunden liegen. Dabei kommt das Roulette Wheel-Verfahren zum Einsatz.

mutateSwitchCustomers();

Eine weitere Möglichkeit der Mutation: Die Lager zweier Kunden werden getauscht.

mutateBanFacility();

Ein sehr mächtiger Mutationoperator, der nur sehr selten eingesetzt werden darf. Es wird ein zufälliges Lager gewählt, welches nicht mehr verwendet werden darf. Kunden mit diesem Lager wird ein anderes zufälliges zugeordnet.

Das Ziel ist dabei das Sparen der Fixkosten eines Lagers.

mutateBanFacilityAndFindNewFacilityByRouletteWheel();

Eine Erweiterung von mutateBanFacility: Den Kunden werden neue Lager zugeordnet, die in der Nähe liegen. Dabei wird das Roulette Wheel-Verfahren eingesetzt.

mutateBanFacilityAndFindNewFromCurrentUsed();

Eine weitere Variante ist das zuordnen der neuen Lager aus den Lagern, die bereits verwendet werden.

Optimierung der Ergebnisqualität

Alle berechneten Ergebnisse werden zusammen mit den verwendeten Parametern in einer zentralen MySQL-Datenbank gespeichert.

Wir haben eine Variante unseres Algorithmus entwickelt, welche aus einer Menge von bisher berechneten Ergebnissen ein sehr gutes Optimum aussucht und dieses als Referenzindividuum verwendet.

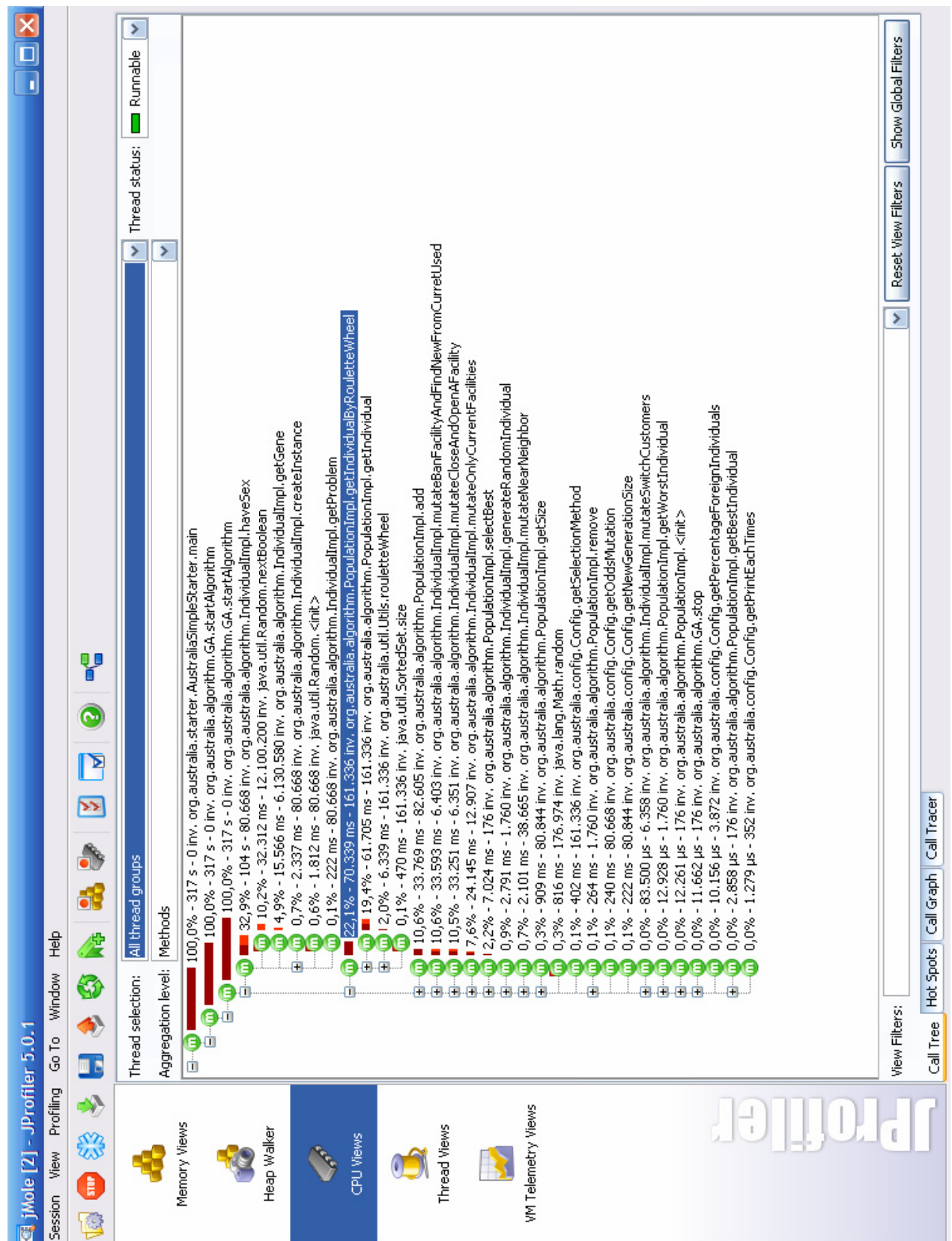
Im Anschluss wird das neu berechnete Optimum wieder in die Datenbank gespeichert.

Wird der Algorithmus iterativ durchgeführt kommt es zu einer kontinuierlichen Verbesserung der Ergebnismenge.

Performance

Im Bereich Software Engineering ist Performance ein zentraler Begriff und kann daher in vielen Anforderungsspezifikationen wiedergefunden 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 detaillierte 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 Ausschnitt 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.

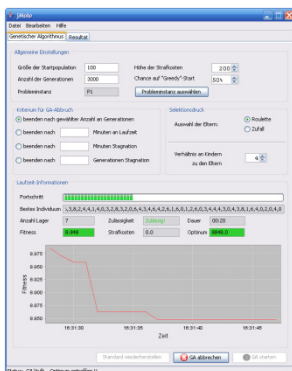
Hot spot	Inherent time ▼	Invocations
org.australia.algorithm.IndividualImpl.haveSex	52.336 ms (16 %)	80.668
java.util.Random.nextBoolean	32.312 ms (10 %)	12.100.200
org.australia.algorithm.PopulationImpl.getIndividual	30.168 ms (9 %)	161.336
java.util.Iterator.next	29.564 ms (9 %)	10.973.405
java.lang.Integer.valueOf	16.140 ms (5 %)	6.200.593
org.australia.algorithm.IndividualImpl.getGene	15.566 ms (4 %)	6.130.580
org.australia.algorithm.IndividualImpl.mutateBanFacilityAndFind...	13.502 ms (4 %)	6.403
org.australia.algorithm.IndividualImpl.mutateCloseAndOpenAFa...	13.437 ms (4 %)	6.351
org.australia.util.Utils.rouletteWheel	12.333 ms (3 %)	631.845
java.util.HashSet.add	12.303 ms (3 %)	3.855.649
org.australia.algorithm.IndividualImpl.mutateOnlyCurrentFacilities	11.856 ms (3 %)	12.907
org.australia.algorithm.IndividualImpl.compareTo(org.australia.a...	8.861 ms (2 %)	847.694

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

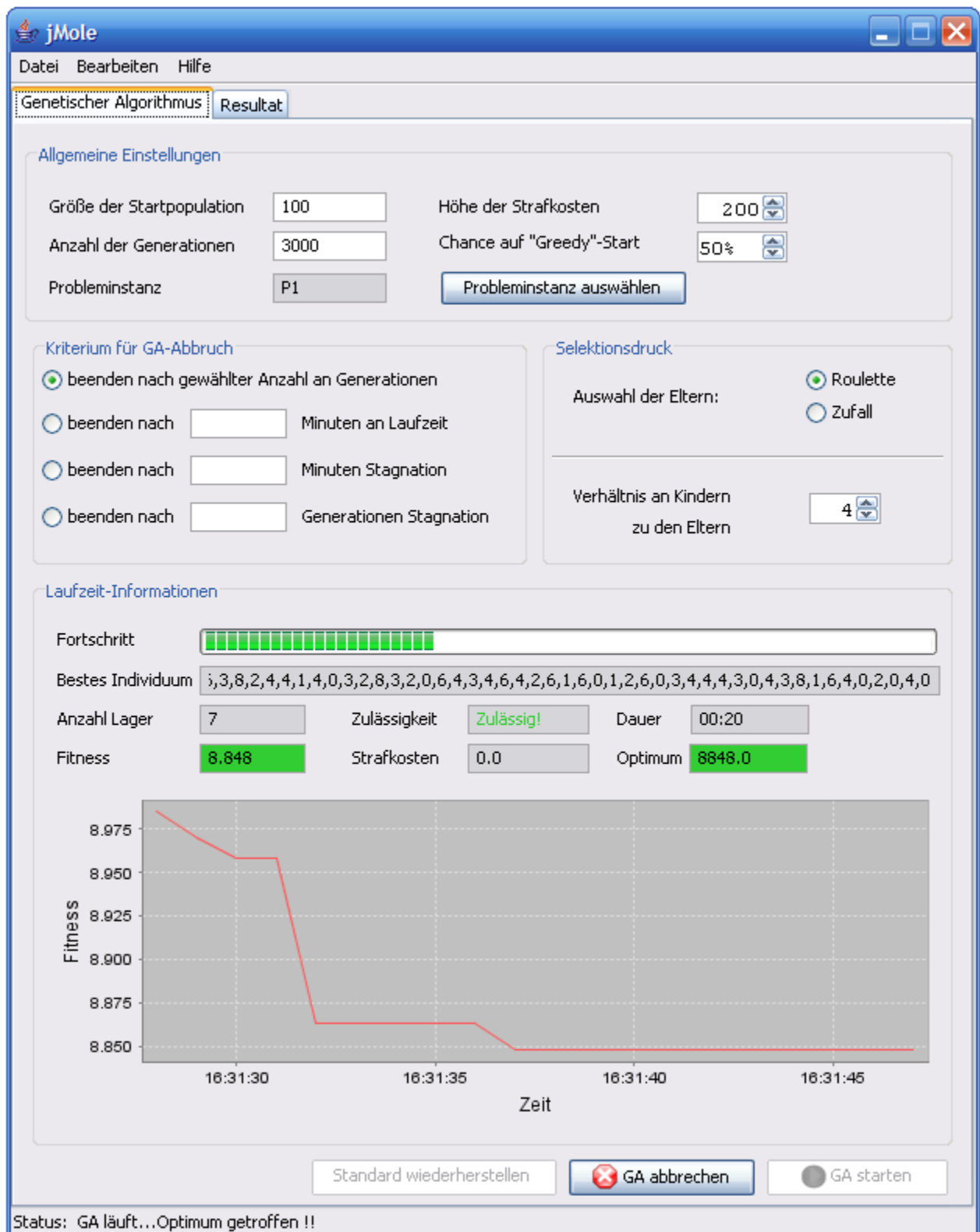
Die GUI



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 detaillierte 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.

```
Output
AustraliaGA (run-single) x AustraliaGA (run-single) #2 x
init:
deps-jar:
compile:
run-single:
Fitness: 35891.0 Fees: 0.0 Warehouses: 25 Gene: 18,19,24,27,2,12,15,27
Fitness: 35697.0 Fees: 0.0 Warehouses: 25 Gene: 18,19,24,27,2,12,15,27
Fitness: 35524.0 Fees: 0.0 Warehouses: 25 Gene: 18,19,24,27,2,12,15,27
Bestes Individuum:
Fitness: 35524.0 Fees: 0.0 Warehouses: 25 Gene: 18,19,24,27,2,12,15,27
Individual successfully added to database
Ende
Dauer: 169058|
BUILD SUCCESSFUL (total time: 2 minutes 54 seconds)
```

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.

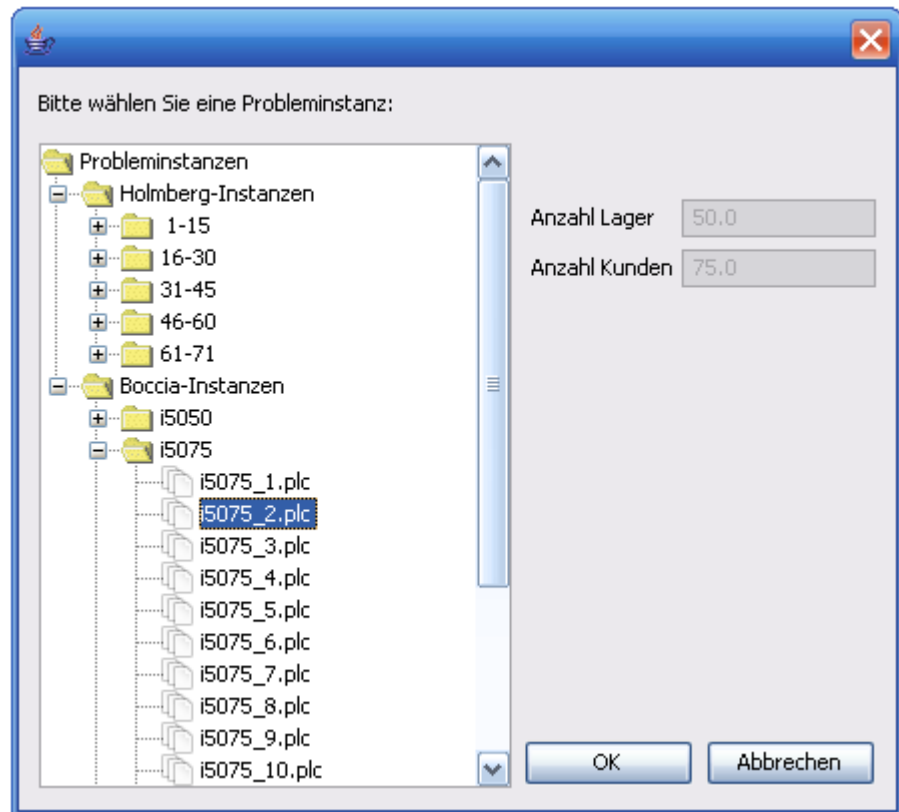
Konfiguration der allgemeinen Einstellungen

Für die allgemeinen Einstellungen sind die grundsätzlichen 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.

Allgemeine Einstellungen

Größe der Startpopulation	<input type="text" value="100"/>	Höhe der Strafkosten	<input type="text" value="200"/>
Anzahl der Generationen	<input type="text" value="3000"/>	Chance auf "Greedy"-Start	<input type="text" value="50%"/>
Probleminstanz	<input type="text" value="P1"/>	<input type="button" value="Probleminstanz auswählen"/>	

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 Problemistanzen nicht den Überblick zu verlieren, wurden sowohl die Holmberg- als auch die Boccia-Problemistanzen 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.

Kriterium für GA-Abbruch

- ☒ beenden nach gewählter Anzahl an Generationen
- ☐ beenden nach Minuten an Laufzeit
- ☐ beenden nach Minuten Stagnation
- ☐ beenden nach Generationen Stagnation

Selektionsdruck

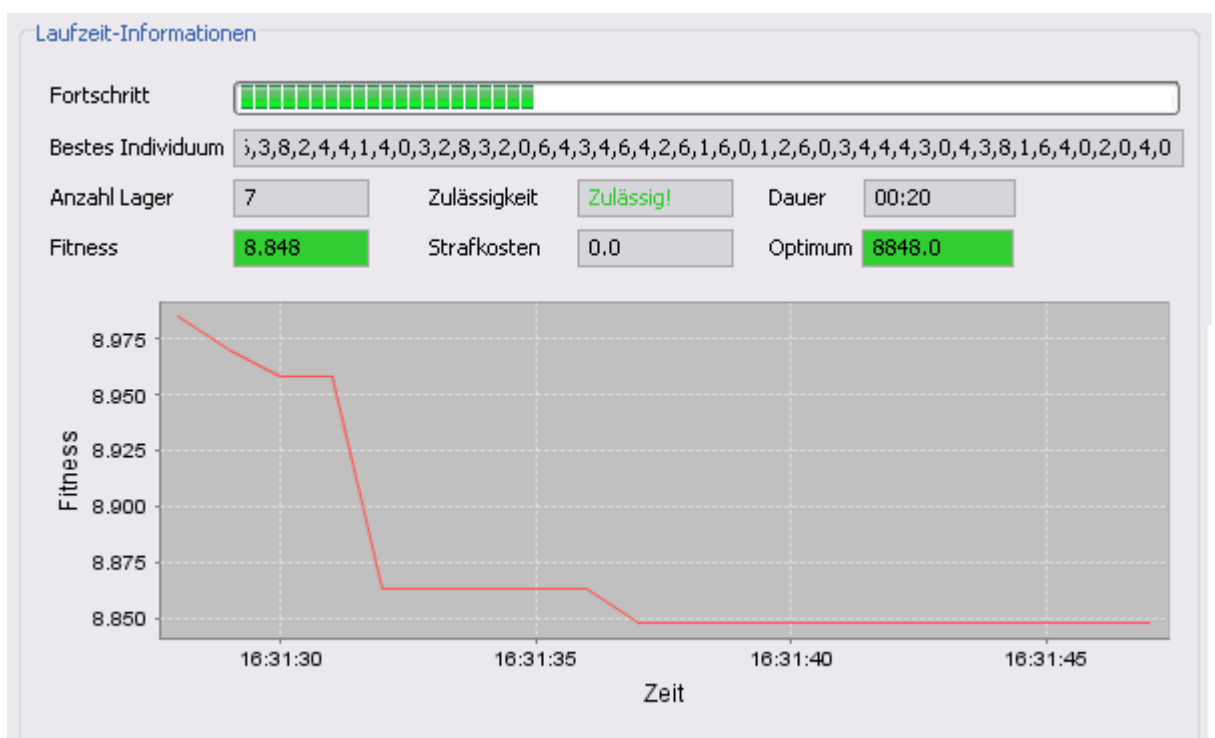
Auswahl der Eltern: ☒ Roulette ☐ Zufall

Verhältnis an Kindern zu den Eltern:

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

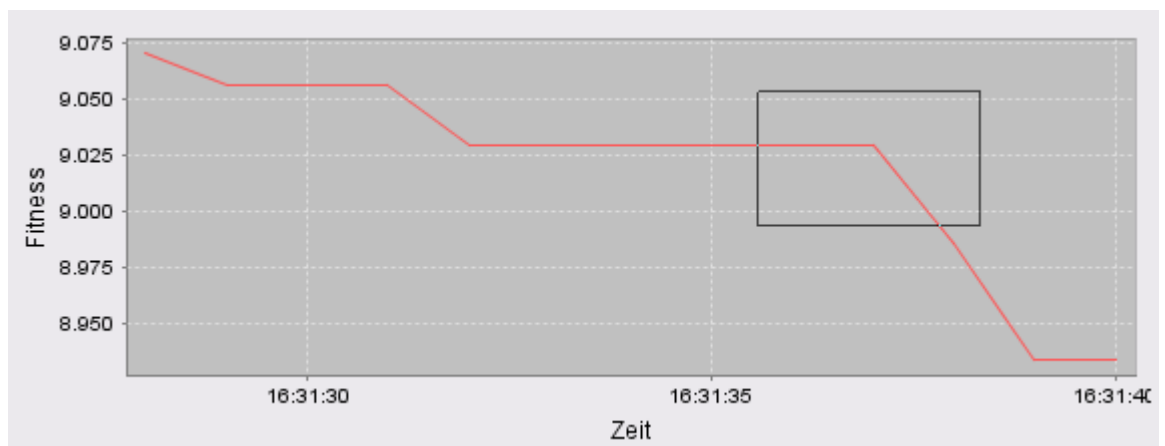
Nach erfolgreicher Konfiguration kann der Algorithmus über den Button „GA starten“ gestartet werden, sodass sich dem Benutzer folgende Informationen offenbaren.



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 Kontextmenü, 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 Ausgangsstellung.

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.

Genetischer Algorithmus		Resultat					
ID	Datum	Problem Instanz	Fitness	Lager	Zulässig	Strafkosten	Dauer
1	28:08:07 16...	P1	8.916	8	<input checked="" type="checkbox"/>		0 00:20
2	28:08:07 16...	P1	8.848	7	<input checked="" type="checkbox"/>		0 00:23
3	28:08:07 17...	P1	8.891	8	<input checked="" type="checkbox"/>		0 00:01
4	28:08:07 17...	P1	8.934	8	<input checked="" type="checkbox"/>		0 00:13

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

Instanz	Ergebnis Berechnung	Ergebnis Holmberg	Absolute Abweichung	Prozentuale Abweichung
P1	8848	8848	0	0,00%
P2	7913	7913	0	0,00%
P3	9314	9314	0	0,00%
P4	10714	10714	0	0,00%
P5	8838	8838	0	0,00%
P6	7777	7777	0	0,00%
P7	9488	9488	0	0,00%
P8	11088	11088	0	0,00%
P9	8462	8462	0	0,00%
P10	7617	7617	0	0,00%
P11	8932	8932	0	0,00%
P12	10132	10132	0	0,00%
P13	8252	8252	0	0,00%
P14	7137	7137	0	0,00%
P15	8808	8808	0	0,00%
P16	10408	10408	0	0,00%
P17	8227	8227	0	0,00%
P18	7125	7125	0	0,00%
P19	8886	8886	0	0,00%
P20	10486	10486	0	0,00%
P21	8068	8068	0	0,00%
P22	7092	7092	0	0,00%
P23	8746	8746	0	0,00%
P24	10273	10273	0	0,00%
P25	11663	11630	33	0,28%
P26	10786	10771	15	0,14%
P27	12371	12322	49	0,40%
P28	13972	13722	250	1,82%
P29	12651	12371	280	2,26%
P30	11408	11331	77	0,68%
P31	13600	13331	269	2,02%

P32	15808	15331	477	3,11%
P33	11629	11629	0	0,00%
P34	10632	10632	0	0,00%
P35	12232	12232	0	0,00%
P36	13832	13832	0	0,00%
P37	11326	11258	68	0,60%
P38	10594	10551	43	0,41%
P39	11951	11824	127	1,07%
P40	13059	13024	35	0,27%
P41	6681	6589	92	1,40%
P42	5680	5663	17	0,30%
P43	5214	5214	0	0,00%
P44	7042	7028	14	0,20%
P45	6297	6251	46	0,74%
P46	5715	5651	64	1,13%
P47	Datei Defekt	6228		
P48	5596	5596	0	0,00%
P49	5386	5302	84	1,58%
P50	8766	8741	25	0,29%
P51	7457	7414	43	0,58%
P52	9178	9178	0	0,00%
P53	8531	8531	0	0,00%
P54	8777	8777	0	0,00%
P55	7685	7654	31	0,41%
P56	21867	21103	764	0,36%
P57	26538	26039	499	1,92%
P58	37486	37239	247	0,66%
P59	28194	27282	912	3,34%
P60	20877	20534	343	1,67%
P61	24784	24454	330	1,35%
P62	33292	32643	649	1,99%
P63	26530	25105	1425	5,68%
P64	20877	20530	347	1,70%
P65	24977	24445	532	2,17%
P66	32423	31415	1008	3,21%
P67	25564	24848	716	2,88%
P68	20872	20538	334	1,63%
P69	25297	24532	765	3,12%
P70	33467	32321	1146	3,55%
P71	26813	25540	1273	4,98%

Ergebnisse Boccia

Instanz	Ergebnis
i5050_1.plc	18291,71
i5050_2.plc	19622,01
i5050_3.plc	20671,75
i5050_4.plc	21749,93
i5050_5.plc	15036,96
i5050_6.plc	6558,53
i5050_7.plc	8303,33
i5050_8.plc	6314,32
i5050_9.plc	7321,2
i5050_10.plc	6385,44
i5050_11.plc	3542,7
i5050_12.plc	4071,47
i5050_13.plc	4193,43
i5050_14.plc	4061,28
i5050_15.plc	6285,91
i5075_1.plc	15246,56
i5075_2.plc	20899,69
i5075_3.plc	19955,14
i5075_4.plc	23476,81
i5075_5.plc	22351,8
i5075_6.plc	8151,54
i5075_7.plc	9289,81
i5075_8.plc	7924,34
i5075_9.plc	8495,9
i5075_10.plc	7775,99
i5075_11.plc	5467,07
i5075_12.plc	5681,08
i5075_13.plc	5819,47
i5075_14.plc	4895,51
i5075_15.plc	6758,68
i50100_1.plc	18302,6
i50100_2.plc	19773,44

i50100_3.plc	18734,84
i50100_4.plc	21651,4
i50100_5.plc	20501,88
i50100_6.plc	9298,86
i50100_7.plc	9335,36
i50100_8.plc	8253,23
i50100_9.plc	9755,42
i50100_10.plc	9270,67
i50100_11.plc	7203,63
i50100_12.plc	6537,19
i50100_13.plc	6376,83
i50100_14.plc	8235,43
i50100_15.plc	7928,77
i75100_1.plc	35912,24
i75100_2.plc	37880,48
i75100_3.plc	27476,88
i75100_4.plc	32200,22
i75100_5.plc	28771,67
i75100_6.plc	10997,48
i75100_7.plc	12133,46
i75100_8.plc	12382,68
i75100_9.plc	10614,38
i75100_10.plc	13197,15
i75100_11.plc	8929,25
i75100_12.plc	6980,19
i75100_13.plc	6744,26
i75100_14.plc	6766,82
i75100_15.plc	6938,49
i7575_1.plc	25492,5
i7575_2.plc	28722,16
i7575_3.plc	28840,61
i7575_4.plc	32901,12
i7575_5.plc	27875,81
i7575_6.plc	9930,19
i7575_7.plc	9286,3
i7575_8.plc	9687,13
i7575_9.plc	11343,08
i7575_10.plc	10502,86
i7575_11.plc	6688,25
i7575_12.plc	5648,63
i7575_13.plc	6432,02
i7575_14.plc	7324,76
i7575_15.plc	5821,02

Verhalten bei Änderungen

Im Folgenden sollen Änderungen der Parameter untersucht werden um das Verhalten des Algorithmus zu evaluieren und um eine optimale Parametrisierung des Algorithmus zu erhalten.

Alle Untersuchungen belaufen sich auf die Boccia-Instanz i75100_5.

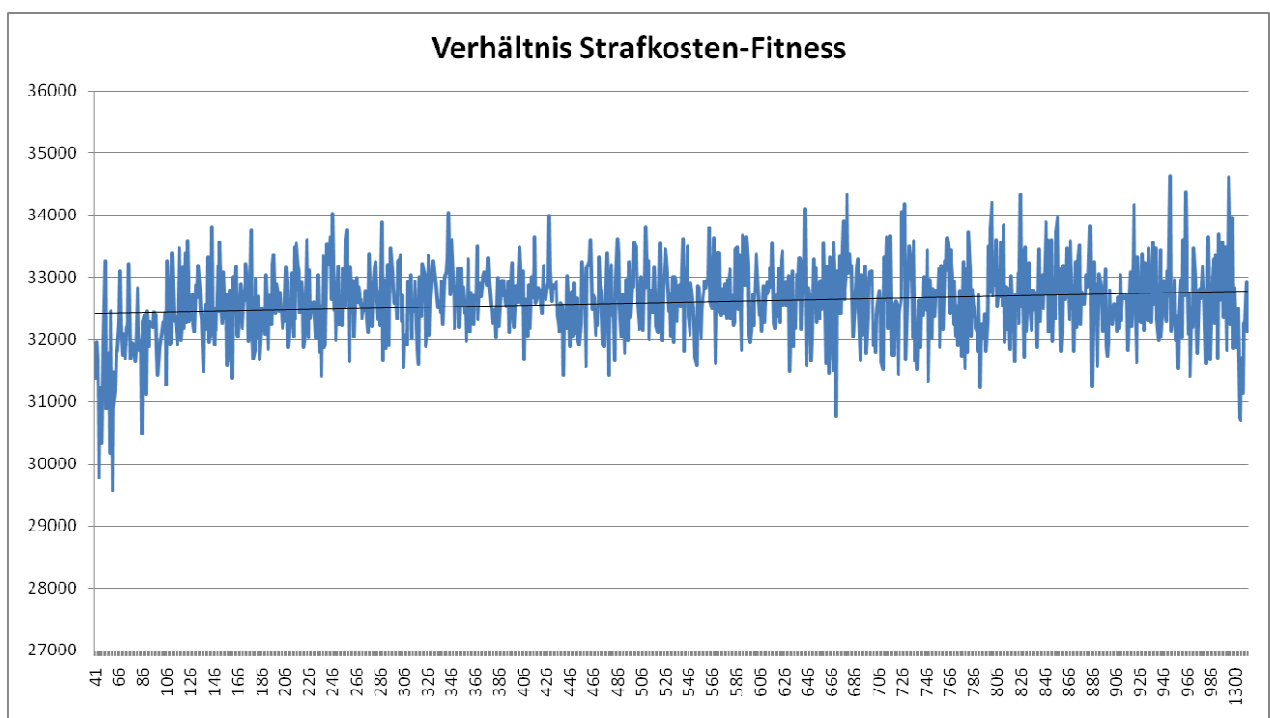
Die Start-Parameter sind folgende, die auch zur Berechnung der Optima eingesetzt wurden:

Strafkosten:	200
Generationenanzahl:	3000
Selektionsmethode:	Roulette Wheel
Verhältnis Kinder/Eltern	2/1
Chance auf Greedy:	50%

Bei den folgenden Untersuchungen wurde jeweils ein Wert verändert.

Änderung der Strafkosten

jMole verwendet den Ansatz der Strafkosten. Es soll daher untersucht werden, wie sich eine Änderung der Strafkosten auf die Qualität der berechneten Optima auswirkt. Im einen Testdurchlauf wurde eine Vielzahl von Strafkosten untersucht. Bis zu einem Strafkosten-Wert von 40 werden nur ungültige Lösungen erzeugt (Strafe zu günstig!)

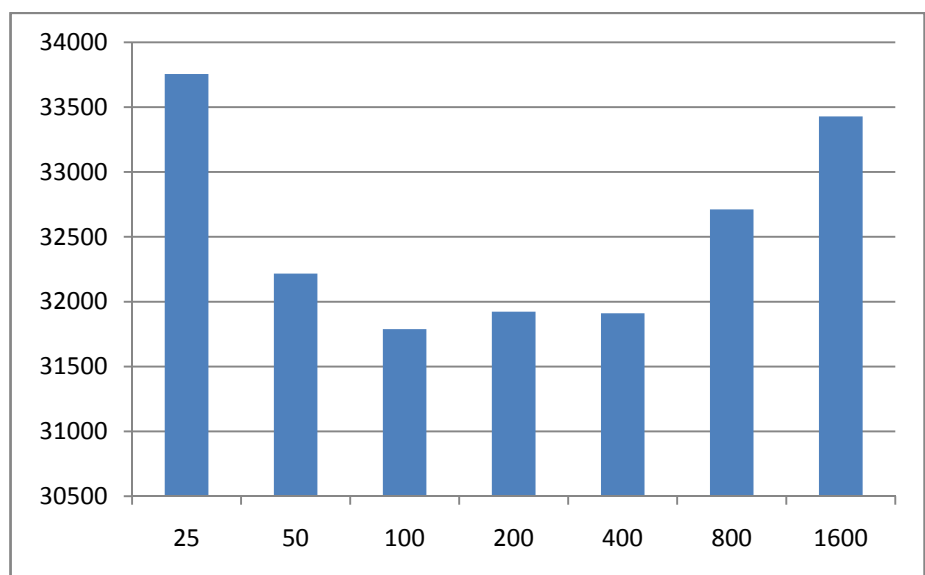


Interessant ist die Tatsache, dass bei einer Erhöhung der Strafkosten die Ergebnisqualität nur sehr leicht abnimmt.

Änderung der Populationsgröße

Es wurden jeweils 5 Läufe mit jeder Einstellung vorgenommen, wobei das Ergebnis der Mittelwert dieser 5 Durchläufe ist.

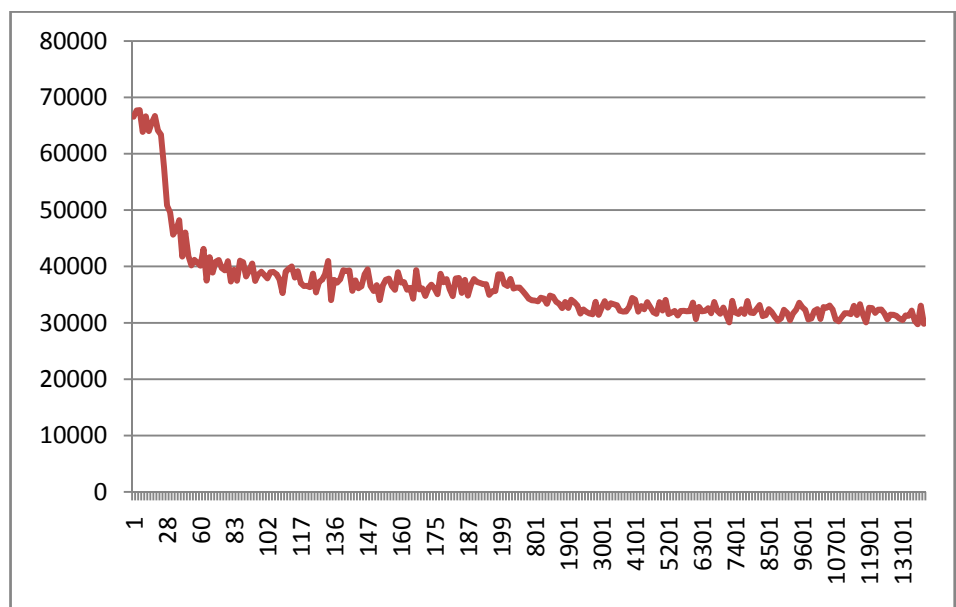
Folgende Populationsgrößen wurden verwendet: 25, 50, 100, 200, 400, 800, 1600



Anhand der Grafik ist zu erkennen, dass bei dieser Instanz und den Standardparametereinstellungen eine Populationsgröße von 100 die besten Ergebnisse erzielt.

Änderung der Generationenanzahl

Verändert sich die Anzahl der durchzuführenden Generationen, so kann die Ergebnisqualität verbessert werden. Eine doppelte Generationenanzahl verdoppelt aber auch die Rechenzeit bis zur Lösung.



Es wird deutlich, dass bis 100 Generationen sich die Qualität sehr schnell verbessert. Bis 2000 Generationen ist noch eine Verbesserung erkennbar. Ab 10000 Generationen verbessert sich die Lösung nur sehr langsam.

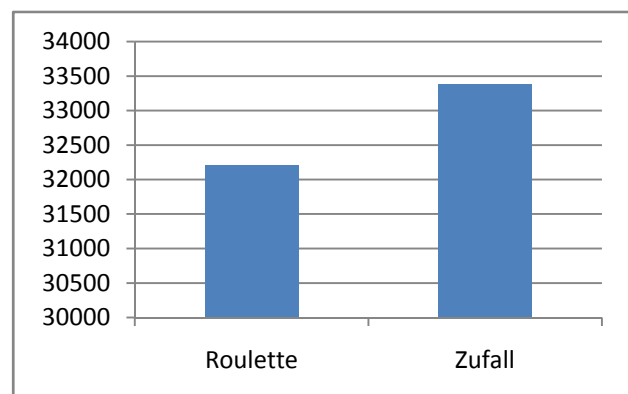
Änderung der Selektionsmethode

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.

Selektionsmethode: Eltern

Bei der Selektion der Eltern für die Rekombination kann entweder zufällig oder nach dem rangbasierten Roulette-Wheel-Verfahren vorgegangen werden.

Selektionsmethode:	Eltern	
	Roulette	Random
Fitness Stichprobe 1	32.191,58	32.879,78
Fitness Stichprobe 2	31.606,86	32.385,83
Fitness Stichprobe 3	31.308,46	34.064,11
Fitness Stichprobe 4	32.768,11	35.103,00
Fitness Stichprobe 5	33.161,32	32.515,36
Durchschnitt	32.207,27	33.389,62
Median	32.191,58	32.879,78
Δ Durchschnitt	-1182,35	0,00
% Durchschnitt	-3,54%	0,00%
Δ Median	-688,20	0,00
% Median	-2,09%	0,00%

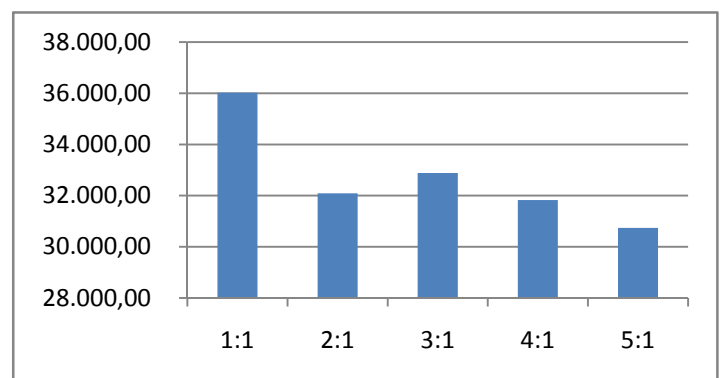


Bei vier der Stichproben war die Selektionsmethode „Roulette“ eindeutig besser, da jeweils niedrigere Fitness-Werte und damit niedrigere Kosten erzielt wurden. Um die Werte zu glätten wurden bei der weiteren Berechnung für die fünf Stichproben der beiden Selektionsmethoden jeweils der durchschnittliche Mittelwert und der Median berechnet.

Selektionsmethode: Eltern

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.

Selektionsmethode:	Kinder: Verhältnis an Kindern zu Eltern				
	1	2	3	4	5
Fitness Stichprobe 1	35.138,51	31.764,93	31.995,08	30.904,41	30.256,75
Fitness Stichprobe 2	34.278,27	31.616,46	32.651,34	30.771,45	30.561,50
Fitness Stichprobe 3	36.506,20	32.686,41	34.566,36	32.679,06	31.613,88
Fitness Stichprobe 4	38.059,65	32.290,42	33.834,83	31.609,92	30.131,11
Fitness Stichprobe 5	36.136,12	32.094,93	31.362,12	33.161,32	31.133,04
Durchschnitt	36.023,75	32.090,63	32.881,95	31.825,23	30.739,26
Median	36.136,12	32.094,93	32.651,34	31.609,92	30.561,50
Δ Durchschnitt	0,00	-3.933,12	-3.141,80	-4.198,52	-5.284,49
% Durchschnitt	0,00%	-10,92%	-8,72%	-11,65%	-14,67%
Δ Median	0,00	-4.041,19	-3.484,78	-4.526,20	-5.574,62
% Median	0,00%	-11,18%	-9,64%	-12,53%	-15,43%



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

GNU LESSER GENERAL PUBLIC LICENSE

Die Ergebnisse dieser Arbeit, der Software und der Algorithmus
werden unter der GNU Lesser General Public License veröffentlicht.

<http://www.gnu.org/licenses/lgpl.html>