jMole

A Genetic Approach to the Australia-Problem

Jochen Christ  
Daniel Kimmig  
Benjamin Kühnlenz

Inhalt

[Einleitung 4](#_Toc174676405)

[Das Australia-Problem 5](#_Toc174676406)

[Der Algorithmus 6](#_Toc174676407)

[Der Algorithmus im Überblick 7](#_Toc174676408)

[Die Codierung 8](#_Toc174676409)

[Der Strafkosten-Ansatz 9](#_Toc174676410)

[Das Erzeugen der Startpopulation 10](#_Toc174676411)

[Die Selektion 11](#_Toc174676412)

[Die Rekombination 12](#_Toc174676413)

[Die Mutation 13](#_Toc174676414)

[Optimierung der Ergebnisqualität 14](#_Toc174676415)

[Performance 15](#_Toc174676416)

[Die GUI 16](#_Toc174676417)

[Die GUI 17](#_Toc174676418)

[Ergebniswerte 18](#_Toc174676419)

[Ergebnisse Holmberg 19](#_Toc174676420)

[Ergebnisse Boccia 20](#_Toc174676421)

[Verhalten bei Änderungen 21](#_Toc174676422)

[Änderung der Strafkosten 22](#_Toc174676423)

[Änderung der Populationsgröße 23](#_Toc174676424)

[Änderung der Generationenanzahl 24](#_Toc174676425)

[Änderung der Selektionsmethode 25](#_Toc174676426)

[Lizenz 26](#_Toc174676427)

[LGPL 27](#_Toc174676428)

# 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 Algorithmus[[1]](#footnote-2) verwendet. Da besondere 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 19 dargestellt.

# Das Australia-Problem

Beschreibung des Problems.

Der Algorithmus

Im Folgenden wird der Algorithmus und die Details vorgestellt.

# Der Algorithmus im Überblick



# Die Codierung

# 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

benni

Greedy, A bit Greety and Random

# 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.

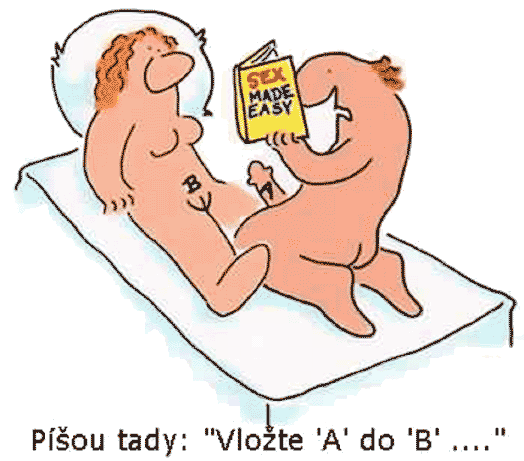
Eine Möglichkeit ist das rangbasierte Roulette-Wheel-Verfahren. Hier werden den Individuen mit einer besseren Fitness eine höhere Wahrscheinlichkeit zugeschrieben, als schlechteren.

2. Größe der Kinderpopulationen

Es können in der neuen Generation beliebig viele Kinder erzeugt werden. Da die Größer 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 eine Facility 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.

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

Mutation

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

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 warden 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.

mutateBanFacilityAndFindNewFromCurretUsed();

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

# Optimierung der Ergebnisqualität

jc

Verwendung der Datenbank und Einspielen bekannter Optima.

# Performance

JProfiler willkommen.

Die GUI

# Die GUI

Console

und unsere Supaduba-Swing-GUI -> Hier kann Daniel ganz viel reinschreiben.

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

a

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

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. [↑](#footnote-ref-2)