Technology
Arts Sciences
TH Köln

Parallelisierter Genetischer Algorithmus für das TSP

Timo Bertsch
Joram Markert
Fabian Meyer

Master Computer Science

Gummersbach, 30.08.2016

Abstract

Thema:	Parallelisierter Genetischer Algorithmus für das TSP
Author:	Timo Bertsch Joram Markert Fabian Meyer
Prüfer:	Prof. Dr. Lutz Köhler Pascal Schönthier
Datum:	30.08.2016

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Inhaltsverzeichnis

1 Einleitung

einleitungs blabla

2 Das Travelling Salesman Problem

Was ist das problem / Ziel? Auf kombinatorische explosion eingehen; Quellen suchen und erklörung darauf stützen

3 Der Genetische Algorithmus

Im Rahmen dieses Projektes wurde ein genetischer Algorithmus entwickelt, mit dem das TSP gelöst wird. Genetische Algorithmen werden im Allgemeinen zur Lösungsfindung bei komplexen Suchproblemen verwendet, bei denen eine kombinatorische Explosion vorliegt und der Lösungsraum so groß ist, dass eine Lösung mit einer optimalen Brute Force Methode nicht möglich ist.

Die allgemeine Funktionsweise dieses Algorithmus und die Varianten, die in diesem Projekt eingesetzt wurden, werden in Abschnitt 3.1 erläutert. Der Basisalgorithmus ist jedoch nicht nebenläufig konzipiert, daher mussten einige Modifikationen vorgenommen werden, um diesen in einem verteilten System einsetzen zu können. Diese Änderungen werden in **??** näher beschrieben.

3.1 Funktionsweise

Genetische Algorithmen basieren auf der Idee der natürlichen Auslese und ermitteln durch Selektion, Kombination und Evolution von möglichen Lösungen eine sehr gute bis perfekte Lösung für das gesuchte Problem.
Eine Lösung des gegebenen Problems wird als *Individuum* bezeichnet. Im Rahmen des TSP sind Individuen Wege, die die Lösungskriterien des TSP erfüllen (siehe Kapitel 2). Die Reihenfolge der Knoten, die bei einem Individuum besucht werden, werden als dessen Eigenschaften oder Gene bezeichnet. Eine Menge von Individuen, auf der der Algorithmus operiert, wird *Population* genannt. Der genetische Algorithmus arbeitet iterativ in sogenannten *Generationen*. Die Individuen einer Population werden schrittweise in sogenannten *Generationen* mit einer *Fitness* bewertet, aussortiert und rekombiniert, um die Qualität der Lösungen zu verbessern. Die Phasen einer Generation werden in Tabelle 3.1 zusammenfassend dargestellt (vgl. [2]).

Phase	Beschreibung
Initialisation	Generiert die initiale Population und wird nur ein einziges Mal zu Beginn des Algorithmus ausgeführt. Die Population wird mit zufällig generierten Individuen erstellt.
Evaluation	Die Individuen der aktuellen Generation werden mit einem Fitnesswert bewertet. Dieser Zahlenwert drückt aus wie gut das Individuum bzw. die Lösung des des Problems ist. In Bezug auf das TSP bedeutet eine kürzere Strecke der Lösung eine höhere Fitness des entsprechenden Individuums.
Selection	Aus den Individuen der aktuellen Generation werden diejenigen ausgewählt, die für die Crossover Phase verwendet und die in die nächste Generation übernommen werden. Hierbei ist es von zentraler Bedeutung, dass Individuen mit hoher Fitness mit einer höheren Wahrscheinlichkeit ausgewählt werden als die Individuen mit niedriger Fitness.
Crossover	Aus den ausgewählten Individuen (sog. Eltern) werden duch die Kombination von deren Eigenschaften neue Individuen (sog. Kinder) erzeugt, die in die nächste Generation übernommen werden. In Bezug auf das TSP werden hier Teilwege der Eltern so miteinander, dass die Kinder wieder eine valide Lösung des TSP darstellen.
Mutation	Die Gene der erzeugten Kinder werden per Zufall verändert, um eine Stagnation des Genpools zu verhindern und neue Lösungsmöglichkeiten zu erzeugen. Diese Stagnation kann auftreten, wenn über mehrere Generationen hinweg besonders fitte Individuen die Population dominieren.

Tabelle 3.1: Phasen des genetischen Algorithmus

Die Umsetzung dieser Phasen im Rahmen dieses Projekts wird in den folgenden Unterabschnitten anhand des Beispielgraphen in Abbildung 3.1 detailliert beschrieben.

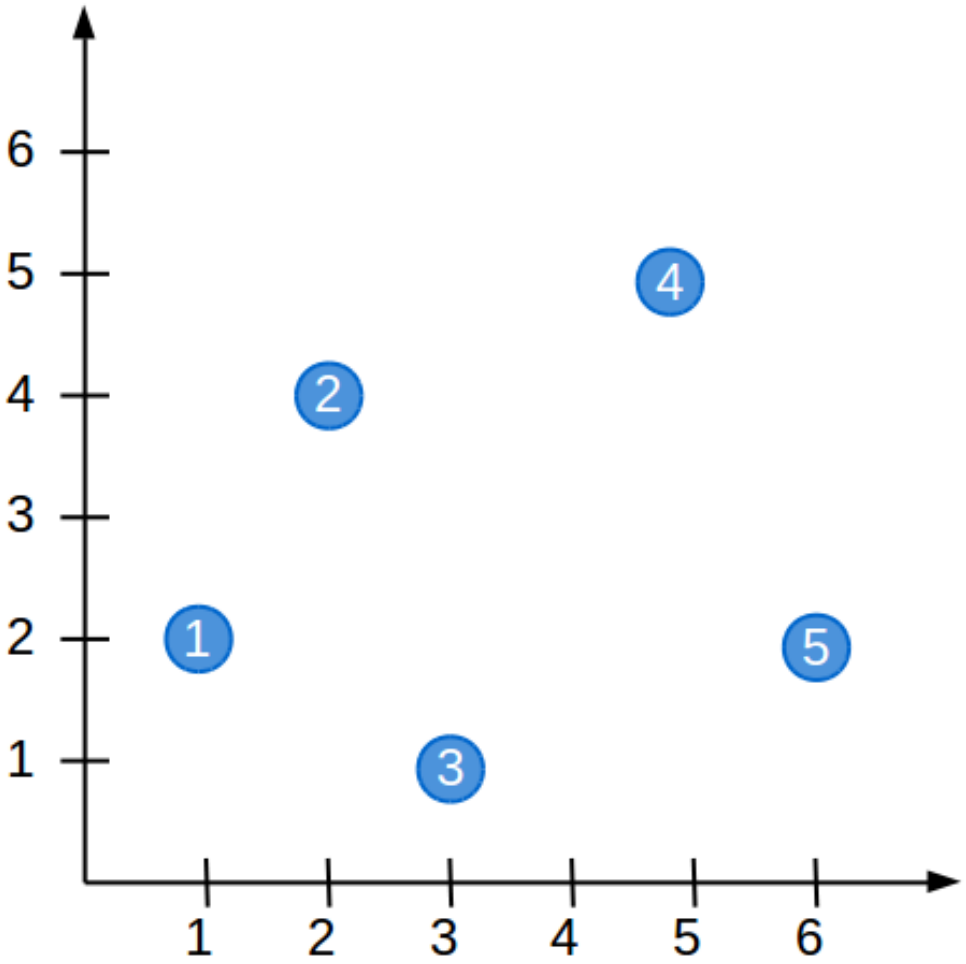


Abbildung 3.1: Beispielgraph

Dieses Beispiel zeigt einen Graphen mit fünf Knoten, die jeweils durch die Nummer in ihrer Mitte identifiziert werden können.

3.1.1 Initialisation

Die *Initialisation* Phase wird nur ein Mal zu Beginn des Algorithmus ausgeführt. In dieser Phase wird die initiale Population generiert, mit der der Algorithmus dann arbeitet. Die Individuen werden hierbei vollständig zufallsgeneriert, wobei jedes Individuum eine valide Lösung des TSP darstellt. Dieser Vorgang wird in Listing 3.1 dargestellt.
Um ein Individuum zu generieren, wird eine Liste (*nodes*) aller möglichen Knoten für das Individuum, also Knoten aus dem Graphen, erstellt (l. 1). Aus dieser Liste wird dann ein Knoten zufällig ausgewählt (l. 5) und dem Individuum hinzugefügt (l. 6). Danach wird der Knoten aus der Liste der übrigen Knoten entfernt

```
1 List nodes = //...
2 List indivduum
3
4 while(not nodes.empty())
5     int index = randomInt()
6     individuum.add(nodes[index])
7     nodes.remove(index)
```

Listing 3.1: Generierung eines Individuums

(l. 7) und dieser Prozess wird solange wiederholt bis keine Knoten mehr in *nodes* vorhanden sind (l. 4).

Mit diesem Verfahren können die Individuen in Abbildung 3.2 aus dem Beispielgraphen in Abbildung 3.1 generiert werden.

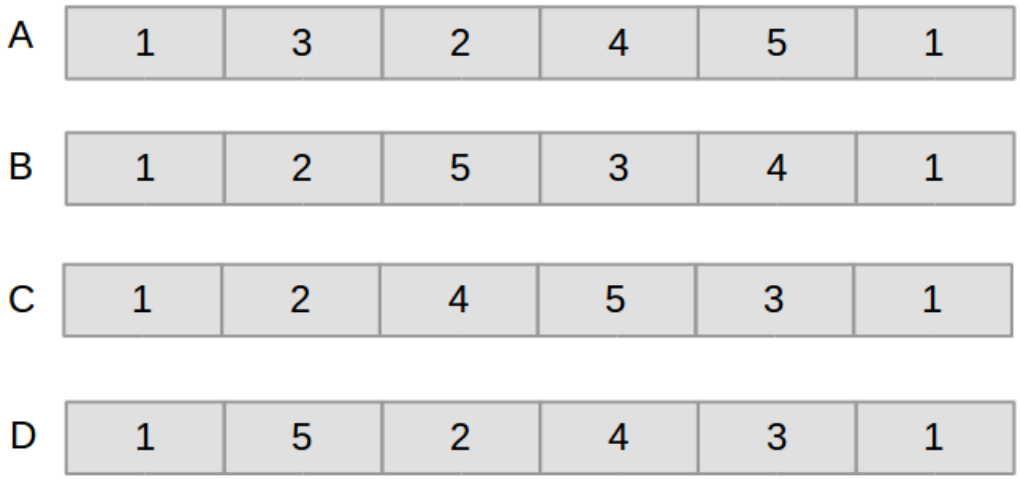


Abbildung 3.2: Beispiel: Zufällig generierte Individuen

3.1.2 Evaluation

In der *Evaluation* Phase werden die Individuen der Population der aktuellen Generation bewertet und damit ihr Fitness Wert errechnet. Jedes Individuum stellt einen Weg durch den Graphen, also eine Liste aufeinander folgender Knoten, dar. Die Fitness errechnet sich aus der Länge dieses Weges. Diese Länge wird durch die Summe des euklidischen Abstands zwischen in der Liste aufeinanderfolgende Knoten beschrieben, wie in der folgenden Gleichung dargestellt wird.

$$l_{individuum} = \sum_{i=1}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

Dies wird für alle Individuen wiederholt. Hierbei gilt, dass umso länger der Weg ist umso niedriger ist der Fitnesswert eines Individuums. Damit verhält sich die Fitness also *antiproportional* zur Länge des Weges.
Aufgrund von Rundungsfehlern bei Gleitkommazahlen reicht ein einfacher Kehrwert der Länge des Weges nicht aus um die Fitness eines Individuums zu berechnen, da der Weg je nach Größe des Graphen entsprechend lang werden kann. Daher wird aus allen Individuen der aktuellen Population dasjenige gesucht, das den längsten Weg l_{max} besitzt. Mit diesem Wert wird nun die Fitness $f_{individuum}$ jedes Individuums auf folgende Weise berechnet.

$$f_{individuum} = \frac{l_{max}}{l_{individuum}}$$

Tabelle 3.2 zeigt die Weglänge und die Fitness der Individuen aus Abbildung 3.2.

Individuum	$l_{individuum}$	$f_{individuum}$
A	$1 + \sqrt{10} + \sqrt{10} + \sqrt{10} + 5 = 15.49$	$\frac{19.34}{15.49} = 1.25$
B	$\sqrt{5} + \sqrt{20} + \sqrt{10} + \sqrt{20} + 5 = 19.34$	$\frac{19.34}{19.34} = 1$
C	$\sqrt{5} + \sqrt{10} + \sqrt{10} + \sqrt{10} + 1 = 12.72$	$\frac{19.34}{12.72} = 1.52$
D	$5 + \sqrt{20} + \sqrt{10} + \sqrt{20} + 1 = 18.11$	$\frac{19.34}{18.11} = 1.07$

Tabelle 3.2: Beispiel Fitness und Distanz

3.1.3 Selection

Während der *Selection* Phase werden die Individuen (Eltern) ausgewählt, aus denen die Individuen für die nächste Generation (Kinder) generiert werden. Die Auswahl geschieht anhand der Fitness der Individuen.Wichtig ist hierbei auch, dass ein einzelnes Individuum mehrfach ausgewählt werden kann.
In diesem Projekt wurde zur Auswahl der Eltern das *Roulette-Wheel-Verfahren* verwendet. Hierbei werden die Individuen durch eine Zufallszahl ausgewählt, doch haben besonders fitte Individuen eine besonders hohe Chance ausgewählt zu werden. Hierbei wird die Fitness der Individuen auf das Intervall [0;1] normiert. Daraufhin wird eine Auswahlintervall für jedes Individuum berechnet, indem die Liste der Individuen angefangen beim fittesten Individuum durchlaufen wird und die Fitness aller vorangegangenen Individuen aufsummiert wird. Der sich daraus ergebende Wert ist die untere Grenze g_{xunten} des Auswahlintervalls des Individu-

ums x. Die obere Grenze g_{xoben} wird durch die Summe aus unterer Grenze und der Fitness des aktuellen Individuums gebildet (vgl. [3]). Die folgenden Gleichungen beschreiben den Rechenweg nochmals.

$$g_{xunten} = \sum_{i=1}^{i<x} f_i$$

$$g_{xoben} = g_{xunten} + f_x$$

Abbildung 3.3 zeigt die Auswahlintervalle der Individuen aus Abbildung 3.2.

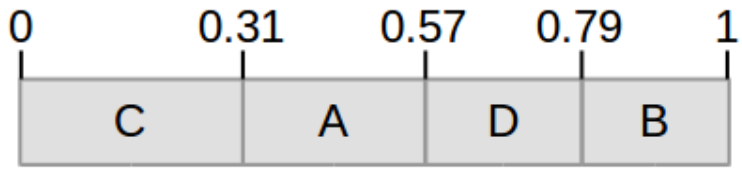


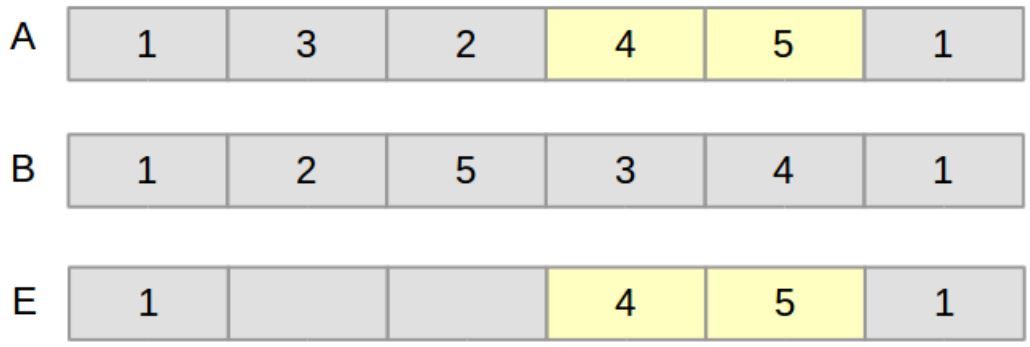
Abbildung 3.3: Beispiel: Auswahlintervalle der Individuen

Nun wird eine Zufallszahl im Intervall [0;1] gewürfelt und es wird dasjenige Individuum zum Elternteil erwählt in dessen Intervall die Zufallszahl fällt. Dabei werden immer paare von zwei Eltern gebildet, aus denen ein neues Individuum generiert wird. Dies wird solange wiederholt bis genügend Individuen ausgewählt wurden um die aktuelle Population vollständig zu ersetzen.

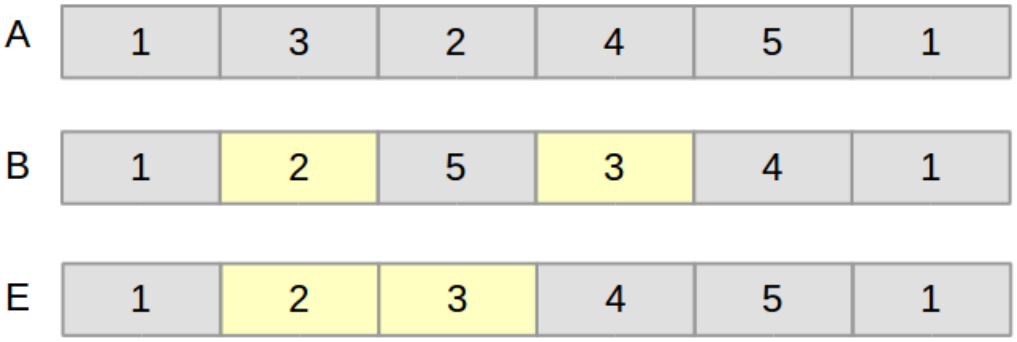
Zusätzlich zur Auswahl der Individuen verwendet der Algorithmus außerdem eine *Elitism* Mechanik, durch die die besten Individuen der aktuellen Generation in die nächste Generation übernommen werden. So wird verhindert, dass bereits gefundene sehr gute Lösungen wieder verworfen werden. Alle anderen Individuen der aktuellen Generation werden jedoch durch neue Individuen, die Kinder, ersetzt.

3.1.4 Crossover

In der *Crossover* Phase werden die aus der *Selection* Phase ausgewählten Eltern kombiniert, um neue Individuen für die nächste Generation zu generieren.
Hierbei wird ein zusammenhängender Teilweg des Elternindividuums A aus zufällig ausgewählt und genau so in das Kind E übernommen. Um die restlichen Knoten des Kindes zu füllen werden der Reihe nach Knoten aus Elternindividuum B Knoten entnommen, falls diese in dem Kind noch nicht vorhanden sind, und in das Kind eingefügt (vgl. [1]). In Abbildung 3.4 wird dieser Prozess nochmal verdeutlicht.



(a)



(b)

Abbildung 3.4: Beispiel: Crossover von zwei Individuen

in Abbildung 3.4a wurde der Teilweg [4,5] aus Individuum A ausgewählt und an der gleichen Stelle in das neue Kind E eingefügt. Im nächsten Schritt werden in Abbildung 3.4b die Knoten in Individuum B der Reihe nach durchgegangen und ,falls sie in Individuum E noch nicht vorhanden sind, entsprechend eingefügt.
Das neu entstandene Individuum E ist nun eine Kombination aus den Lösungen von A und B und umfasst ohne weitere Korrekturen immer direkt eine valide Lösung des TSP.
Dieser Vorgang wird nun für alle im *Selection* Schritt ausgewählten Elternpaare wiederholt.

3.1.5 Mutation

Die *Mutationsphase* dient dazu, um die Variation im Genpool der Population aufrecht zu erhalten. So kann es passieren, dass ein besonders fittes Individuum die Selection Phase dominiert und der Genpool gegen diese Lösung konvergiert. Durch die genetische Verarmung können potentiell bessere Lösungen nicht mehr

gefunden werden (vgl. [3]). Um diesem Effekt entgegenzuwirken wird ein sehr kleiner Anteil aller Kinder, die im *Crossover* Schritt generiert wurden, zufällig mutiert.
Im Rahmen dieses Projekts wurde die *Swap Mutation* verwendet, um Individuen zu mutieren. Hierbei werden zwei zufällig ausgewählte Knoten aus einem Individuum miteinander vertauscht. Die hieraus entstandene Lösung ist ebenfalls wieder eine valide Lösung des TSP (vgl. [1]). Abbildung 3.5 zeigt diesen Vorgang nochmals anhand des in Unterabschnitt 3.1.4 erzeugten Individuums E.

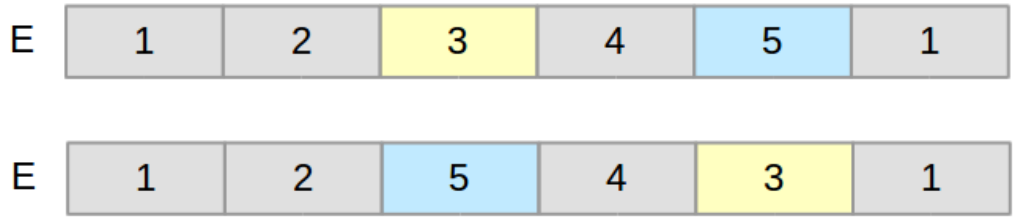


Abbildung 3.5: Beispiel: Mutation eines Individuums

Hierbei werden per Zufall die beiden Knoten 3 und 5 ausgewählt. Deren Position in der Liste wird daraufhin einfach vertauscht, wodurch die Lösung unter Umständen völlig neue Möglichkeiten in die Population einbringt.

3.1.6 Exchange

Die bisher vorgestellten Phasen beschreiben einen genetischen Algorithmus aus dem Lehrbuch, der rein single threaded funktioniert. Um diesen Algorithmus nun zu parallelisieren wird er um eine weitere Phase erweitert.
Die Problematik beim TSP ist, dass die Komplexität des Lösungsraumes exponentiell mit der Größe des verwendeten Graphen steigt (siehe Kapitel 2). Um das TSP für größere Graphen mit einem genetischen Algorithmus zu lösen muss die Größe der verwendeten Population vergrößert werden, damit die Population allein durch die schiere Anzahl an Lösungsmöglichkeiten nicht genetisch verhungert. Jedoch benötigen größere Populationen deutlich mehr Rechenleistung pro Generation und so können sehr große Graphen nur langsam gelöst werden.
Um diese Problematik zu lösen wurde die Population auf mehrere Prozesse verteilt. Jeder dieser Prozesse ist single threaded und berechnet den genetischen Algorithmus gemäß den bereits vorgestellten Phasen vollständig unabhängig von den anderen Prozesse. Am Ende jeder Generation treten jedoch alle Prozesse in die *Exchange* Phase ein. In dieser Phase senden alle Prozesse einen Anteil von 10% ihrer Population an einen anderen Prozess. Der sendende Prozess löscht diese Individuen aus seiner Population und der empfangende Prozesse fügt die erhaltenen Individuen in seine Population ein. Hierzu müssen alle Prozesse die gleiche

Populationsgröße besitzen, sodass nach dem Austausch die Population wieder gleich groß ist wie zuvor. Durch diesen Austausch von Individuen entsteht ein Austausch von Genmaterial, sodass die verschiedenen Populationen der Prozesse näherungsweise wie eine einzige große Population funktionieren. So agieren 10 Prozesse mit einer Population von 1000 Individuen so wie ein einziger Prozess mit 10000 Individuen. Darüber hinaus kann jeder Prozess die rechenintensiven Phasen auf einem separaten Kern mit einer Populationsgröße von 1000 statt 10000 ausführen, sodass die Prozessgemeinschaft abzüglich des Kommunikationsoverheads näherungsweise so schnell abläuft wie ein einzelner Prozess mit nur 1000 Individuen.
In Abbildung 3.6 ist zu sehen, wie der Austausch der Individuen mit fünf Prozessen funktioniert.

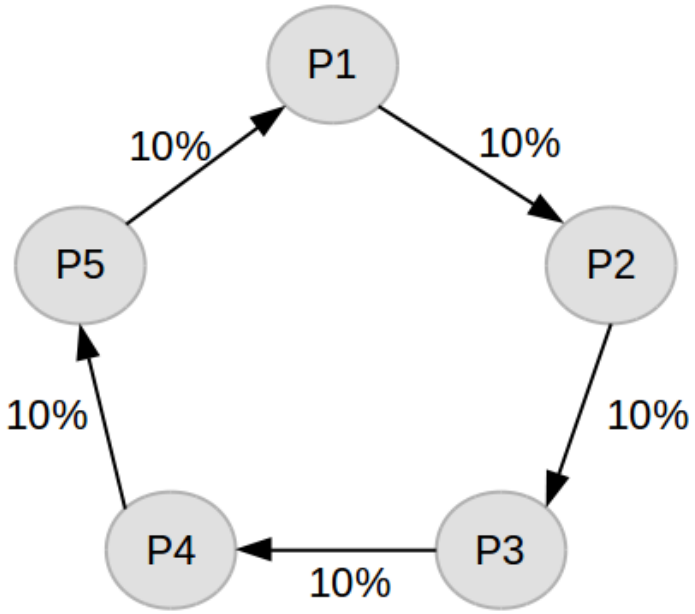


Abbildung 3.6: Beispiel: Austausch der Individuen

Hierbei wird ein Ringaustauschverfahren verwendet. Jeder Prozess hat eine eindeutige ID und gibt seine Individuen an den jeweils nächsten Prozess weiter, also P1 senden an P2, P2 an P3, usw. So wird ein kompliziertes Kommunikationsverfahren zur Findung von Partnerprozessen vermieden, um den Netzwerkoverhead möglichst niedrig zu halten.

4 Technologie

Programmiersprache; json; build system; unit test; github; travis CI; gnuplot etc.

4.1 MPI

wie funktioniert mpi prinzipiell?; p2p verbindung; ssh; broadcast, send, receive

5 Testumgebung

MAC pool; programmierung auf linux;

6 Ergebnisse

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Lee Jacobsen. Applying a genetic algorithm to the traveling salesman problem. http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5, 2012. Accessed: 20.05.2016.

[2] Lee Jacobsen. Creating a genetic algorithm for beginners. http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3, 2012. Accessed: 20.05.2016.

[3] Dr Rong Qu. Artificial intelligence search methodologies. Artificial Intelligence 2, Lecture HTWG Konstanz, 2014.