Evolutionäre Algorithmen - WS2012/13

Übungsblatt 3

Sebastian Nuck

6. Dezember 2012

1 Herangehensweise

1.1 Problemstellung und Vorüberlegungen

Die zweite Projektaufgabe besteht darin, einen Mutator und einen Crossover für einen existierenden genetischen Algorithmus zu implementieren. Dabei steht das Problem als Permutation zur Verfügung.

Allein der Fakt, dass es sich um eine Permutation handelt, verlangt nach speziellen Mutationen und Crossover um die Permutation nicht zu zerstören. So ist es beispielsweise nicht möglich einen einfachen 1-Punkt-Crossover, welcher den ersten Teil der ersten und den zweiten Teil der zweiten Permutation kombiniert zu implementieren. Dieser würde die Permutation zerstören.

1.2 Implementierte Mutationen

1.2.1 Random Swap Mutation

Diese Mutation war schon im Programm implementiert und wurde für Tests verwendet. Sie wurde allerdings nicht für die finale Konfiguration genutzt.

1.2.2 Scramble Mutation

Inhalt dieser Mutation ist das zufällige Vermischen eines zufällig ausgewählten Bereichs innerhalb der Mutation. Dadurch, dass keine neuen Zahlen hinzugefügt werden, sondern nur die Elemente der Permutation neu gemischt werden, bleibt sie gültig.

Im Laufe der Tests stellte sich heraus, dass diese Art der Mutation bei dieser Art Problem wenig nützlich ist. Die Permutation wird im schlechtesten Falle komplett vermischt und damit entsteht eine zu große Varietät. Infolgedessen wird auch diese Mutation nicht verwendet.

1.2.3 Inversion Mutation

Die letztendlich verwendete *Inversion Mutation* invertiert einen zufällig gewählten Teil der Permutation. Grundsätzlich hat sie folgenden Ablauf:

1. Zwei Indices werden zufällig generiert. Sie stellen dabei die Grenzen des zu invertierenden Bereichs dar.
2. Die beiden Elemente an den Stellen der Indices werden getauscht.
3. Der erste Index wird inkrementiert und er zweite dekrementiert.
4. Falls einer der beiden Indices am Ende bzw. am Anfang der Permutation angekommen ist springt der Index vom Ende auf den Anfang und umgekehrt. Damit kann auch der Fall behandelt werden, in dem der erste Index größer als der Zweite ist und damit über die Grenzen hinaus invertiert werden muss.
5. Der Tausch der Elemente wird solange wiederholt, bis beide Indices den selben Wert besitzen und damit alle Elemente invertiert wurden.

1.3 Implementierte Crossover

1.3.1 Order-Based Crossover

Als Crossover wird der *Order-Based Crossover* verwendet. Dieser wählt zufällig eine Anzahl von Elementen aus der ersten Permutation und fügt diese an den Stellen in der zweiten Permutation ein, wo sich die selben Zahlen befinden, allerdings in der Reihenfolge der ersten Permutation. Folgendes Beispiel soll dies verdeutlichen:

- Permutation $A = (\mathbf{5},1,\mathbf{2},\mathbf{3},4)$ (5,2,3 wurden zufällig gewählt)
- Permutation $B = (\mathbf{2},1,4,\mathbf{3},\mathbf{5})$
- Permut. $A \otimes B = (\mathbf{5},1,4,\mathbf{2},\mathbf{3})$

Der Algorithmus wurde wie folgt implementiert:

1. Eine zufällige Anzahl von Positionen aus der ersten Permutation werden in einer Liste gespeichert.
2. Es wird nun in der zweiten Permutation nach den selben Zahlen gesucht und auch deren Positionen in einer zweiten Liste gespeichert.
3. Die beiden Listen mit den Positionen der Zahlen werden nun sortiert.
4. Jetzt werden in einer Schleife die Zahlen an den Positionen der ersten Permutation in geordneter Reihenfolge an den Postionen der zweiten Liste gespeichert.

1.4 Ermittelte Werte

Länge	Anz. Iterationen	Populationsgr.	Turniergr.	Crossover-Wkt.	Durchschn. Güte
14	49 999	199	2	0.8	363.365 517 1
30	399 999	199	2	0.8	2108.122 98

Tabelle 1.1: Ermittelte Parameterwerte für die beiden geforderten Kombinationen