Seminar

„Optimierung industrieller Anwendungsprobleme“

im
Projekt
Fahrplanoptimierung

Thema

„Eine Fallstudie zu EA“

Autor
Oliver Harm

Inhaltsverzeichnis

1. Einleitung

Diesem Seminar liegt der Artikel „The transportation problem“ von Zbigniew Michalewicz, aus dem „Handbook of Evolutionary Computation release 97/1“ zu Grunde. In diesem Artikel wird ein evolutionärer Algorithmus, Genetic-2n, vorgestellt, der speziell zur Lösung des nicht linearen Transportproblems entwickelt wurde. Dabei wird vor allem auf die Besonderheiten des Algorithmus, die ihn von klassischen evolutionären Algorithmen unterscheidet, genauer eingegangen. Der Artikel erläutert zuerst allgemein das Transportproblem. Dann werden die wichtigsten Komponenten des Algorithmus, Genetic-2, vorgestellt und an einigen Beispielen deren Vorgehensweise erklärt. Zum Schluß wird der vorgestellte Algorithmus, Genetic-2n, mit einem Standartoptimierungs-Tool GAMS verglichen. Für weiter Einzelheiten des Algorithmus wurde noch das Buch „Genetic Algorithms + Data Structures = Evolution Programs“ Kapitel 9 und den original Sourcecode von Michalewicz herangezogen.

Im folgenden Kapitel wird zuerst einmal das Transportproblem vorgestellt und an einem Beispiel veranschaulicht. Darauf folgt eine kurze Einführung in die genetischen Algorithmen um später die Unterschiede des Algorithmus, Genetic-2n, zu diesen zu erläutern. Wie auch im original Artikel werden die Besonderheiten von Genetic-2n ausführlich beschrieben, die Unterschiede zu klassischen genetischen Algorithmen aufgezeigt, und an Beispielen erklärt. Der anschließende Vergleich von Genetic-2n mit einem Standartoptimierungs-Tool dient als Test und zeigt das der Algorithmus funktioniert.

2. Das Transportproblem

2.1 Allgemeine Beschreibung

Bei dem Transportproblem handelt es sich um eines der einfachsten Optimierungsprobleme mit Contraints. Es geht darum, Waren von einer bestimmten Anzahl von Quellen, zu einer bestimmten Anzahl von Zielen zu transportieren. Dabei steht an jeder Quelle ein bestimmter Vorrat an Ware zur Verfügung, und jedes Ziel hat einen bestimmten Bedarf an dieser Ware. Bei den Waren handelt es sich nur um eine Art von Ware, es wird also nicht zwischen verschiedenen Arten von Waren unterschieden. Auf diese Weise ist es egal, von welcher Quelle ein Ziel seine Waren bezieht. Ein Ziel kann deshalb seinen Bedarf auch aus mehreren Quellen decken. Der Transport einer Ware von einer Quelle zu einem Ziel verursacht gewisse Kosten. Diese Kosten werden als Transportkosten bezeichnet. Summiert man diese Transportkosten für alle Warenflüsse einer Lösung eines Transportproblems auf, so erhält man die gesamten Transportkosten für diese Lösung des Transportproblems.

Das Ziel ist nun einen Transportplan, für ein vorgegebenes Transportproblem, zu erstellen, so daß die gesamten Transportkosten möglichst niedrig sind. Dabei sind bestimmte Contraints zu beachten. Zum Einen können von einer Quelle natürlich nur so viele Waren abtransportiert werden, wie an dieser Quelle Waren als Vorräte zur Verfügung stehen. Zum Anderen müssen mindestens so viel Waren zu einem Ziel transportiert werden, daß dessen Bedarf gedeckt wird. Dies setzt natürlich voraus, daß die Vorräte aller Quellen zusammen mindestens so groß sind, wie der Bedarf aller Ziele zusammen. Die Möglichkeit, daß insgesamt weniger Waren an den Quellen zur Verfügung stehen, als von den Zielen benötigt werden, wird bei dieser Art von Problem, wie auch schon der Verzicht auf verschiedene Waren, nicht berücksichtigt. Es müssen also immer genügend Vorräte zur Verfügung stehen, so daß alle Bedürfnisse erfüllt werden können. Im Folgenden wird nur noch das ausgeglichene Transportproblem behandelt. Man spricht von einem ausgeglichenen Transportproblem, wenn die Vorräte aller Quellen zusammen genauso groß sind, wie der Bedarf aller Ziele zusammen. Das heißt wenn die Bedürfnisse aller Ziele erfüllt wurden, dann sind die Vorräte aller Quellen erschöpft. Das Transportproblem läßt sich durch die Art seiner Kostenfunktion, die zur Berechnung der Transportkosten dient, als ein lineares oder nicht lineares Transportproblem klassifizieren. Man spricht von einem linearen Transportproblem, wenn die Kosten für den Transport von Waren, direkt proportional zur Anzahl der transportierten Waren sind, die Kostenfunktion also linear ist, sonst von einem nicht linearen Transportproblem. Im folgenden wird hauptsächlich der nicht lineare Fall betrachten.

2.2 Darstellung

Ein Transportproblem ist gegeben durch eine Menge von n Quellen und k Zielen.
Der Vorrat einer Quelle i ist gegeben durch source(i) wobei $i = 1,..,n$ ist. Der Bedarf eines Ziels j ist gegeben durch dest(j) wobei $j = 1,..,k$ ist. Eine $(n \times k)$ Matrix V, die den Transportplan enthält, stellt eine Lösung des Transportproblems dar. Jede Matrix dieser Form, deren Elemente reelle Zahlen sind, kann eine Lösung des Transportproblems sein.

$$V = \begin{pmatrix} x_{1,1} & \dots & x_{1,k} \\ . & & . \\ x_{n,1} & \dots & x_{nk} \end{pmatrix}$$

Jedes Element x_{ij} der Matrix V gibt an, wieviel Einheiten der Ware von Quelle i nach Ziel j Transportiert werden soll. Die Zeilen- und Spaltensummen der Matrix V müssen die Contraints des Transportproblems erfüllen. Die Zeilensummen dürfen die Vorräte der entsprechenden Quellen nicht überschreiten, bzw. im ausgeglichenen Fall gleich dem Vorrat sein.

$$\sum_{j=1}^{k} x_{ij} = \text{source}(i) \quad \text{für } i = 1,2,\dots,n$$

Die Spaltensummen der Matrix V müssen mindestens so groß sein, wie die Bedürfnisse der entsprechenden Ziele, bzw. im ausgeglichenen Fall genauso groß sein.

$$\sum_{i=1}^{n} x_{ij} = \text{dest}(j) \quad \text{für } j = 1,2,\dots,k$$

Als weiteres Constraint werden für alle Elemente x_{ij} der Matrix V nur positive Zahlen zugelassen.

$$x_{ij} \geq 0 \quad \text{für } i = 1,2,\dots,n \text{ und } j = 1,2,\dots k$$

Die Kostenfunktion $f_{ij}(x)$ bestimmt die Kosten für den Transport von x Einheiten einer Ware von der Quelle i zum Ziel j. Zur Kostenfunktion gehört eine Parametermatrix(Kostenmatrix) mit z.B. den Basiskosten für einen Transport von einer Quelle i zu einem Ziel j.

$$C = \begin{pmatrix} c_{1,1} & \dots & c_{1,k} \\ . & & . \\ c_{n,1} & \dots & c_{nk} \end{pmatrix}$$

Eine einfache lineare Kostenfunktion könnte wie folgt aussehen:

$$f_{ij}(x_{ij}) = c_{ij} x_{ij}$$

Für eine Lösung des Transportproblems in Form einer Matrix V können die gesamten Transportkosten für diese Lösung mit Hilfe der Kostenfunktion errechnet werden.

$$\text{gesamte Transportkosten} = \sum_{i=1}^{n}\sum_{j=1}^{k} f_{ij}(x_{ij})$$

Die gesamten Transportkosten geben an wie gut eine Lösung ist. Um so niedriger die gesamten Transportkosten für eine Matrix V sind, desto besser ist die Lösung. Bei der Behandlung des Transportproblems wird demzufolge folgendes Optimum gesucht.

$$\text{minimalste Transportkosten} = \min\left(\sum_{i=1}^{n}\sum_{j=1}^{k} f_{ij}(x_{ij})\right)$$

2.3 Beispiel

Folgendes Transportproblem ist gegeben durch:

Quellen : n = 3
Ziele : k = 4

Contraints:

source[1] = 10.0, source[2] = 5.0, source[3] = 10.0
dest[1] = 7.0, dest[2] = 5.0, dest[3] = 10.0, dest[4] = 3.0

Eine typische Lösung dieses Transportproblems sieht wie folgt aus (die graue Spalte und Zeile gehört nicht zur Lösungsmatrix V):

	7.0	**5.0**	**10.0**	**3.0**
10.0	0.0	5.0	2.0	3.0
5.0	5.0	0.0	0.0	0.0
10.0	2.0	0.0	8.0	0.0

An den Zeilen- und Spaltensummen(grau hervorgehoben) ist schnell zu überprüfen ob die Constraints erfüllt sind. Mit der folgenden Kostenfunktion und Kostenmatrix können die gesamten Transportposten für diese Lösung errechnet werden.

$$f_{ij}(x_{ij}) = c_{ij}x_{ij} \text{ und } C = \begin{Bmatrix} 2 & 4 & 3 & 2 \\ 3 & 2 & 2 & 5 \\ 7 & 4 & 1 & 6 \end{Bmatrix}$$

Die gesamten Transportkosten für dies Lösung belaufen sich auf:

$$\sum_{i=1}^{3}\sum_{j=1}^{4} f_{ij}(x_{ij}) = 69.0$$

3. Klassische evolutionäre Algorithmen

Um die Abweichungen des evolutionären Algorithmus, Genetic-2n, zu den klassischen genetischen Algorithmen zu verdeutlichen, wird in diesem Kapitel die allgemeine Vorgehensweise der genetischen Algorithmen kurz erläutert. Dies ist nur ein kurze Einführung und soll den groben Ablauf und die Vorgehensweise andeuten.

3.1 Ablauf eines evolutionären Algorithmus

Die meisten evolutionären Algorithmen halten sich an das folgende Ablaufschema, das den Prozeß pro Generation beschreibt.

Zu Beginn wird eine Startpopulation erzeugt, diese enthält eine Menge von Lösungen des Problems. Alle Lösungen in der Startpopulation werden bewertet, um festzustellen, wie gut sie das Problem lösen.
Nun kann mit der ersten Generation begonnen werden. Aus der Startpopulation $P(t)$ werden durch bestimmte Selektionsverfahren eine Menge P' von Lösungen ausgewählt, die als Eltern für die Rekombination benötigt werden. Aus der Menge der Eltern P' wird mit Hilfe von Rekombinationsmethoden eine Menge P'' von neuen Lösungen, den Kindern, erzeugt. Diese neu erzeugten Lösungen werden eventuell noch einer Mutation unterzogen und können dann bewertet werden. Aus der Menge der Kinder P'' wird nun eine Menge von Lösungen ausgewählt, um eine neue Population $P(t+1)$ für die nächste Generation zu bilden. Dieser Vorgang wiederholt sich bis eine bestimmte Endbedingung erfüllt ist. t dient dabei als Generationszähler.

Allgemeines Schema eines evolutionären Algorithms:

$t \leftarrow 0$
erzeuge Population $P(t)$
bewerte Population $P(t)$
while Endbedingung nicht erfüllt **do**
    selektiere Eltern P' aus $P(t)$
    $t \leftarrow t+1$
    erzeuge Kinder P'' aus P'
    führe Mutationen an P'' durch
    bewerte Population P''
    selektiere $P(t)$ aus P''
od

3.2 Datenstruktur

Die Lösungen werden bei klassischen genetischen Algorithmen in Bitstrings als Reihenfolge von Einsen und Nullen mit Hilfe von bestimmten Kodierverfahren, um die Daten auf Wertebereiche einzuschränken, kodiert. Diese Bitstrings oder auch Chromosomen genannt, in Anlehnung an die Biologie, sind im Allgemeinen weitgehend unabhängig von dem zu bearbeitenden Problem und bieten wenig Möglichkeiten bestimmte Constraints vollständig zu berücksichtigen. Auf diesen allgemeinen Bitstrings setzten dann die Operatoren an.

3.3 Operatoren

Die Operatoren klassischer genetischen Algorithmen sind sehr allgemein gehalten und arbeiten auf einem oder mehreren Bitstrings, ohne Kenntnisse der darin kodierten Daten oder Informationen über das gestellte Problem zu verwenden. Rekombinations-Operatoren wie z.B. Crossover brechen die Bitstrings zweier Lösungen an einer beliebigen Stelle auseinander und setzten die Bitstrings der beiden Lösungen über Kreuz wieder zusammen. Auf diese Weise entstehen zwei neue Lösungen, die jeweils einen Teil der Eigenschaften jedes Elternteils beinhalten. Dadurch sollen die guten Eigenschaften zweier Lösungen miteinander vereinigt werden, um somit eine bessere Lösung zu gewinnen. Bei den Mutations-Operatoren wie z.B. Bitflipping werden einfach einzelne Bits aus dem Bitstring invertiert um auf diese Weise geringfügige Änderungen oder aber auch größere Sprünge an den Lösungen zu erreichen. Die Auswirkung dieser Manipulationen auf die Lösungen werden erst bei der Dekodierung der Bitstring erkennbar.

3.4 Probleme bei der Optimierung

Die Probleme die sich bei klassischen genetischen Algorithmen z.B. im Bezug auf das Transportproblem ergeben liegen auf der Hand. Wie schon bei der Darstellung erwähnt, sind Constraint nur schwer oder gar nicht zu berücksichtigen, da die Algorithmen zu allgemein gehalten sind. Die Darstellung von Lösungen als Bitstrings in den meisten Fällen nichts mit der natürlichen Darstellung des Problems zu tun hat. Dadurch das Contraints nur schwer mit einzubeziehen sind, entstehen oft ungültige und damit unbrauchbare Lösungen. Um diese Dinge zu vermeiden müssen speziell ein bestimmte Probleme oder Problemklassen angepaßte evolutionäre Algorithmen entwickelt werden.

4. Der evolutionäre Algorithmus „Genetic-2n“

Um die Probleme eines klassischen evolutionären Algorithmus bei der Lösung des Transportproblems zu vermeiden, wurde von Vignuax und Michalewicz(1991) ein spezieller evolutionärer Algorithmus für das nicht lineare Transportproblem entwickelt.
Die Ergebnisse bisheriger Systemen zur Lösung des Transportproblems, die auf der Darstellung der Lösungen als Bitstrings basierten, waren meist nicht zufriedenstellend. Sie lieferten oft ungültige Lösung, da die Contraints nicht ausreichend berücksichtigt werden konnten. Deshalb wurde bei Genetic-2n als Darstellung einer Lösung eine zweidimensionale Struktur, die der natürlichen Darstellung des Problems sehr nahe liegt, verwendet. Auch die Operatoren wurden der Darstellung angepaßt, so daß sie die Constraints berücksichtigen können. Auf diese Weise erzeugt der Algorithmus Genetic-2n nur gültige Lösungen, die alle Constraints erfüllen.
Genetic-2n ist eine modifizierte Version von Genetic-2, die für den linearen Fall des Transportproblems entwickelt wurde. Diese basierte bereits auf der Darstellung der Lösungen als zweidimensionale Struktur, im Gegensatz zu dessen Vorgänger Gentic-1, das auf Vektoren basierte. Die Initialisierungsroutine und der Mutations-Operator ‚mutation-1‘ sind identisch mit denen von Genetic-2, die Änderungen bestehen in der modifizierten Initialisierungsroutine, dem Mutations-Operator ‚mutation-2‘ und dem Crossover-Operator.

Im Folgenden werden die Besonderheiten von Genetic-2n im Detail vorgestellt und an Beispielen erläutert.

4.1 Der Algorithmus

Der Ablauf von Genetic-2n entspricht im Wesentlichen dem der klassischen evolutionären Algorithmen. Der einzige Unterschied besteht in der Verwendung der Mutations-Operatoren. Anstatt die durch Rekombination erzeugten Kinder einer Mutation zu unterziehen, wird für die Mutations-Operatoren ein Elter gewählt, davon eine Kopie erstellt und diese mutiert. Auf diese Weise werden durch die Mutation neue Kinder erzeugt, sie dient also als eigenständige Reproduktionsmethode wie die Rekombinations-Operatoren. Die Auswahl der Eltern und die Selektion der Lösungen für die Population der nächsten Generation, geschieht meist über fitnessproportionale Selektion. Allerdings ist auch jede andere Selektionsmethode denkbar.

Genetic-2n besteht aus folgenden besonderen Komponenten:

Initialisierungsroutine

Initialisierungsroutine (modifiziert)

Mutations-Operator ‚mutation-1‘

Mutations-Operator ‚mutation-2‘

Rekombinations-Operator ‚Arithmetischer Crossover‘

Von der Initialisierungsroutine gibt es zwei verschiedene Versionen, auf die später noch genauer eingegangen wird. Zur Initialisierung der Startpopulation zu Beginn des Algorithmus wird für jede erzeugte Lösung per Zufall entschieden, welche der beiden Initialisierungsroutine verwendet wird. Zur Selektion wird wie schon erwähnt eine fitnessproportionale Selektionsmethode verwendet. Über einen Parameter kann eine Wahrscheinlichkeit angegeben werden, mit der entschieden wird, welcher Mutations-Operator bei einer Mutation verwendet werden soll.

4.1.1 Datenstruktur

Zur Darstellung einer Lösung wird bei Genetic-2n eine Matrix V verwendet, in der jedes Element x_{ij} eine reelle Zahl und den Transportfluß darstellt, wie sie in Kapitel 2.2 zur Darstellung einer Lösung vorgestellt wurde. Diese problemspezifische Darstellung ermöglicht die Verwendung von speziell angepaßten Operatoren, die auf den Elementen der Matrix arbeiten, und dies ermöglicht die Einhaltung der Constraints.

4.1.2 Die Initialisierung

Die Initialisierung dient dazu, zu Beginn des Algorithmus ausreichend viele Lösungen für die Startpopulation zu erzeugen, darüber hinaus ist dieses Routine Hauptbestandteil der beiden Mutations-Operatoren. Die Prozedur ‚initialization‘ erzeugt für die vorgegebenen Constraints eine individuelle Lösung, die alle Contraints erfüllt.

Als Eingabe benötigt die Prozedur ‚initialization‘ zwei Arrays $\text{dest}[k]$ und $\text{source}[n]$, wobei $\text{dest}[k]$ die Bedürfnisse der Ziele und $\text{source}[n]$ die Vorräte der Quellen enthält. Als Ergebnis liefert die Prozedur eine Matrix, die eine Lösung, des durch $\text{dest}[k]$ und $\text{sour}[n]$ angegebenen Transportproblems, darstellt. Diese Matrix enthält höchstens $k+n-1$ Elemente die nicht Null sind und erfüllt alle Contraints.

Folgendes Schema beschreibt die Initialisierungsroutine:

Eingabe: Arrays $\text{dest}[k], \text{source}[n]$

Ausgabe: Eine $(n \times k)$ Matrix (x_{ij}) mit $x_{ij} \geq 0 \, \forall \, i$ und j, $\sum_{\text{j}=1}^{\text{k}} x_{ij} = \text{sour}[i]$ für i = 1,2,...,n ,

und $\sum_{i=1}^{n} x_{ij} = \text{dest}[j]$ für j=1,2,...,k , d.h. alle Contraints erfüllt.

procedure initialization

markiere alle Elemente der Matrix als nicht besucht

while es gibt ein nicht besuchtes Element x_{ij} **do**

set val $\leftarrow \min(\text{source}[i], \text{dest}[j])$

set $\text{x}_{\text{ij}} \leftarrow$ val

set $\text{source}[i] \leftarrow \text{source}[i] -$ val

set $\text{dest}[j] \leftarrow \text{dest}[j] -$ val

od

Das folgende Beispiel beschriebt, wie die Prozedur ‚initialization‘ eine Lösung erzeugt. Wir betrachten folgendes Problem mit 3 Quellen und 4 Zielen:

```
source[1] = 10.0, source[2] = 5.0, source[3] = 10.0
dest[1] =  7.0, dest[2] = 5.0, dest[3] = 10.0, dest[4] = 3.0
```

Als erstes wird zufällig das Element $x_{2,1}$ ausgewählt und dieses als besucht markiert. Der Wert *val* ergibt sich aus dem Minimum von $\text{source}[2] = 5.0$ und $\text{dest}[1] = 7.0$, somit ist $val = 5.0$. Dieser Wert wird in die Matrix eingetragen, somit ist $x_{2,1} = 5.0$. Danach wird der Wert *val* von source[2] und $\text{dest}[1]$ subtrahiert, so daß danach $\text{source}[2] = 0.0$ und $\text{dest}[1] = 2.0$ ist. Dies wird für die nächsten 3 zufällig ausgewählten Elemente $x_{3,1}$, $x_{1,1}$ und $x_{3,3}$ der Matrix, die bis dahin noch nicht als besucht markiert sind, wiederholt. Die daraus resultierende Matrix hat folgenden Inhalt, wobei die grauen Zeilen und Spalten nicht zur Matrix gehören, sondern die Arrays $\text{dest}[k]$ und $\text{source}[n]$ darstellen:

	0.0	**5.0**	**2.0**	**3.0**
10.0	0.0			
0.0	5.0			
0.0	2.0		8.0	

Man beachte, dies sind die Werte von $\text{dest}[j]$ und $\text{source}[i]$ nach 4 Iterationen. Angenommen die weiteren zufällig besuchten Elemente seien $x_{2,2}$, $x_{1,4}$, $x_{3,4}$, $x_{2,3}$, $x_{1,3}$, $x_{3,2}$, $x_{2,4}$ und $x_{1,2}$, dann ergibt sich folgende Matrix:

	0.0	**0.0**	**0.0**	**0.0**
0.0	0.0	5.0	2.0	3.0
0.0	5.0	0.0	0.0	0.0
0.0	2.0	0.0	8.0	0.0

Nachdem alle 12 Iterationen durchgeführt wurden und somit jedes Element einmal besucht wurde, sind alle $\text{dest}[j]$ und $\text{source}[i]$ gleich 0.0. Das heißt die Vorräte aller Quellen wurden abtransportiert und die Bedürfnisse alle Ziele wurden befriedigt und somit erfüllt die Lösung auch die Constraints.

4.1.3 Mutations-Operatoren

Der Algorithmus Genetic-2n besitzt 2 Mutations-Operatoren. Der Mutations-Operator ‚mutation-1‘ ist identisch mit dem von Genetic-2 für den linearen Fall. Er versucht so viele 0.0 Einträge in der Matrix zu erreichen wie nur möglich. Außerdem besitzt er die Eigenschaft, daß wenn er auf einer Lösung angewendet wird, deren Matrix nur aus Elementen mit ganzen Zahlen besteht, das erzeugte Kind auch nur Elemente aus ganzen Zahlen besitzt. Der zweite Mutations-Operator ‚mutation-2‘ wurde dahingehend modifiziert, die 0.0 Einträgen zu vermeiden. Dieser erzeugt im Gegensatz zum ersten Mutations-Operator Elemente aus reellen Zahlen. Beide Mutations-Operatoren erzeugen aus einem Elter ein neues Kind. Durch einen Parameter p_m kann das Verhältnis festgelegt werden, wie die beiden Operatoren zum Einsatz kommen.

4.1.4 Der Mutations-Operator ‚mutation-1‘

Eine $(n \times k)$ Matrix $\mathrm{V} = (\mathrm{x}_{\mathrm{ij}})$ dient als Elter für die Mutation. Aus dieser wird per Zufall eine $(p \times q)$ Teilmatrix entnommen. Dazu wird eine Untermenge $\{i_1, i_2, \ldots, i_p\}$ aus der Menge $\{1,2,\ldots,n\}$ der Zeilen und eine Untermenge $\{j_1, j_2, \ldots, j_q\}$ aus der Menge $\{1,2,\ldots,k\}$ der Spalten der Matrix V ausgewählt, so daß $2 \le p \le n, 2 \le q \le k$ ist. Wobei p und q per Zufall bestimmt werden. Auf diese Weise läßt sich eine $(p \times q)$ Teilmatrix $\mathrm{W} = (w_{ij})$ aus den Elementen der Matrix V auf folgende Art bilden: ein Element $x_{ij} \in \mathrm{V}$ ist in W, genau dann wenn $i \in \{i_1, i_2, \ldots, i_p\} \wedge j \in \{j_1, j_2, \ldots, j_q\}$ (Das Element x_{ij} der Matrix V wird in der Matrix W in der r-ten Zeile und s-ten Spalte eingetragen, wenn $i = i_r \wedge j = j_s$ ist).
Aus der so konstruierten Matrix W können die Werte für $\mathrm{source}_{\mathrm{w}}[i]$ und $\mathrm{dest}_{\mathrm{w}}[j]$ mit $(1 \le i \le p, 1 \le j \le q)$ berechnet werden:

$$\mathrm{source}_{\mathrm{w}}[i] = \sum_{j \in \{j_1, j_2, \ldots, j_q\}} x_{ij} \quad 1 \le i \le p$$

$$\mathrm{dest}_{\mathrm{w}}[j] = \sum_{i \in \{i_1, i_2, \ldots, i_p\}} x_{ij} \quad 1 \le j \le q$$

Mit der Prozedur ‚initialization‘ werden der Matrix W anschließend neue Werte zugewiesen, so daß die Contraints $\mathrm{source}_{\mathrm{w}}[i]$ und $\mathrm{dest}_{\mathrm{w}}[j]$ erfüllt sind. Dann werden die Elemente der Matrix W wieder an die entsprechenden Stellen in der Matrix V, woher sie ursprünglich stammten, eingetragen. Die Matrix V erfüllt nach der Mutation auch weiterhin die globalen Contraints ($\mathrm{source}[i]$ und $\mathrm{dest}[j]$), sofern sie diese zuvor erfüllte.

Das folgende Beispiel veranschaulicht den Mutations-Operator ‚mutation-1'.
Gegeben ist ein Transportproblem mit 3 Quellen, 4 Zielen und folgenden Contraints:

```
   source[1] = 10.0, source[2] = 5.0, source[3] = 10.0
dest[1] = 7.0, dest[2] = 5.0, dest[3] = 10.0, dest[4] = 3.0
```

An folgender Matrix V, die als Elter für die Mutation dient, soll die Mutation ‚mutation-1' durchgeführt werden.

	7.0	5.0	10.0	3.0
10.0	0.0	5.0	2.0	3.0
5.0	5.0	0.0	0.0	0.0
10.0	2.0	0.0	8.0	0.0

Zuerst werden p und q per Zufall bestimmt. Angenommen $p = 2$ und $q = 3$ wurden per Zufall gewählt, so bedeutet dies, das aus der Matrix V 2 Zeilen und 3 Spalten ausgewählt werden müssen. Es werden also 2 Zeilen $\{1,2\}$ und 3 Spalten $\{1,3,4\}$ per Zufall ausgewählt. Die ausgewählten Werte sind in der obigen Matrix fett hervorgehoben. Aus diesen Werten setzt sich nun die Matrix W zusammen.

$W = \{$

0.0	2.0	3.0
5.0	0.0	0.0

Für die erzeugte Matrix W werden nun die Contraints $\text{source}_w[i]$ und $\text{dest}_w[j]$ errechnet.

$\text{source}_w[1] = 5.0$, $\text{source}_w[2] = 5.0$
$\text{dest}_w[1] = 5.0$, $\text{dest}_w[2] = 2.0$, $\text{dest}_w[3] = 3.0$

Mit den Contraints $\text{source}_w[i]$ und $\text{dest}_w[j]$ kann die Matrix W mit der Prozedur ‚initilaization' neu bestimmt werden. Nach der Reinitialisierung könnte die Matrix W folgende Werte enthalten:

$W = \{$

2.0	0.0	2.0
3.0	0.0	0.0

Zur Initialisierung der Matrix W wurden die Elemente in folgender Sequenz besucht: $\langle 5,1,4,6,2,3 \rangle$. Nun können die Elemente der Matrix W wieder an die Stellen in der Matrix V eingefügt werden, wo sie entnommen wurde. So ergibt sich nach der Mutation der Matrix W das folgende Kind:

0.0	0.0	5.0	0.0	3.0
0.0	5.0	5.0	0.0	0.0
0.0	0.0	0.0	2.0	3.0
3.0	2.0	0.0	8.0	0.0

Da die Contraints, die zur Erzeugung der Matrix W verwendet wurden, aus den Elementen, die ursprünglich aus der Matrix V stammten, errechnet wurden, erfüllt die Matrix V die Contraints $\text{source}[i]$ und $\text{dest}[j]$ nach der Mutation immer noch.

4.1.5 Der Mutations-Operator ‚mutation-2‘

Der Mutations-Operator ‚mutation-2‘ ist weitgehend identisch zu ‚mutation-1‘. Der einzige Unterschied besteht darin, daß zur Reinitialisierung der Matrix W eine modifizierte Version der Prozedur ‚initialization‘ verwendet wird. Die Initialisierungsroutine unterscheidet sich von der zuvor beschriebenen in folgenden Punkten:

Die Anweisung in Zeile 4

$$\text{set val} \leftarrow \min(\text{source}[i], \text{dest}[j])$$

wird ersetzt durch die Anweisungen

$\text{set val}_1 \leftarrow \min(\text{source}[i], \text{dest}[j])$
if (x_{ij} last i) oder (x_{ij} last j) **then**
  $\text{val} \leftarrow \text{val}_1$
else
  $\text{set val} \leftarrow \text{Zufallszahl(reell) imBereich } \langle 0, \text{val}_1 \rangle$
fi

(Wobei x_{ij} last i bedeutet, x_{ij} ist das letzte nicht besuchte Element in der Zeile i und x_{ij} last j bedeutet, x_{ij} ist das letzte nicht besuchte Element in der Spalte j.)

Durch diese Änderung wird die Matrix W bei der Reinitialisierung nicht wie zuvor mit ganzen Zahlen und vielen Nullen gefüllt, sondern mit reellen Zahlen und wenigen Nullen besetzt. Wobei natürlich der Wert 0.0 auch auftreten kann. Nach dieser Änderung können bei der Initialisierung von Matrizen allerdings die Constraints verletzt werden. Deshalb muß bei Prozedur ‚initialization‘ noch eine Zeile

mache notwendige Additionen

am Ende der gesamten Routine, außerhalb der Schleife, eingefügt werden. Dadurch, daß die Elemente meistens aus Zufallszahlen im Bereich $\langle 0, \text{val}_1 \rangle$ bestehen, ist es möglich, daß nach dem Durchlauf der Schleife, in einer Zeile, also bei einer Quelle Waren übrig bleiben, und dementsprechend auf nicht alle Bedürfnisse der Ziele gedeckt wurden. Dieser Rest muß durch eine oder mehrere Additionen auf die Elemente der entsprechenden Zeile verteilt werden, so daß alle Constraints erfüllt sind.

Die modifizierte Version der Initialisierungsroutine sieht dann wie folgt aus.

procedure initialization
  markiere alle Elemente der Matrix als nicht besucht
  while es gibt ein nicht besuchtes Element x_{ij} **do**
    set $\mathrm{val}_1 \leftarrow \min(\mathrm{source}[i], \mathrm{dest}[j])$
    if (x_{ij} last i) oder (x_{ij} last j) **then**
      $\mathrm{val} \leftarrow \mathrm{val}_1$
    else
      set $\mathrm{val} \leftarrow$ Zufallszahl(reell) imBereich $\langle 0, \mathrm{val}_1 \rangle$
    fi
    set $x_{ij} \leftarrow \mathrm{val}$
    set $\mathrm{source}[i] \leftarrow \mathrm{source}[i] - \mathrm{val}$
    set $\mathrm{dest}[j] \leftarrow \mathrm{dest}[j] - \mathrm{val}$
  od
  mache notwendige Additionen

Das folgende Beispiel veranschaulicht die modifizierte Prozedur ‚initialization‘ für den Mutations-Operator ‚mutation-2‘.

Gegeben ist ein Transportproblem mit 3 Quellen, 4 Zielen und folgenden Contraints:

```
source[1] = 10.0, source[2] = 5.0, source[3] = 10.0
dest[1] =  7.0, dest[2] = 5.0, dest[3] = 10.0, dest[4] = 3.0
```

Angenommen die Sequenz der zufällig ausgewählten Elemente sei $x_{2,2}$, $x_{1,3}$, $x_{1,1}$, $x_{1,2}$, $x_{2,4}$, $x_{3,2}$, $x_{2,3}$, $x_{1,4}$, $x_{3,3}$, $x_{2,1}$, $x_{3,1}$, $x_{3,4}$. Für das erste Element kann der val_1 berechnet werden, durch $val_1 = \min(\mathrm{source}[2] = 5.0, \mathrm{dest}[2] = 5.0)$, wodurch sich $val_1 = 5.0$ ergibt. Da das Element $x_{2,2}$ nicht das letzte zu besetzende Element in der Zeile $i = 2$ und nicht das letzte zu besetzende Element in der Spalte $j = 2$ ist, wird für den Wert val nun eine Zufallszahl im Bereich $\langle 0, val_1 \rangle$ bestimmt. Angenommen für val ergibt sich die Zufallszahl 2.3, so wird nun dem Element $x_{2,2}$ der Wert $val = 2.3$ zugewiesen. Nach den ersten 5 Elementen aus der oben genannten Sequenz ergibt sich folgende Matrix:

	5.9	0.7	9.4	1.8
6.3	1.1	2.0	0.6	
1.5		2.3		1.2
10.0				

Für das 6. Element $x_{3,2}$ in der 3. Zeilen und 2. Spalte wird für val_1 wieder das Minimum von $\mathrm{source}[3] = 10.0$ und $\mathrm{dest}[2] = 0.7$ bestimmt, da das Element $x_{3,2}$ das letzte zu besetzende Element in der 2. Spalte ist. Nun wird für val keine Zufallszahl bestimmt, sondern, wie in der ursprünglichen Version der Initialisierungsroutine, einfach das bereits für val_1 bestimmte Minimum val zugewiesen. Somit wird dem Element $x_{3,2}$ schließlich der Wert 0.7 zugewiesen.
Nachdem alle Felder der Matrix neu besetzt wurden, und dabei die komplette Sequenz von Zufallszahlen, die für val benötigt wurden $\langle 2.3, 0.6, 1.1, 2.0, 1.2, 0.7 \rangle$ sei, ergibt sich folgende Matrix:

	4.5	0.0	0.0	0.0
4.5	1.1	2.0	0.6	1.8
0.0	0.8	2.3	0.7	1.2
0.0	0.6	0.7	8.7	0

Es ist leicht zu erkennen das diese Matrix die Constraints nicht erfüllt. Deshalb muß bei diesem Beispiel zu dem Element $x_{1,1}$ der Wert 4.5 dazu addiert werden um die Constraints zu erfüllen. Somit ergibt sich als Kind der Mutation folgende Matrix:

	0.0	0.0	0.0	0.0
0.0	5.6	2.0	0.6	1.8
0.0	0.8	2.3	0.7	1.2
0.0	0.6	0.7	8.7	0

4.1.6 Arithmetischer Crossover

Der Crossover benötigt zwei Eltern, die Matrizen U und V. Durch den Crossover werden aus den Matrizen U und V zwei Kinder X und Y auf folgende Art erzeugt:

$$X = c_1 U + c_2 V$$

$$Y = c_1 V + c_2 U$$

Wobei $c_1, c_2 \geq 0$ und $c_1 + c_2 = 1$ sein muß. Diese beiden Constraints bewirken, daß wenn die Eltern U und V die Contraints des Transportproblems erfüllen, dann erfüllen die so erzeugten Kinder X und Y auch die Constraints.

Folgendes Beispiel zeigt wie der Crossover funktioniert:
Gegeben ist ein Transportproblem mit 3 Quellen, 4 Zielen und folgenden Contraints:

```
source[1] = 10.0, source[2] = 5.0, source[3] = 10.0
dest[1] =  7.0, dest[2] = 5.0, dest[3] = 10.0, dest[4] = 3.0
```

Angenommen als Eltern wurden folgende zwei Matrizen U und V ausgewählt.

$$U = \begin{Bmatrix} 2.0 & 3.0 & 3.0 & 2.0 \\ 1.0 & 1.0 & 2.0 & 1.0 \\ 4.0 & 1.0 & 5.0 & 0 \end{Bmatrix} \qquad \mathrm{V} = \begin{Bmatrix} 1.0 & 4.0 & 5.0 & 0 \\ 2.0 & 0 & 1.0 & 2.0 \\ 4.0 & 1.0 & 4.0 & 1.0 \end{Bmatrix}$$

Für die Parameter wurden $c_1 = 0.35$ und $c_2 = 0.65$ gewählt.
Die zwei Kinder können nun wie folgt errechnet werden.

$$X = 0.35 * \begin{Bmatrix} 2.0 & 3.0 & 3.0 & 2.0 \\ 1.0 & 1.0 & 2.0 & 1.0 \\ 4.0 & 1.0 & 5.0 & 0 \end{Bmatrix} + 0.65 * \begin{Bmatrix} 1.0 & 4.0 & 5.0 & 0 \\ 2.0 & 0 & 1.0 & 2.0 \\ 4.0 & 1.0 & 4.0 & 1.0 \end{Bmatrix} = \begin{Bmatrix} 1.35 & 3.65 & 4.3 & 0.7 \\ 1.65 & 0.35 & 1.35 & 1.65 \\ 4 & 1 & 4.35 & 0.65 \end{Bmatrix}$$

$$Y = 0.35 * \begin{Bmatrix} 1.0 & 4.0 & 5.0 & 0 \\ 2.0 & 0 & 1.0 & 2.0 \\ 4.0 & 1.0 & 4.0 & 1.0 \end{Bmatrix} + 0.65 * \begin{Bmatrix} 2.0 & 3.0 & 3.0 & 2.0 \\ 1.0 & 1.0 & 2.0 & 1.0 \\ 4.0 & 1.0 & 4.0 & 0 \end{Bmatrix} = \begin{Bmatrix} 1.65 & 3.35 & 3.7 & 1.3 \\ 1.35 & 0.65 & 1.65 & 1.35 \\ 4 & 1 & 4.65 & 0.35 \end{Bmatrix}$$

Wie man sieht erfüllen die erzeugten Kinder X und Y die Constraints des Transportproblems, wie die Eltern.

	7.0	5.0	10.0	3.0
10.0	1.35	3.65	4.3	0.7
5.0	1.65	0.35	1.35	1.65
10.0	4	1	4.35	0.65

	7.0	5.0	10.0	3.0
10.0	1.65	3.35	3.7	1.3
5.0	1.35	0.65	1.65	1.35
10.0	4	1	4.65	0.35

4.2 Eigenschaften

In folgenden Kapitel werden ein paar Eigenschaften des Algorithmus Genetic-2n erläutert.

4.2.1 Unausgeglichenes Transportproblem

Obwohl in dem zu Grunde liegenden Artikel ausschließlich von einem ausgeglichenen Transportproblem ausgegangen wird, ist anzumerken das der Algorithmus Genetic-2n auch mit einem unausgeglichenen Transportproblem zurecht kommt. Die einzige Stelle an der die Eigenschaft, ob das Transportproblem ausgeglichen oder unausgeglichen ist, zu tragen kommt ist in der Initialisierungsroutine, zu Beginn des Algorithmus bei der Erstellung der ersten Lösungen für die Startpopulation. Im weiteren Verlauf des Algorithmus wird nur noch auf diesen Lösungen gearbeitet und die globalen Constraints $\text{source}[i]$ und $\text{dest}[j]$werden nicht mehr benötigt. Nur bei der modifizierten Initialisierungsroutine, an der Stelle „mache notwendige Additionen“ ist nicht klar in wie weit die Contraints an dieser Stelle benötigt werden. An dieser Stelle muß durch eine entsprechende Implementierung dafür gesorgt werden, das die Constraints erfüllt werden. Da bei beiden Versionen der Initialisierungsroutine immer nur maximal das Minimum zwischen der Quelle und dem Ziel für die Zuweisung verwendet wird, bleiben überschüssige Vorräten an den Quellen einfach übrig, bzw. der Mehrbedarf der Ziele wird nicht befriedigt. Bis auf die oben erwähnte Stelle in der modifizierten Version. Auf diese Weise behandelt Genetic-2n unausgeglichene Transportprobleme genauso wie ausgeglichene Transportprobleme.

4.2.2 Suchraum

Bei der Verwendung einer $(n \times k)$ Matrix, deren Felder reelle Zahlen innerhalb eines durch die Constraints eingeschränkten Bereichs sind, gibt es theoretisch unendlich viele Lösungen. Ob alle Lösungen erreicht werden können hängt von den verwendeten Operatoren ab. Die Initialisierungsroutine(unveränderte Version) und den Mutations-Operator ‚mutation-1‘ können lange nicht aller Lösungen erzeugt werden, da bei der Aufteilung auf die einzelnen Felder immer in möglichst großen Blöcken gearbeitet wird, deren Aufteilung maßgeblich von den Constraints abhängt. Erst durch die Verwendung des Crossover-Operators und der modifizierten Initialisierungsroutine des Mutations-Operators ‚mutation-2‘ können alle Lösungen erzeugt werden.

4.2.3 Lokale Optima

Durch die Mutations-Operatoren ist der Algorithmus in der Lage aus lokalen Optima wieder heraus zu springen. Da die Mutations-Operatoren nicht wie bei klassischen genetischen Algorithmen nur minimale Änderungen an den Lösungen vornehmen, sondern die ganze Lösung komplett neu durch die Initialisierung erzeugen können, ist die Gefahr, das der Algorithmus in lokalen Optima hängen bleibt sehr gering. Beziehungsweise die Chance gegeben, das er aus einem lokalen Optimum, durch die Erzeugung eine „Zufallslösung“ durch die Mutation, wieder herauskommen kann.

4. Test & Vergleich von Genetic-2n

Um den Algorithmus, Genetic-2n, zu testen, wurde er mit einem Standart-Optimierungs-Tool verglichen. Dazu wurden 6 Kostenfunktion ausgewählt, die von Genetic-2n und dem Optimierungs-Tool GAMS bearbeitet werden sollen.
Vorweg sei angemerkt, das der Vergleich nur dem Zweck dient, zu demonstrieren, daß Genetic-2n in der Lage ist eine ‚optimale‘ Lösung oder eine Lösung nahe dem Optimum zu finden. Verglichen wird deshalb die Qualität der gefundenen Lösungen. Betrachtet man die Anzahl der untersuchten Lösungen, sind nicht lineare Verfahren wie z.B. GAMS schneller alles evolutionäre Verfahren, da diese nur ein paar bestimmte Lösungen innerhalb eines lokalen Optimums untersuchen, in dem sie meisten hängen bleiben.

4.1 Standart Optimierungs-Tool GAMS

GAMS (General Algebraic Modeling System) ist ein System zur Konstruktion und Lösung von mathematischen Programmodellen. Es wurde für diesen Test herangezogen, da es ein typisches Beispiel eines industriellen Standartlösungsverfahren darstellt. Das System basiert im wesentlichen auf einer gradienten-gesteuerten Methode. Als Anmerkung sei erwähnt, das GAMS einige der gestellten Probleme als zu schwierig oder gar für unlösbar hielt. In diesen Fällen wurden Modifikationen an den Kostenfunktionen vorgenommen, so daß GAMS wenigstens eine Näherungslösung finden konnte. In das System können zur Berechnung der Zielfunktion spezielle ‚Solvers‘ eingebracht werden. Für die durchgeführten Test wurde der ‚Solver‘ MINOS verwendet.
Mit den durchgeführten Test wurden die Limits der Studentenversion von GAMS erreicht, bei der die Problemgröße beschränkt ist, wogegen die Vollversion nur durch den verfügbaren Hauptspeicher und interner Limits beschränkt ist.

4.2 Die Kostenfunktionen

Für Testzwecke wurden, potentielle Zielfunktionen in dem Artikel eigenmächtig in 3 Gruppen eingeteilt. Die erste Gruppe ‚practical‘ enthält Funktionen, die denkbar für praktische Operation-Research Probleme wären. In der zweiten Gruppe ‚reasonable‘ befinden sich Funktionen, die hauptsächlich als Beispiele in Büchern über Optimierung vorkommen. Als dritte Gruppe ‚other‘ wurden Funktionen gewählt, die häufig als schwere Testfälle für Optimierungstechniken verwendet werden.

Practical:

Funktion A

$$A(x)=\begin{cases} 0 & \text{falls } 0 < x \le S \\ c_{ij} & \text{falls } S < x \le 2S \\ 2\mathrm{c}_{\mathrm{ij}} & \text{falls } 2S < x \le 3S \\ 3\mathrm{c}_{\mathrm{ij}} & \text{falls } 3S < x \le 4S \\ 4\mathrm{c}_{\mathrm{ij}} & \text{falls } 4S < x \le 5S \\ 5\mathrm{c}_{\mathrm{ij}} & \text{falls } 5S < x \end{cases}$$

Wobei S kleiner als ein typischer x Wert ist, der nicht 0 ist.

Funktion B

$$B(x)=\begin{cases} c_{ij}\dfrac{x}{s} & \text{falls } 0 \le x \le S \\ c_{ij} & \text{falls } S < x \le 2S \\ c_{ij}\left(1+\dfrac{x-2S}{S}\right) & \text{falls } 2S < x \end{cases}$$

Wobei S ein typische x Wert ist, der nicht 0 ist.

Reasonable:

Funktion C

$$C(x)=c_{ij}x^2$$

Funktion D

$$D(x)=c_{ij}\sqrt{x}$$

Other:

Funktion E

$$E(x)=c_{ij}\left(\frac{1}{1+(x-2S)^2}+\frac{1}{1+\left(x-\frac{9}{4}S\right)^2}+\frac{1}{1+\left(x-\frac{7}{4}S\right)^2}\right)$$

Wobei S ein typische x Wert ist, der nicht 0 ist.

Funktion F

$$F(x) = c_{ij} x \left(\sin\left(x \frac{5\pi}{4S} \right) + 1 \right)$$

Wobei S ein typische x Wert ist, der nicht 0 ist.

Die Zielfunktion war von der Form

$$\sum_{ij} f(x_{ij})$$

wobei $f(x)$ eine der oben genannten Funktionen ist, die Parameter c_{ij} stammen aus der Kostenmatrix und S wurde aus den Attributen des zu testenden Problems bestimmt.

4.3 Der Test

Im Folgenden werden die Testbedingungen beschrieben, mit denen die Tests durchgeführt wurden. Zum Vergleich der beiden Systeme wurde nicht die benötigte Zeit, sondern das im Rahmen der vorgegeben Testsets gefundene beste Optimum herangezogen. Da die Optima im nicht linearen Fall nicht bekannt sind, wurden die jeweils besten gefundenen Optima der beiden Systeme miteinander verglichen. Das System Genetic-2n wurde auf einer Sun SPARCStation 1 ausgeführt wogegen GAMS auf einem Olivetti 386. Obwohl der Geschwindigkeitsvergleich der beiden Computer schwierig ist, ist zu erwähnen, daß GAMS jeden Durchlauf deutlich schneller beendete als das evolutionäre System. Eine Ausnahme bildet die Funktion A, bei der GAMS zur Berechnung der Kostenfunktion zahlreiche Arctang-Funktionen durchführen muß.

Testbedingungen:

Für jede der unter Kapitel 4.2 genannten Funktionen wurden 5 (10×10) Kostenmatrizen, aus einer Menge von unabhängig gleichmäßig verteilten c_{ij}, erstellt. Für die Constraints wurden zufällige *source* und *dest* Arrays verwendet, mit einem totalen Transportfluß von 100 Einheiten. Für jede Kombination aus Kostenfunktion und Kostenmatrix wurden 5 Durchläufe, mit unterschiedlichen Startwerten für den Zufallszahlengenerator, über 10.000 Generation mit dem GA durchgeführt.

Parameter:

Allgemein

S (Konstante für die Kostenfunktionen)

Für die Funktionen A : S = 2
Für die Funktionen B, E und F : S = 5

Für Genetic-2n

Populationsgröße = 40

Mutation
Mutationsrate: $p_m = 20\%$
Proportion für Mutation: $m_1 = 50\%$

Crossover
Crossoverrate: $p_c = 5\%$
Konstanten: $c_1 = 0.35$ und $c_2 = 0.65$

Im Vergleich zu klassischen evolutionären Algorithmen erscheint die Mutationsrate viel zu hoch und die Crossoverrate zu niedrig. Dies beruht darauf, daß sich die Operatoren von Genetic-2n von denen klassischen evolutionären Algorithmen unterscheiden. Die Mutations-Operatoren verändern nicht nur an einzelnen Stellen , sondern ein großer Teil der Struktur wird einer Mutation unterworfen. Auf diese Weise können durch die Mutation komplett neue Zufallslösungen erzeugt werden. Dies Zusammen mit der hohen Mutationsrate erklärt warum die Ausgangslösungen nicht von großer Bedeutung sind. Im Gegensatz zu klassischen evolutionären Algorithmen, bei den durch Rekombination der Ausgangslösungen und kleinen Veränderungen durch Mutation versucht wird eine optimale Lösung zu finden.

4.4 Ergebnisse

Die folgende Tabelle zeigt den Vergleich der gefundenen Optima zwischen GAMS und Genetic-2n, für ein einzelnes Problem, wobei bei Genetic-2n der Durchschnitt von 5 Durchläufen mit unterschiedlichen Startwerten für den Zufallszahlengenerator angegeben ist.

Funktion	GAMS	Genetic-2n	% Unterschied
A	281.0	202.0	-28.1
B	180.8	163.0	-9.8
C	4402.0	45556.2	+3.5
D	408.4	391.1	-4.2
E	145.1	79.2	-45.4
F	1200.8	201.9	-83.2

Für die Klasse der ‚practical‘ Funktionen ist Genetic-2n im Durchschnitt deutlich besser als GAMS. Bei der Klasse der ‚reasonable‘ Funktionen sieht das Ergebnis anders aus. Für die quadratische Funktion schließt Genetic-2n etwas schlechter ab als GAMS, wogegen für die Quadratwurzelfunktion Genetic-2n wieder leicht besser abschneidet. Bei der Klasse der ‚other‘ Funktionen dominiert Genetic-2n deutlich über GAMS.
Als Durchschnitt über die 5 Probleme ergaben sich folgende Ergebnisse, die angeben, wie Genetic-2n im Vergleich gegenüber GAMS abgeschnitten hat:

Funktion	% Unterschied
A	-24.5
B	-11.5
C	+7.5
D	+2.0
E	-33.0
F	-54.5

5. Ausblicke

Im Bezug auf zahlreiche Transportproblem erscheint die Verwendung von reellen Werte in den Lösungen als nicht allzu brauchbar. Da bei vielen Problem eine zusätzliche Bedingung ist, daß die Lösungen ganzzahlig sein müssen. Denn z.B. beim Transport von Autos aus den Werkstätten zu den Autohäusern sind Werte Ergebnisse wie z.B. Autohaus A bekommt 0.7 Autos von Werkstatt X und 0.3 Autos von Werkstatt Y, nicht unbedingt sinnvoll. Dies gilt für alle Waren oder Güter, die nicht wie z.B. Flüssigkeiten in beliebig kleiner Mengen aufteilen lassen. Eine entsprechende Modifikation des Crossover-Operators dahingehend das die erzeugten Kinder nur aus ganzen Zahlen bestehen oder die Übernahme des Crossover-Operators aus dem Algorithmus Genetic-2 für den linearen Fall, der diese Eigenschaft besitzt, wären zu überlegen. Eine nachträglich Anpassung der Lösung, z.B. durch Runden, hängt von der Kostenfunktion ab und ist nicht immer möglich, da dadurch die optimale Eigenschaft der Lösung zerstört werden kann.

Da Genetic-2n ein speziell an das Transportproblem angepaßter evolutionärer Algorithmus ist, wäre zu erwägen, die Kostenfunktion mit in die Operatoren einzubeziehen. So könnte z.B. bei der Initialisierung die Auswahl der Felder nicht rein zufällig, sondern proportional zu den Kosten stattfinden, auf diese Weise würde versucht, zuerst möglichst viel auf den billigsten Transportrouten zu transportieren. Ähnliches wäre auch für die Mutations-Operatoren denkbar.

Wie auch bereits im Artikel erwähnt ist es möglich den Algorithmus so zu modifizieren, daß ähnlich Operation-Research Probleme wie das Zuordnugsproblem(allocation) und einige ‚scheduling‘ Probleme bearbeitet werden können.

6. Anhang

6.1 Referenzen

- Michalewicz Z 1997 „G9.8 The transportaion problem“ aus „Handbook of Evolutionary Computation release 97/1“

- Michalewicz Z 1996 „Genetic Algorithms + Data Structures = Evolution Programs“ 3rd edn (New York: Springer)