HTWK Leipzig

Entwicklung eines evolutionären Algorithmus zur Bearbeitung des eindimensionalen Behälterproblems

Projektarbeit Evolutionäre Algorithmen SS 2012
Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fakultät für Informatik, Mathematik und Naturwissenschaften

Autor: *Uwe Sommerlatt* <usommerl@imn.htwk-leipzig.de>

Datum: 20. Juni 2012

1 Charakterisierung des Problems

Das eindimensionale Behälterproblem (*one-dimensional bin packing problem*) ist ein NP-schweres, kombinatorisches Optimierungsproblem, bei dem eine geeignete Verteilung von $n \in N$ Objekten auf $k \in N$ Behälter gesucht wird. Die Behälter haben ein einheitliches Fassungsvermögen $c \in N$ und die einzelnen Objekte unterschiedliche, nichtnegative Größen $a_1, \ldots, a_n \leq c$. Das Ziel der Optimierung ist die Minimierung der Anzahl benötigter Behälter k für eine fest definierte Zusammenstellung von n Objekten. Formal kann das Problem wie folgt beschrieben werden.

$$\exists f : \{1, \ldots, n\} \to \{1, \ldots, k\} \mid \forall j \in \{1, \ldots, k\} \sum_{i:f(i)=j} a_i \leq c, \text{ die } k \text{ minimiert.}$$

Da bei diesem Problem nur eine Dimension der zu verpackenden Objekte berücksichtigt wird, kann es auch als eindimensionales Zuschnittproblem (*one-dimensional cutting stock problem*) betrachtet werden. [KV08, S. 485]

2 Beschreibung des entwickelten Algorithmus

Das entwickelte Programm zur Bearbeitung des eindimensionalen Behälterproblems wurde so gestaltet, dass sich verschiedene Teilaspekte des evolutionären Algorithmus über Parameter verändern lassen. Dadurch wird eine gezielte Untersuchung der Effektivität von unterschiedlichen Konfigurationen ermöglicht. In diesem Abschnitt sollen zunächst die einzelnen Teile des Algorithmus, wie z.B. der Aufbau des Individuums oder verschiedene Dekodierungsfunktionen, erläutert werden. Das Zusammenwirken der unterschiedlichen Komponenten im evolutionären Zyklus wird ganz am Ende des Abschnitts erörtert. Der Quellcode der gesamten Anwendung kann über ein öffentliches Repository[1] abgerufen werden.

[1] `https://bitbucket.org/usommerl/bpevolution`

2.1 Genotyp und Phänotyp

Um Methoden der evolutionären Algorithmik auf das in Abschnitt 1 beschriebene Problem anwenden zu können, muss zunächst eine geeignete Repräsentation für alle Lösungskandidaten des Problems modelliert werden. Diese auch als Individuum bezeichnete Repräsentation besteht aus dem Genotyp und dem Phänotyp. Der Genotyp ist das Erbgut des Individuums und stellt seinen spezifischen Satz von Genen dar. In der hier vorgestellten Implementierung ist ein Gen genau ein Objekt aus einer Instanz des Behälterproblems. Dementsprechend ist der Genotyp eine Permutation aller Objekte in der Probleminstanz. Der Phänotyp ist das Erscheinungsbild bzw. die Menge aller äußeren Merkmale des Individuums und wird im Wesentlichen durch den Genotyp bestimmt. In der konkreten Umsetzung ist der Phänotyp daher eine eindeutige Zuordnung der Objekte im Genotyp zu einem Behälter. Die Translation des Genotyps in den zugehörigen Phänotyp wird durch eine Decodierungsfunktion definiert.

2.2 Decodierungsfunktionen

Bei der Verwirklichung des Algorithmus wurde mit drei verschiedenen Decodierungsfunktionen experimentiert. Häufig werden einfache Heuristiken für das Behälterproblem in einen evolutionären Algorithmus eingebettet, um die Suche nach einer möglichst optimalen Lösung zu verbessern [GC99, S. 64]. Dieser Ansatz wurde auch bei der Realisierung der Decodierungsfunktion gewählt, indem die Greedy-Heuristiken *Next-Fit*, *First-Fit* und *Best-Fit* als Decodierungsfunktionen verwendet wurden. Die Verarbeitung der einzelnen Objekte des Genotyps durch die Decodierungsfunktion erfolgt in der Reihenfolge ihrer Indizes. Anhand der im Anschluss vorgestellten Verfahrensweisen platziert die jeweilige Funktion das bearbeitete Objekt in einen Behälter.

- Die ***Next-Fit***-Heuristik weist ein Objekt dem aktuellen Behälter zu, insofern es in diesen passt. Falls kein aktueller Behälter existiert oder das Objekt zu groß für den aktuellen Behälter ist, wird ein neuer Behälter erzeugt. Das Objekt wird anschließend dem neu erzeugten Behälter zugewiesen und dieser wird zum aktuellen Behälter.

- Bei der ***First-Fit***-Heuristik wird ein Objekt dem ersten bereits vorhandenen Behälter zugewiesen, der ausreichend Kapazität für das Objekt aufweist. Falls das Objekt in keinen vorhandenen Behälter passt, wird einer neuer Behälter erzeugt und es darin platziert.

- Die ***Best-Fit***-Heuristik weist das Objekt dem Behälter zu, der über ausreichend Kapazität für das Objekt verfügt und gleichzeitig das geringste verbleibende Fassungsvermögen aller Behälter, die das Objekt aufnehmen können, besitzt. Auch bei dieser Vorgehensweise wird ein neuer Behälter erzeugt, insofern kein vorhandener Behälter das Objekt aufnehmen kann.

2.3 Bewertungsfunktionen

Die eigentliche Optimierung durch den evolutionären Algorithmus erfolgt auf Basis der Güte der einzelnen Lösungskandidaten. Die Güte eines Individuums gibt an wie gut bzw. wie schlecht das Individuum das jeweilige Problem löst. Entscheidend ist der durch eine Bewertungsfunktion ermittelte Gütewert vor allem für die Selektionsoperatoren des Algorithmus. Während der praktischen Untersuchungen wurde mit zwei verschiedenen Bewertungsfunktionen q_1 und q_2 gearbeitet. Je kleiner der Funktionswert dieser Bewertungsfunktionen ist, desto besser ist die durch das Individuum repräsentierte Lösung. Die relativ einfache Bewertungsfunktion q_1 ermittelt lediglich die Anzahl der benötigten Behälter k für ein beliebiges Individuum x.

$$q_1(x) = x.k$$

Die Bewertungsfunktion q_2 berücksichtigt neben der Anzahl der benötigten Behälter auch das von jedem Behälter b_i nicht genutzte Fassungsvermögen r. Dabei bezeichnet c das Gesamtfassungsvermögen eines Behälters und n die Anzahl der Objekte in der Probleminstanz. Durch die Funktion q_2 werden die Individuen der Population besser bewertet, die das Fassungsvermögen der von ihnen genutzten Behälter möglichst effizient ausnutzen.

$$q_2(x) = x.k + \frac{\sum_{i=1}^{k} x.b_i.r^2}{c^2 * n}$$

2.4 Operatoren

Die Operatoren für Selektion, Rekombination und Mutation beeinflussen maßgeblich die Evolution einer Population von Individuen. Für jeden dieser Evolutionsfaktoren wurden mehrere Algorithmen implementiert. Da es sich dabei um Standardalgorithmen handelt, sollen diese hier nur genannt und nicht näher erläutert werden.

Selektion: Die realisierten Selektionsalgorithmen können sowohl als Eltern- als auch als Umweltselektion eingesetzt werden. Implementiert wurden die *Besten-Selektion* [Wei07, S. 65], die *Q-Stufige-Turnier-Selektion* [Wei07, S. 69] und ein einfacher Algorithmus, der eine probabilistische Indexselektion realisiert.

Rekombination: Die Rekombinationsalgorithmen müssen infolge der Beschaffenheit des Genotyps für Permutationen geeignet sein, da ansonsten nicht gewährleistet ist, dass ein gültiges Individuum entsteht. Aus diesem Grund wurden die *Abbildungsrekombination* [Wei07, S. 133] und die *Ordnungsrekombination* [Wei07, S. 29] implementiert.

Mutation: Der evolutionäre Algorithmus wurde so angelegt, dass der Einsatz des Mutationsoperators optional ist. Es ist jedoch auch möglich, mehrere Mutationen

hintereinander auszuführen. Die für die Mutation verwendeten Algorithmen müssen ebenfalls für Permutationen geeignet sein. Demzufolge wurde die *Vertauschende-Mutation* [Wei07, S. 27], die *Invertierende-Mutation* [Wei07, S. 28] und die *Verschiebende-Mutation* [Wei07, S. 132] implementiert.

2.5 Diversitätsmaß

Die Diversität ist ein Maß für die Vielfalt der Genotypen in einer Population. Zur Ermittlung dieses Maßes wird in der konkreten Umsetzung ein Hashwert von jedem Genotyp in der Population erzeugt. Der Hashwert von zwei Genotypen ist gleich, insofern diese die gleiche Permutation von Objekten verkörpern. In der praktischen Umsetzung wird die Diversität berechnet, indem die Anzahl der einzigartigen Hashwerte in der Population durch die Anzahl der Individuen dividiert und das Ergebnis anschließenden mit 100 multipliziert wird. Folglich drückt ein Diversitätswert von 100 aus, dass jeder Genotyp in der Population einzigartig ist. Umso stärker der errechnete Wert unterhalb von 100 liegt, desto mehr Duplikate eines odere mehrerer Genotypen existieren in der Population. Aufgrund von Hash-Kollisionen kann es in seltenen Fällen vorkommen, dass die tatsächliche Diversität größer ist, als durch das Verfahren berechnet wurde. In Ermangelung von alternativen Ansätzen wird dieser Fehler toleriert.

2.6 Zyklus des evolutionären Algorithmus

In Listing 2.1 ist der Ablauf des gesamten evolutionären Algorithmus als Pseudocode dargestellt. Neben den dargestellten Parametern muss zusätzlich spezifiziert werden, welche Decodierungsfunktion alle Individuen verwenden sollen. Die Initialisierung der Population erfolgt, indem die Liste der einzusortierenden Objekte für jedes Individuum zufällig permutiert wird. Das Individuum bildet mittels der Decodierungsfunktion aus dem so entstandenen Genotyp den Phänotyp und kann

anschließend mit der übergebenen Gütefunktion bewertet werden. Der Zyklus des Algorithmus terminiert, sobald ein Individuum gefunden wurde, das so viele Behälter verwendet wie theoretisch für die bearbeitete Probleminstanz minimal ist. Eine Diskussion zu der theoretisch minimal benötigten Anzahl an Behältern für eine Probleminstanz befindet sich in Abschnitt 3. Als zusätzliche Terminierungsbedingung kann eine maximale Anzahl an Generationen festgelegt werden, die vom Algorithmus berechnet werden.

```
1  EA-BIN-PACKING( Bewertungsfunktion ξ, Recombination δ,
2                  Elternselektion α_1, Umweltselektion α_2,
3                  Populationsgröße λ, Mutationen ϒ )
4  t ← 0
5  P(t) ← initialisiere Population mit λ Individuen
6  bewerte alle Individuen aus P(t) mit Bewertungsfunktion ξ
7  while Terminierungsbedingung nicht erfüllt
8  do ⌜P′ ← selektiere λ/3 Eltern aus P(t) mit α_1
9      P″ ← erzeuge Nachkommen durch Rekombination δ aus P′
10     for υ ∈ ϒ
11     do ⊏ P″ ← mutiere die Individuen in P″ mit υ
12     bewerte alle Individuen aus P″ mit Bewertungsfunktion ξ
13     t ← t + 1
14    ⌟P(t) ← selektiere λ Individuen aus P″ ∘ P(t - 1) mit α_2
15 return bestes Individuum aus P(t)
```

Listing 2.1: Evolutionärer Bin-Packing-Algorithmus

Bei jedem Evolutionsschritt wird zunächst $\frac{1}{3}$ der Population mittels des spezifizierten Selektionsalgorithmus als Eltern ausgewählt. Aus den Eltern werden mithilfe des Rekombinationsalgorithmus Nachkommen erzeugt, wobei auf den Einsatz von Rekombinationswahrscheinlichkeiten verzichtet wurde. Für die Rekombination werden die Eltern in zufälliger Reihenfolge in eine zyklische Liste einsortiert. Anschließend vermehrt sich jeder Elter je einmal mit seinen direkten Nachbarn in der Liste. Insofern ein oder mehrere Mutationsalgorithmen spezifiziert wurden, werden diese in der angegebenen Reihenfolge auf jedes neu erzeugte Individuum angewendet. Am Ende des Evolutionsschritts wird eine Umweltselektion auf die neu erzeugten

und sämtliche alten Individuen angewendet. Dadurch wird gewährleistet, dass die Populationsgröße stets konstant bleibt.

3 Probleminstanzen

Für die praktische Analyse des entwickelten Algorithmus werden vorhandene Datensätze aus der *Operations Research* (OR) Bibliothek[2] von J. E. Beasley verwendet. Ursprünglich wurden diese Datensätze zur Untersuchung des *Hybrid Grouping Genetic Algorithm* (HGGA) von Emanuel Falkenauer konstruiert [Fal96]. Die OR-Bibliothek stellt insgesamt acht Dateien mit 160 Instanzen des eindimensionalen Behälterproblems zur Verfügung. Diese wurden direkt in die Ressourcen der entwickelten Anwendung integriert, sodass alle Probleminstanzen komfortabel bearbeitet werden können. Bei den vorhandenen Datensätzen wird zwischen Problemen der Klasse *Uniform* und der Klasse *Triplets* unterschieden. Probleme der *Uniform*-Klasse bestehen aus Gegenständen mit einer gleichmäßig verteilten Größe zwischen 20 und 100 Einheiten. Diese müssen in Behälter mit einem Fassungsvermögen von 150 Einheiten einsortiert werden. Die Probleme der *Triplets*-Klasse enthalten Gegenstände mit einer Größe von 25 bis 50 Einheiten und die zu bestückenden Behälter besitzen ein Fassungsvermögen von 100 Einheiten. Aufgrund der Konstruktion der *Triplets*-Probleminstanzen, wird für einen gut gefüllten Behälter ein großer Gegenstand und zwei kleinere benötigt. Dabei muss der große Gegenstand mehr als die Hälfte des Fassungsvermögens des Behälters einnehmen. Die Intention bei der Entwicklung der *Triplets*-Klasse war die Erforschung der praktischen Grenzen von Falkenauers HGGA [Fal96, S. 17]. Aus diesem Grund sind Probleminstanzen der *Triplets*-Klasse als komplexer einzustufen als Instanzen der *Uniform*-Klasse.

Alle Probleminstanzen werden durch einen eindeutigen Identifikator gekennzeichnet. Probleme deren Identifikator mit einem 'u' beginnt gehören zur *Uniform*-Klasse. Identifikatoren, die mit einem 't' beginnen, bezeichnen dagegen Probleme

[2] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

der *Triplets*-Klasse. Nach dem Buchstaben für die Klassenzugehörigkeit folgt die Anzahl der Gegenstände in der Probleminstanz und die Nummer des Problems. Für die *Uniform*-Klasse existieren jeweils zwanzig Probleme mit 120, 250, 500 und 1000 Gegenständen. Die Nummerierung beginnt für jede Problemgröße von neuem bei 0. Valide Identifikatoren für Probleme der *Uniform*-Klasse sind somit beispielsweise `u120_00`, `u250_00` oder `u500_19`. Analog dazu existieren jeweils zwanzig Probleme mit 60, 120, 249 und 501 Gegenständen für die *Triplets*-Klasse. Der Aufbau der Identifikatoren ist mit denen der *Uniform*-Klasse identisch.

Die theoretisch minimal benötigte Anzahl an Behältern für eine Probleminstanz kann wie folgt dargestellt werden.

$$k_{min} = \left\lceil \frac{\sum_{i=1}^{n} a_i}{b} \right\rceil$$

Im Verlauf der Untersuchungen von E. Falkenauer mit dem HGGA konnte diese theoretisch optimale Lösung für alle Instanzen außer `u120_08`, `u120_19`, `u250_07`, `u250_12`, `u250_13`, `t60_07` und `t60_18` auch praktisch ermittelt werden. Für die eben genannten Instanzen wurde dagegen eine Lösung gefunden, die jeweils einen Behälter mehr benötigt als die theoretische minimale Lösung. [Fal96, S. 19-22]

4 Untersuchung des entwickelten Algorithmus

Das primäre Ziel bei der Untersuchung des Algorithmus war die Ermittlung einer sinnvollen Konfiguration zur effektiven Bearbeitung der in Abschnitt 3 beschriebenen Probleminstanzen. Zu diesem Zweck wurde die in Listing 4.1 abgebildete Standardkonfiguration festgelegt. Sämtliche in den Abschnitten 4.1 bis 4.4 beschriebenen Experimente nutzen diese Konfiguration und variieren ausschließlich die Einstellung des jeweils untersuchten Parameters.

```
1  Populationsgröße      ← 500
2  Decodierungsfunktion  ← Best-Fit
3  Bewertungsfunktion    ← q_1
4  Elternselektion       ← Besten-Selektion
5  Rekombination         ← Abbildungsrekombination
6  Mutation              ← Verschiebende-Mutation
7  Umweltselektion       ← Q-Stufige-Turnier-Selektion (q = 3)
```

Listing 4.1: Standardkonfiguration für die Evaluierungsexperimente

4.1 Vergleich der Decodierungsfunktionen

Zur Feststellung der effektivsten Decodierungsfunktion wurde willkürlich eine Probleminstanz ausgewählt (`u120_00`) und anschließend mehrfach mit dem Algorithmus bearbeitet. Insgesamt wurden 11 Testläufe je Decodierungsfunktionen durchgeführt. Im Anschluss wurde für jede Funktion der Datensatz ausgewählt, bei dem die Ausführungszeit des Testlaufs dem Median aller 11 Laufzeiten entsprach. In Abbildung 4.1 wurden diese drei Datensätze exemplarisch gegenübergestellt. Das Diagramm verdeutlicht, wie stark der Decodierungsalgorithmus den Phänotyp des Individuums und letztendlich die Effizienz des gesamten Algorithmus beeinflusst. Die evolutionäre Suche unter Verwendung der *Best-Fit*-Heuristik nähert sich in der Gegenüberstellung deutlich schneller der unteren Grenze der benötigten Behälter (k_{min}) an. Dagegen führt die Suche mit der *Next-Fit*- und *First-Fit*-Heuristik in den Tests erst sehr viel später oder häufig auch überhaupt nicht zu einer minimalen Lösung.

4.2 Vergleich der Mutationsoperatoren

Um die Mutationsalgorithmen zu vergleichen, wurde die Probleminstanz `u120_02` unter Verwendung der implementierten Algorithmen jeweils 11-mal bearbeitet. Im

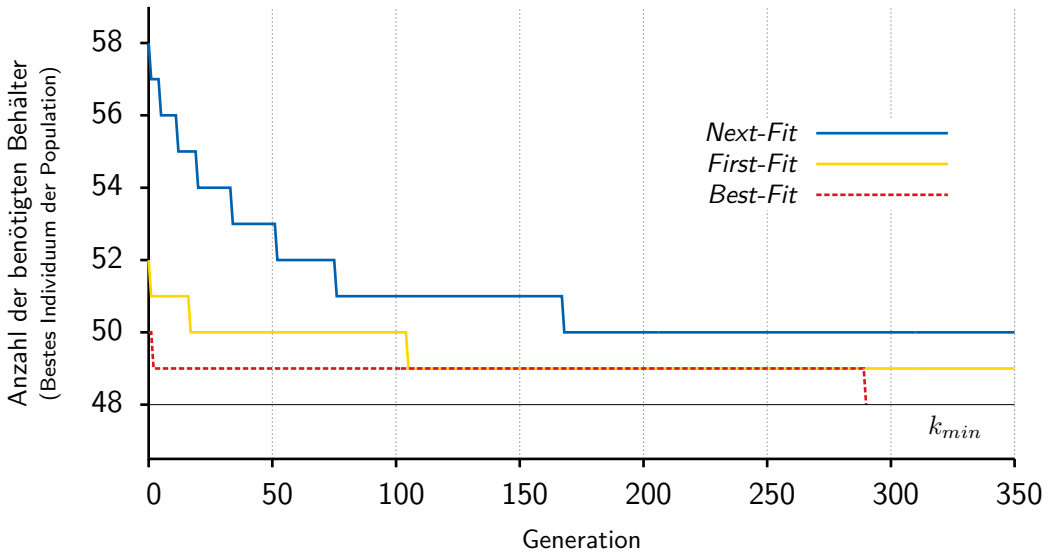

Abbildung 4.1: Vergleich der *Next-Fit*-, *First-Fit*- und *Best-Fit*-Decodierungsfunktion

Anschluss wurde bei allen Testläufen ermittelt, in welcher Generation ein Individuum gefunden wurde, das einer theoretisch minimalen Lösung entspricht. Von diesen Werten wurde für jeden Algorithmus der Median gebildet und zum Vergleich der Mutationsverfahren verwendet. Konnte in einem Test keine minimale Lösung gefunden werden, wurde der Wert des Testlaufs auf ∞ gesetzt. Der Vergleich auf Basis der Generation wurde gewählt, da dabei die unterschiedlichen Laufzeiten der Mutationsalgorithmen keinen Einfluss haben.

Abbildung 4.2 zeigt das Resultat dieses Vergleichs. Das Verhältnis über den Balken des Diagramms gibt an, wie häufig die Suche in der entsprechenden Konfiguration mit einer theoretisch minimalen Lösung beendet wurde. Interessant am Endergebnis des Experiments ist, dass die Suche im Mittel am schnellsten zu einer optimalen Lösung führte, wenn überhaupt kein Mutationsoperator eingesetzt wurde. Des Weiteren erwies sich die Invertierende-Mutation als am schlechtesten geeignet für die bearbeitete Problemstellung. Über die gesamte Anzahl an Testläufen konnte beim Einsatz der Verschiebenden-Mutation oder der Vertauschenden-Mutation immer

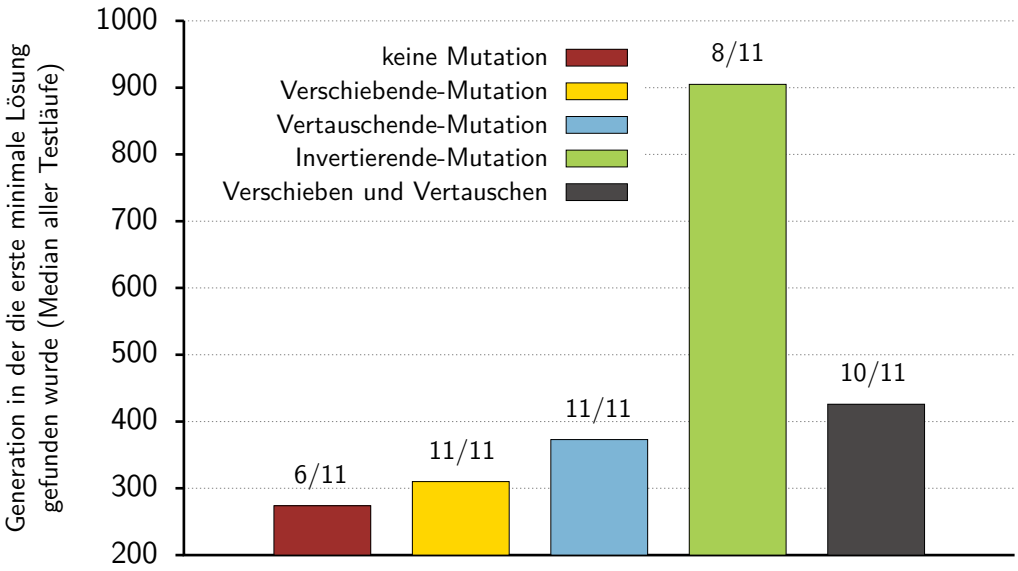

Abbildung 4.2: Untersuchung verschiedener Mutationsalgorithmen

eine minimalen Lösung ermittelt werden. Ein Hintereinanderausführen beider Mutationen konnte die Suche jedoch nicht beschleunigen und führte nur in 10 von 11 Testläufen zu einer optimalen Lösung.

Wurde die Suche ohne Mutationsalgorithmus und ausschließlich mit Hilfe des Rekombinations- und der Selektions-Operatoren durchgeführt, konnte nur in 54% der Fälle eine minimale Lösung gefunden werden. Eine Ursache dafür ist wahrscheinlich die sinkende Diversität im Verlauf der Evolution. Abbildung 4.3 vergleicht die Entwicklung der Diversität anhand zweier Testläufe. Ein Testlauf wurde ohne Mutationsoperator durchgeführt und ein weiterer mit der Verschiebenden-Mutation. In der Abbildung ist deutlich zu erkennen, wie die Verschiebende-Mutation die Diversität konstant bei einem Wert von 100 hält, während diese ohne Einsatz eines Mutationsoperators immer weiter absinkt.

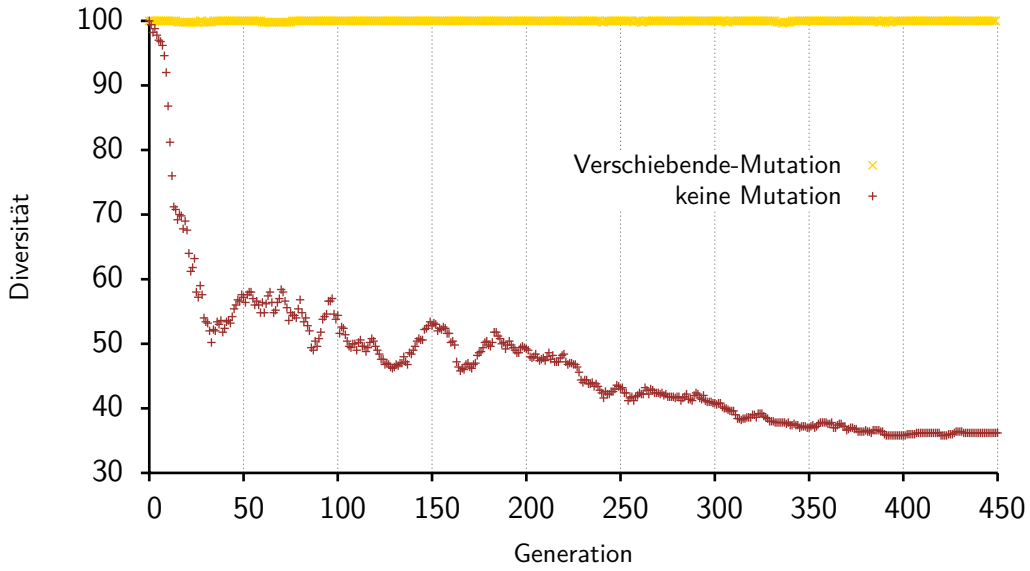

Abbildung 4.3: Entwicklung der Diversität mit und ohne Mutationsoperator

4.3 Vergleich der Rekombinationsalgorithmen

Beim Experiment zur Untersuchung der Rekombinationsverfahren wurde analog zur Untersuchung der Mutationsoperatoren vorgegangen (vgl. Unterabschnitt 4.2). Als Eingabe-Datensatz diente die Probleminstanz `u120_15`. Das Ergebnis des Vergleichs ist in Abbildung 4.4 dargestellt. In der Abbildung wird deutlich, dass die Ermittlung einer minimalen Lösung unter Verwendung der Ordnungsrekombination wesentlich länger dauerte als mit der Abbildungsrekombination und nur in 73% der Testläufe überhaupt erfolgreich war.

Während der Untersuchungen wurde noch ein weiterer Rekombinationsoperator entwickelt, der bei jeder Anwendung zufällig entweder eine Abbildungsrekombination oder eine Ordnungsrekombination durchführt. Dieser Ansatz erwies sich im Mittel am effektivsten, wobei der Unterschied zur Abbildungsrekombination nicht signifikant ist.

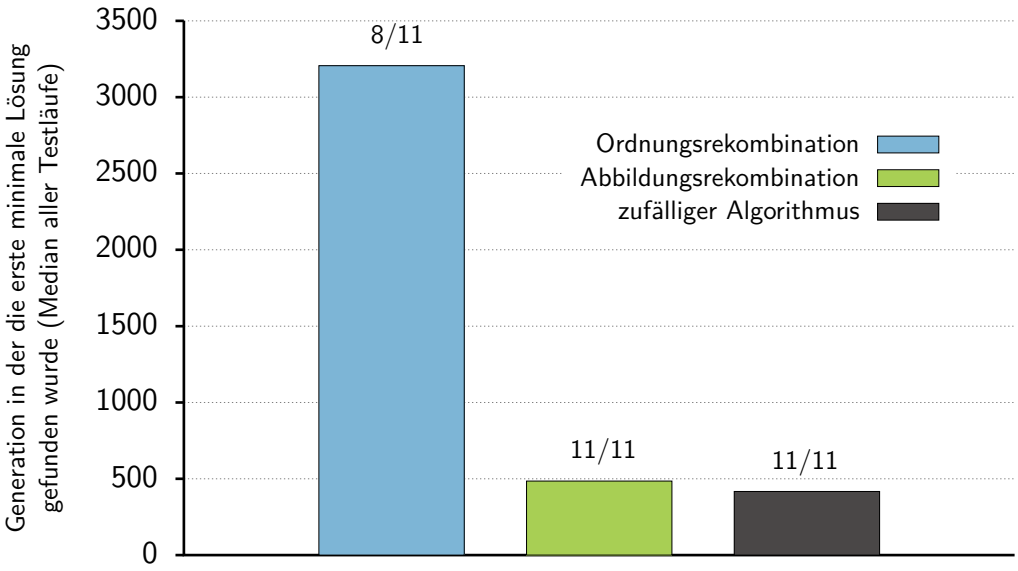

Abbildung 4.4: Gegenüberstellung verschiedener Recombinationsalgorithmen

4.4 Vergleich der Bewertungsfunktionen

Die in Unterabschnitt 2.3 vorgestellten Bewertungsfunktionen q_1 und q_2 wurden anhand von Probleminstanz `u120_13` untersucht. Im Unterschied zu den vorangegangenen Experimenten wurden diesmal jeweils 21 Testläufe pro Bewertungsfunktion durchgeführt. Bis auf die Einstellung der Bewertungsfunktion entsprachen alle Parameter der am Anfang von Abschnitt 4 vorgestellten Standardkonfiguration.

Unter Verwendung der Funktion q_1 konnte in allen 21 Testläufen eine minimale Lösung ermittelt werden. Der Median der benötigten Generation bis zum Auffinden dieser Lösung lag bei 101 Generationen. Im Gegensatz dazu konnte mithilfe der Bewertungsfunktion q_2 in keinem Testlauf eine Behälterbelegung errechnet werden, die dem theoretischen Optimum von 49 Behältern entspricht. Da sich die Population bei der Verwendung von q_2 in Richtung des Optimums entwickelte, kann ein grundsätzlicher Implementierungsfehler ausgeschlossen werden. Am Ende eines

Testlaufs mit der Funktion q_2 lag die Anzahl der benötigten Behälter in der Regel ein Behälter über dem theoretischen Optimum. Es ist daher zu vermuten, dass die Funktion ungeeignet ist, sobald nur noch sehr wenig Platz in allen verwendeten Behältern verfügbar ist.

4.5 Laufzeitvergleich mit dem *Hybrid Genetic Grouping Algorithm*

Zum Abschluss der Untersuchungen trat der entwickelte evolutionäre Bin-Packing-Algorithmus (EBPA) in einem Laufzeitvergleich gegen den *Hybrid Genetic Grouping Algorithm* (HGGA) an. Die für den HGGA erforderlichen Messwerte wurden dem Anhang von [Fal96] entnommen. Aufgrund der Erkenntnisse aus den vorangegangenen Versuchen wurde die Standardkonfiguration des EBPA dahin gehend verändert, dass bei der Rekombination zufällig einer der beiden verfügbaren Algorithmen ausgewählt wird (siehe Unterabschnitt 4.3). Für den Vergleich wurden insgesamt 40 Probleminstanzen ausgewählt und jeweils einmal mit dem EBPA bearbeitet. Die Menge der Probleme bestanden aus 20 Instanzen der *Uniform*-Klasse mit jeweils 120 Objekten und 20 Instanzen der *Triplets*-Klasse mit jeweils 60 Objekten.

Die Ergebnisse für die Probleminstanzen der *Uniform*-Klasse sind in Abbildung 4.5 dargestellt. Die Laufzeiten geben an, wie lange das Berechnen einer minimale Lösung in Anspruch nahm. Gelang dies einem oder beiden Algorithmen bei einer Probleminstanz nicht, wurde das Ergebnis nicht in das Diagramm übernommen. Bezüglich der bearbeiteten Probleme der *Uniform*-Klasse fand der HGGA bei 18 von 20 Instanzen eine minimale Lösung und der EBPA bei 14 von 20. Neben der höheren Zuverlässigkeit hinsichtlich des Auffindens einer optimalen Lösung war der HGGA auch in Bezug auf die Laufzeit bei allen Vergleichen deutlich überlegen. Eine sinnvolle Gegenüberstellung für die Probleme der *Triplets*-Klasse war nicht möglich, da der EBPA bei keiner der Instanzen eine minimale Lösung ermitteln konnte. Dem HGGA gelang dies dagegen bei 18 von 20 Problemen. Bei sämtlichen Tests mit dem EBPA, die nicht zu einer minimalen Lösung führten, benötigte die beste ermittelte Lösung einen Behälter mehr als das theoretische Optimum. Die

Ausführung des Testlaufs wurde bei diesem Experiment automatisch nach jeweils 5000 berechneten Generationen abgebrochen.

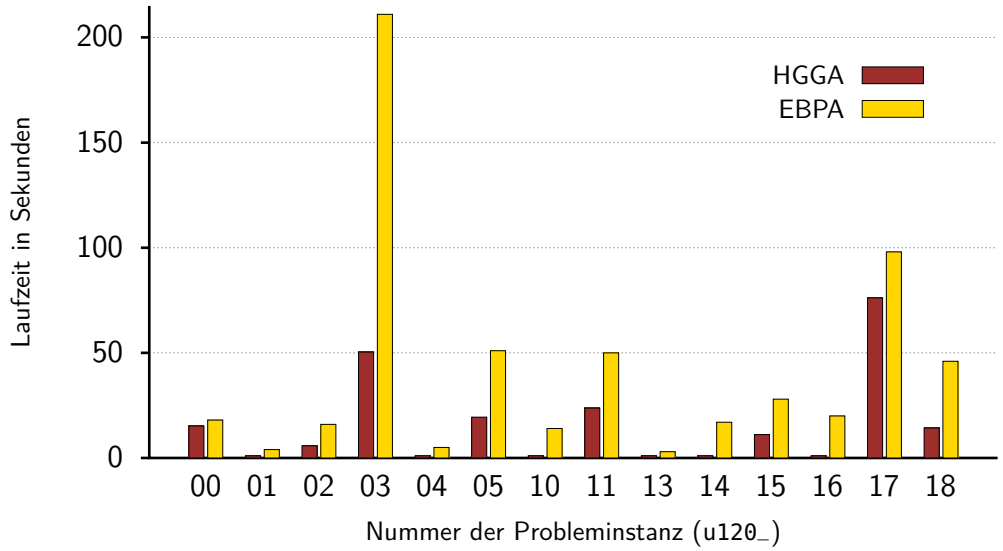

Abbildung 4.5: Laufzeitvergleich anhand von Probleminstanzen der *Uniform*-Klasse

4.6 Fazit

Die Untersuchungen haben gezeigt, dass die Effektivität des entwickelten Algorithmus von einer ganzen Reihe von Faktoren beeinflusst wird. In Bezug auf die untersuchten Decodierungsfunktionen ist der Einsatz der *Best-Fit*-Heuristik unerlässlich, um möglichst schnell eine möglichst minimale Lösung zu ermitteln. Die Verwendung des Mutationsoperators ist dagegen nicht zwingend erforderlich, um eine Lösung mit der theoretisch minimalen Anzahl an Behältern zu bestimmen. Teilweise wird durch das Weglassen der Mutation die Suche sogar beschleunigt. Dennoch ist die Verwendung eines Mutationsalgorithmus sinnvoll, da dadurch wirksam eine frühzeitige Konvergenz der Population verhindert werden kann. Wurde der Mutationsoperator eingesetzt, war die Verschiebende-Mutation im Vergleich

am effektivsten. Eine Steigerung der Leistungsfähigkeit durch mehrere, hintereinander ausgeführte Mutationen konnte nicht beobachtet werden. Als wirksamster Rekombinationsalgorithmus erwies sich die Abbildungsrekombination. Durch die zufällige Verwendung von Ordnungsrekombination und Abbildungsrekombination konnten ebenfalls gute Ergebnisse erzielt werden. Bei den vorgestellten Bewertungsfunktionen q_1 und q_2 erwies sich die Funktion q_2 als weniger geeignet, um eine optimale Lösung zu ermitteln.

Das in Unterabschnitt 4.5 beschriebene Experiment verdeutlicht, dass der entwickelte Algorithmus nicht jedes Problem gleich gut löst. Besonders die untersuchten Instanzen der *Triplets*-Klasse bereiteten dem EBPA Probleme. Im direkten Vergleich mit dem *Hybrid Genetic Grouping Algorithm* war der EBPA deutlich unterlegen. Die Ursache dafür ist in der unterschiedlichen Codierung des Problems durch den jeweiligen Algorithmus zu suchen. Der EBPA verwendet eine objektbasierte Codierung, während der HGGA die Gruppierung mehrerer Objekte codiert. Die objektbasierte Codierung hat den Nachteil, dass verschiedene Permutationen von Objekten die gleiche Zuordnung der Objekte zu den Behältern repräsentieren. Dadurch ist der Suchraum des Algorithmus weitaus größer, als es für die Problemstellung eigentlich notwendig ist.

Literaturverzeichnis

[Fal96] Falkenauer, E.: *A Hybrid Grouping Genetic Algorithm for Bin Packing*. 1996.

[GC99] Gen, M./Cheng, R.: *Genetic Algorithms and Engineering Optimization*. John Wiley & Sons, 1999. ISBN: 978-0471315315.

[KV08] Korte, B./Vygen, J.: *Kombinatorische Optimierung. Theorie und Algorithmen*. Springer-Verlag, 2008. ISBN: 978-3-540-76919-4.

[Wei07] Weicker, K.: *Evolutionäre Algorithmen*. 2. Aufl. Teubner, 2007. ISBN: 978-3-8351-0219-4.