D I P L O M A R B E I T

Verfahren zur Lösung eines Glasverschnittproblems

ausgeführt am

Institut für Computergraphik und Algorithmen

der Technischen Universität Wien

unter Anleitung von

Ass.-Prof. Univ.-Doz. Dr. Günther Raidl

Univ.-Ass. Dr. Gabriele Kodydek

durch

Jakob Puchinger

Paulinengasse 9/14/11
1180 WIEN

Datum

Unterschrift

Zusammenfassung

In der vorliegenden Diplomarbeit werden Verfahren zur Lösung eines Glasverschnittproblems beschrieben. Es handelt sich dabei um ein aus der Industrie kommendes Problem, welches einem um Nebenbedingungen erweiterten zweidimensionalen *Bin Packing* Problem mit Guillotineschnitt entspricht. Die Nebendbedingungen ergeben sich aus den Eigenschaften der Produktionslinie: Am Ende der Glasschneideanlage steht ein Ablegeroboter, welcher die Fertigteile auf drei verschiedene Transportwagen ablegt. Die Fertigteile sind in logischen Gruppen zusammengefasst, und es dürfen nur Fertigteile derselben logischen Gruppe auf einem Transportwagen abgelegt werden. Die Transportwagen können nicht beliebig ausgewechselt werden. All dies führt zu Einschränkungen für die Reihenfolge der auszuschneidenden Elemente.

Nach einer genauen Problemdefinition werden das zweidimensionale *Bin Packing* Problem, sowie verschiedene aus der Literatur bekannte Verfahren zur Lösung desselben vorgestellt. Im Anschluss daran werden zwei *greedy* Heuristiken präsentiert. Die erste basiert auf *Finite First-Fit Decreasing Height* und die zweite auf einer *Best Fit* Strategie. Weiters werden zwei auf *Branch and Bound* (B&B) beruhende Verfahren vorgestellt, wobei das erste iterativ Abschnitte mittels B&B lokal optimal erzeugt und das zweite ganze Rohlinge mittels B&B optimiert. Die letzten drei Verfahren sind Evolutionäre Algorithmen, welche sich durch die Repräsentation der Lösungskandidaten und durch die Dekodierfunktionen unterscheiden. Es werden zwei verschiedene permutationsbasierte Repräsentationen verwendet. Die Dekodierfunktionen erzeugen aus einer im Chromosom kodierten Reihenfolge eine korrekte Lösung für das Glasverschnittproblem. In der Dekodierfunktion der dritten Variante kommt die Abschnittsoptimierung mittels B&B zum Einsatz. Es werden auch verschiedene Rekombinationsoperatoren verwendet. Die verschiedenen Verfahren werden ausführlich experimentell evaluiert und untereinander verglichen. Es stehen überdies Ergebnisse eines kommerziellen Optimierungsprogramms (XOPTS) zur Verfügung, welche mit den erzielten Ergebnissen verglichen werden. Bei diesen Vergleichen muss berücksichtigt werden, dass die Lösungen von XOPTS im Allgemeinen die Nebenbedingungen nicht einhalten und somit potenziell weniger Verschnitt aufweisen können, als dies überhaupt tatsächlich für unser Problem möglich ist. Trotzdem gelingt es in den meisten Fällen, mit den evolutionären Algorithmen bessere Resultate als mit XOPTS zu erzielen.

Abstract

This thesis is about methods developed to solve a glass-cutting problem, a problem, which has its origin in a real-world industrial application. It is a two-dimensional bin packing problem only allowing guillotine cuts and enhanced by some auxiliary constraints which result from the characteristics of the production process. A robot placed at the end of the glass-cutting plant puts the finished elements on three different wagons. The elements are partitioned in logical groups, and each wagon can only carry elements from the same logical group. Since the wagons cannot be arbitrarily exchanged, only elements from a maximum of three logical groups can be cut out until a wagon is filled.

After describing the problem in detail, the classical two-dimensional bin packing problem is presented and methods from the literature used to solve two-dimensional bin packing will be examined. Then, two greedy heuristics will be introduced. The first one is based on the *Finite First-Fit Decreasing Height* algorithm and the second one on the *Best Fit* strategy. After that, two *branch and bound* (B&B) based algorithms are presented. The first one iteratively generates local optimal shelves using B&B which are then placed on the sheets. The second one produces entire sheets using B&B. The last three methods are evolutionary algorithms. The difference between these algorithms are the representation of candidate solutions and the decoding functions. Two different permutation-based representations are used. The decoding functions generate a correct solution from the order given by a chromosome. In the decoding function of the third variant, shelf-optimization through B&B is also used. In addition, different crossover operators are tested. The different algorithms are experimentally evaluated and compared to each other. Furthermore, results obtained by a commercial optimizer (XOPTS) are available. They are compared with the results obtained by our methods. The results produced by XOPTS do not respect the auxiliary conditions and are therefore potentially better. In most cases, however, the evolutionary algorithms provide better results.

Danksagung

An dieser Stelle möchte ich Univ.-Doz. Dr. Günther Raidl und Dr. Gabriele Kodydek für die Betreuung meiner Arbeit sowie die besonders hilfreichen Ratschläge und anregenden Gespräche meinen Dank aussprechen.

Mein Dank gilt auch der Firma Soglatec, welche eine ausführliche Problembeschreibung sowie die realen Instanzen und Ergebnisse des kommerziellen Optimierers XOPTS zur Verfügung gestellt hat.

Ich möchte mich auch bei meiner Familie sehr herzlich bedanken, die mich während des gesamten Studiums nicht nur materiell, sondern vor allem auch moralisch unterstützt hat.

Zu guter Letzt danke ich Miriam Hamidi, die mir beim Korrekturlesen der Diplomarbeit behilflich war und mir den notwendigen persönlichen Rückhalt gegeben hat.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einleitung

In vielen Bereichen der Industrie sowie im Transportwesen spielen Packungs- und Zuschneideprobleme eine wichtige Rolle. Heute werden in vielen Unternehmen Computer eingesetzt, um diese zumeist NP-schwierigen kombinatorischen Optimierungsprobleme zu lösen.

Das Hauptthema dieser Diplomarbeit ist ein spezielles Glasverschnittproblem, welches in diesem Kapitel näher erläutert wird. Im zweiten Kapitel werden grundlegende und in der Literatur bekannte zweidimensionale Packungsprobleme, insbesondere zweidimensionales *Bin Packing* erläutert. Die darauf folgenden Kapitel beschreiben verschiedene Lösungsansätze für das zentrale Problem dieser Arbeit: Kapitel 3 beschreibt einfache *greedy* Verfahren, Kapitel 4 beschreibt *Branch and Bound* Verfahren, und Kapitel 5 beschreibt metaheuristische Verfahren. In Kapitel 6 werden die verschiedenen Verfahren anhand experimenteller Ergebnisse verglichen. Die Anhänge enthalten eine Beschreibung der im Laufe dieser Arbeit entwickelten Anwendung „cutter“, Details zur Implementierung sowie Beschreibungen und Lösungen ausgewählter Instanzen.

1.1 Flachglas: Herstellung und Verarbeitung

Grabner und Roschitz [15] definieren technisches Glas als ein „anorganisches Schmelzprodukt, eine amorphe erstarrte Schmelze, welche durch eine kontrollierte Abkühlung ohne Kristallisation vom flüssigen in den festen Zustand übergeht“. Um Glas herzustellen, müssen zuerst die Ausgangssubstanzen (wie Quarzsand, Soda, Dolomit und Kalkstein) geschmolzen werden, danach wird es mittels verschiedenster Verfahren wie Blasen, Ziehen oder Gießen in Form gebracht.

Petzold, Marusch und Schramm [31] verstehen unter Flachglas „alle Gläser, die in Form flacher Tafeln hergestellt werden, deren Dicke im Verhältnis zu Länge und Breite gering ist und die sowohl glatte, planparallele Oberflächen haben als auch strukturiert sein können“. Heute gibt es zwei wesentliche Verfahren zur Flachglasherstellung:

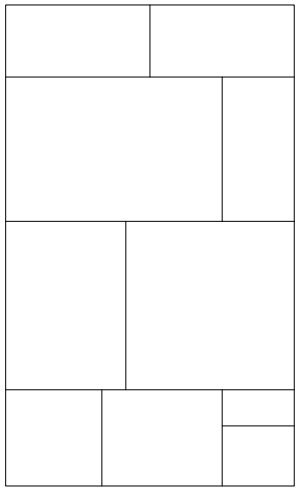
Walzverfahren: Es wird meist ornamentiertes, undursichtiges sogenanntes Gussglas her-

gestellt.

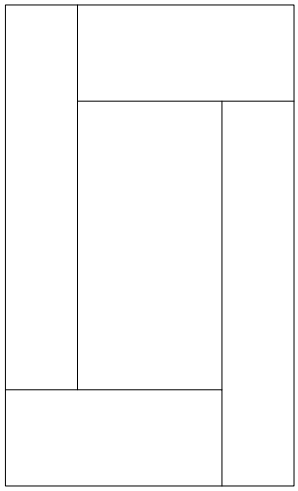
Floatglasverfahren: Dabei wird transparentes sogenanntes Spiegelglas hergestellt.

Das Floatglasverfahren – es wurde 1959 von der Firma Pilkington präsentiert – ist das günstigste und effizienteste Verfahren, transparentes Glas herzustellen. Dabei wird das flüssige Glas auf flüssiges Zinn gegossen. Die Oberfläche des flüssigen Zinns bildet eine ideal glatte Oberfläche, das Glas wird also auf beiden Seiten glatt. Da das Glas einen weit höheren Schmelzpunkt als das Zinn hat, kann es abkühlen und erstarren, während das Zinn flüssig bleibt. Floatglas ist heute laut Grabner und Roschitz [15] das Basisprodukt für die Weiterverarbeitung zu Funktionsgläsern.

Das sogenannte Schneiden des Flachglases ist nach Petzold, Marusch und Schramm [31] ein Vorgang, der ein Ritzen mit nachfolgendem Brechen darstellt. Das Ritzen wird meist mittels Schneiderädchen aus Hartmetall durchgeführt. Das nachfolgende Brechen wird mittels Biegebeanspruchung, Abbrechen oder Klopfen erreicht. Mit diesen Verfahren können meist nur Guillotineschnitte erzeugt werden. Das sind Schnitte, die immer von einem Rand des Rohlings zum gegenüberliegenden Rand verlaufen (siehe Abbildung 1.1).



Durch Guillotineschnitte herstellbar



Nicht durch Guillotineschnitte herstellbar

Abbildung 1.1: Guillotineschnitte

1.2 Problembeschreibung

Das zentrale Thema dieser Diplomarbeit ist ein industrielles Glasverschnittproblem, es wird in der Folge mit **zweidimensionales Bin Packing mit Wagen-Restriktion (2BP-WR)** bezeichnet. Das 2BP-WR ist ein aus der Industrie stammendes reales Problem mit verschiedenen Nebenbedingungen, welche in der existierenden Literatur zu zweidimensionalen Verschnittproblemen in dieser Form noch nicht berücksichtigt sind.

1.2.1 Allgemeine Beschreibung

Beim 2BP-WR sollen „kleine“ rechteckige Fertigteile, in der Folge Elemente genannt, aus „großen“ rechteckigen Rohlingen mittels Guillotineschnitt herausgeschnitten werden. Elemente können drehbar sein, das heißt, dass sie auch um 90° gedreht ausgeschnitten werden können. Die Elemente sind in logischen Gruppen zusammengefasst, dies können zum Beispiel Elemente sein, die zu einer Auftragsgruppe gehören oder in dieselbe Abteilung zur Weiterverarbeitung kommen. Ziel ist es, die Anzahl der insgesamt notwendigen Rohlinge zu minimieren.

Ein Fertigungsauftrag besteht aus einer Liste von Elementtypen. Die Beschreibung jedes Elementtyps enthält die Maße, ein Flag, ob der Elementtyp drehbar ist, die logische Gruppe der er angehört und die Anzahl der zu erzeugenden Elemente.

Das Ausschneiden der Elemente erfolgt durch eine Flachglasschneideanlage, welche ausschließlich orthogonale Guillotineschnitte mit maximaler Schnitttiefe drei erzeugen kann. Die Schnitttiefe ist die maximale Anzahl von geschachtelt ausgeführten horizontalen und vertikalen Schnitten, die zur Produktion eines Fertigteils benötigt werden. In Abbildung 1.2 ist ein mit Guillotineschnitt erzieltes Schnittmuster zu sehen, wobei bei jedem Schnitt die Schnitttiefe eingezeichnet ist.

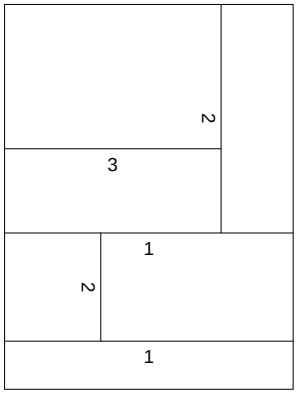


Abbildung 1.2: Schnitttiefe

Die Elemente werden am Ausgang der Schneideanlage von einem Abstellroboter übernommen. Dieser legt die Elemente auf Transportwagen ab, die auf drei Abstellplätzen stehen können. Auf einen Transportwagen dürfen nur Fertigteile derselben logischen Gruppe abgelegt werden. Ein Wechsel eines Transportwagens ist nur dann möglich, wenn entweder alle Elemente der logischen Gruppe abgelegt wurden oder eine gewisse Mindestfüllmenge erreicht wurde. Wenn eine Maximalfüllmenge erreicht wird, muss der Transportwagen gewechselt werden. Nach einem Wechsel des Transportwagens kann auch die logische Gruppe gewechselt werden, wenn diese noch nicht fertig abgearbeitet wurde.

Die Nebenbedingungen die durch die logischen Gruppen und Transportwagen entstehen, wirken sich stark auf die möglichen Lösungsverfahren aus. Die Reihenfolge der Fertigteile auf einem Rohling ist wichtig: Zum Beispiel können nach einem erfolgten Wechsel eines Trans-

portwagens nicht mehr Fertigteile auf dem bereits ausgewechselten Wagen abgelegt werden.

1.2.2 Definitionen

Die Definitionen in diesem Abschnitt dienen in der Folge zur einheitlichen Darstellung des Problems.

1.2.2.1 Probleminstanz des 2BP-WR

Eine Probleminstanz des 2BP-WR besteht aus:

- den Maßen der Rohlinge: Breite, Länge, Stärke: $B \times L \times S$ (alle Rohlinge einer Instanz sind gleich);
- der Mindestfüllmenge B_{min} und der Maximalfüllmenge B_{max} der Transportwagen; die Füllmenge eines Transportwagens ist die Summe der Stärken der darauf abgelegten Elemente. Ein Transportwagen darf erst dann ausgewechselt werden, wenn B_{min} überschritten wurde oder wenn keine Elemente der logischen Gruppe, die dem Transportwagen zugeordnet ist, mehr vorhanden sind. Wenn die Füllmenge des Transportwagens B_{max} erreicht, muss dieser ausgewechselt werden.
- der Menge der Elemente, welche durch die Menge der Elementtypen definiert ist: $\mathcal{E} = \{E_i : i \in [1 \dots K]\}$. Dabei ist K die Anzahl der Elementtypen und $E_i = (b, l, a, d, g)$, mit b: Breite, l: Länge, a: Anzahl, d: Drehbarkeit und g: logische Gruppe Ein Element des Typs E_i wird mit e_i bezeichnet. Im weiteren Verlauf werden die Attribute eines Elements bzw. seines Elementtyps mit $b(e)$, $l(e)$, $a(e)$, $d(e)$ und $g(e)$ bezeichnet. Es muss gelten:
  $b > 0$, $l > 0$ und $b < l$
  $a > 0$, $d \in (TRUE, FALSE)$ und $g > 0$.

1.2.2.2 Lösung

Eine Lösung einer Probleminstanz besteht aus einer Liste von Schnittmustern für die erforderlichen Rohlinge $\mathcal{L} = \{r_i : i \in [1 \dots N]\}$. Jedes Schnittmuster ist durch einen Schnittbaum mit maximaler Höhe drei repräsentiert und enthält eine sortierte Liste $st(r_i)$ von Knoten, die alle *top-level* Schnitte repräsentieren. Die Schnittrichtung ergibt sich aus der Schnitttiefe: Ist diese ungerade (eins und drei), handelt es sich um einen horizontalen Schnitt, ist sie gerade (zwei), handelt es sich um einen vertikalen Schnitt.

Die Knoten k_j des Baumes enthalten:

- die Schnittkoordinate $c(k_j)$ mit $c(k_j) \in (0, B)$ wenn der Knoten ungerade Schnitttiefe hat und $c(k_j) \in (0, L)$ sonst.

- eine sortierte Liste von Verweisen auf Unterbäume $st(k_j)$, wenn die Schnitttiefe es erlaubt.

Die Reihenfolge, in welcher die Elemente aus dem Rohling herausgeschnitten bzw. auf dem Rohling platziert werden, ergibt sich aus einer *depth-first* Traversierung des Schnittbaumes. Die Reihenfolge auf dem Rohling ist von oben nach unten und von links nach rechts.

Wenn es sich bei den Knoten um Blätter handelt, stellen diese endgültige Elemente dar. Das dargestellte Element liegt unter bzw. links neben dem durch das Blatt beschriebenen Schnitt. Zusätzlich zu den Attributen der Knoten enthalten die Blätter noch:

- den zum Schnitt gehörenden Elementtyp $E(k_j)$.
- die Nummer des Abstellplatzes $anr(k_j) \in \{0, 1, 2\}$ wo der Wagen steht, auf dem das Element abgelegt werden soll.
- einen möglichen Wagenwechsel $wc(k_j) \in (TRUE, FALSE)$, welcher nach dem Ausschneiden des Elements erfolgt.

Falls ein Element genau in den Bereich passt, der über bzw. rechts des Schnittes liegt, wird ein sogenannter Null-Schnitt eingeführt, der dann das Element definiert. Ein Null-Schnitt kann auch erforderlich sein, um Schnitte einer tieferen Schnittebene zu ermöglichen. In Abbildung 1.3 ist ein Schnittbaum und das daraus entstehende Schnittmuster dargestellt. Bei der Darstellung des Schnittbaumes sind die normalen Knoten eckig, die Knoten, die einen Null-Schnitt darstellen, rund und die Blätter stark umrandet. Der Verschnitt ist im Schnittmuster schraffiert dargestellt.

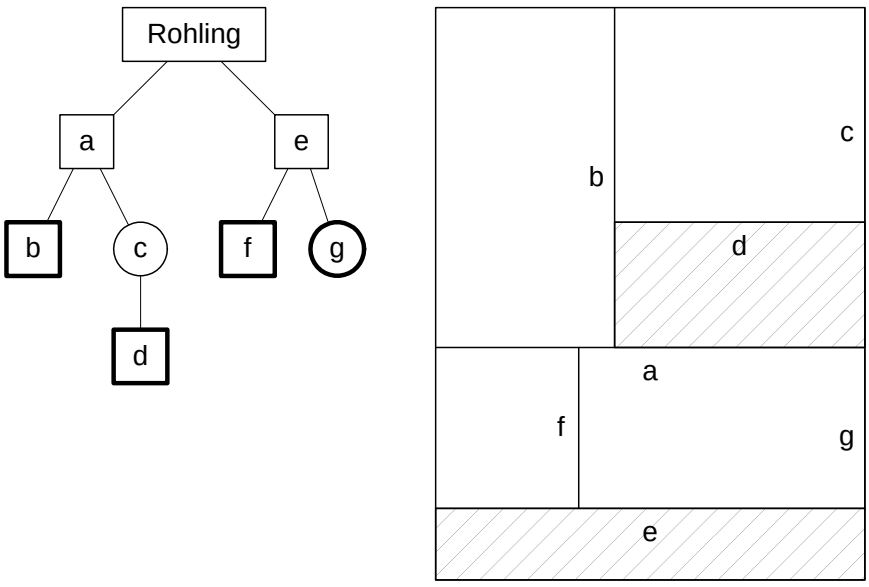


Abbildung 1.3: Schnittbaum und daraus entstehendes Schnittmuster

1.2.2.3 Logische Gruppen

Auf einem Transportwagen dürfen nur Elemente aus derselben logischen Gruppe g_i abgelegt werden. Wenn g_i keine Elemente mehr enthält oder die Füllmenge des g_i zugeordneten Transportwagens B_{max} erreicht, muss der Wagen gewechselt werden. Zusätzlich sind Wagenwechsel erlaubt, wenn ein B_{min} überschritten wird. Mit einem neuen Transportwagen kann ein Wechsel der betroffenen logischen Gruppe erfolgen, auch wenn diese noch nicht abgearbeitet ist.

Während die Elemente aus den Rohlingen herausgeschnitten werden, befinden sich die logischen Gruppen zum Zeitpunkt t in einem der folgenden vier Zustände: aktiv gesperrt (AG), aktiv frei (AF), inaktiv (I) und geleert (G). Die Reihenfolge, in der Elemente aus einem Rohling herausgeschnitten werden, entspricht der aus Abschnitt 1.2.2.2.

Zu Beginn sind alle logischen Gruppen im Zustand I. Folgende Zustandsübergänge (siehe auch Abbildung 1.4) sind für Gruppe g_i definiert:

I	→	**AG**	Ein Element aus g_i wird platziert.
AG	→	**AF**	Der Füllstand des g_i zugeordneten Wagens erreicht B_{min}.
AG	→	**G**	Es sind keine Elemente mehr in g_i vorhanden.
AF	→	**I**	Der Füllstand des g_i zugeordneten Wagens erreicht B_{max} oder es erfolgt ein Wechsel der logischen Gruppe.
AF	→	**G**	Es sind keine Elemente mehr in g_i vorhanden.

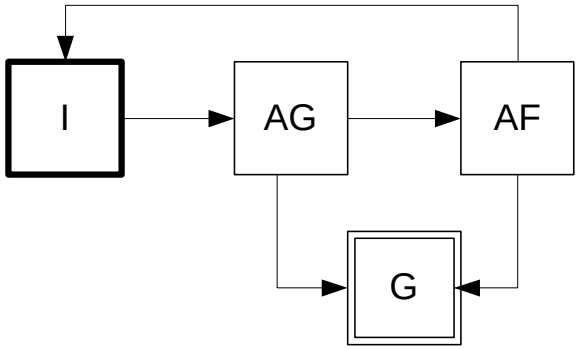


Abbildung 1.4: Zustandsübergänge der logischen Gruppen

Eine logische Gruppe g heißt *erlaubte Gruppe*, wenn:

- g aktiv ist und $g \neq \emptyset$
- g inaktiv ist und $g \neq \emptyset$ und für die Anzahl k der Gruppen im Zustand AG $k \leq 3$ gilt

Ein Element heißt *erlaubtes Element*, wenn es einer erlaubten Gruppe angehört.

1.2.2.4 Zielfunktion

Das primäre Ziel ist, die Anzahl der insgesamt notwendigen Rohlinge N zu minimieren. In der Folge wird jedoch eine erweiterte, verfeinerte Zielfunktion Z(2BP-WR) (in der Folge mit Z bezeichnet) verwendet, in der der letzte Rohling nur anteilsmäßig gerechnet wird:

$$Z = N - \frac{L - c(k_l)}{L}$$

k_l ist hierbei der letzte horizontale Schnitt der ersten Schnittebene des letzten Rohlings. Mit Z_{ALG} wird das von Algorithmus ALG erreichte Ergebnis bezeichnet.

1.2.2.5 Kontinuierliche untere Schranke

Die kontinuierliche untere Schranke KUS für die Zielfunktion (sowohl N als auch Z) ist die Fläche der zu platzierenden Elemente dividiert durch die Fläche eines Rohlings:

$$KUS = \frac{\sum_{i=1}^{K} l(e_i) \cdot b(e_i) \cdot a(e_i)}{L \cdot B}$$

Kapitel 2

Zweidimensionale Packungsprobleme

Zweidimensionale orthogonale Packungsprobleme kommen in der Praxis in vielen Varianten vor. Es handelt sich dabei zumeist um NP-schwierige Probleme, die in der Praxis oft nur mit heuristischen Verfahren näherungsweise gelöst werden können. Es sollen dabei immer „kleine“ Rechtecke (Elemente) möglichst gut in „großen“ Rechtecken (Bins/Kisten/Rohlingen) untergebracht werden. Diese Probleme entsprechen den zweidimensionalen Verschnittproblemen, bei denen Elemente aus Rohlingen herausgeschnitten werden müssen. Es wird zwischen folgenden Grundproblemen unterschieden:

Zweidimensionales Strip Packing (2SP): Ein Streifen fester Breite und unendlicher Länge wird gefüllt, wobei die benutzte Streifenlänge minimiert werden soll.

Zweidimensionales Bin Packing (2BP): Rechtecke mit fester Breite und Länge werden gefüllt, wobei die Anzahl dieser Rechtecke minimiert werden soll.

2.1 Klassifikation

Die verschiedenen Varianten des 2BP entstehen durch industrielle Anforderungen wie zum Beispiel dem Guillotineschnitt oder der Möglichkeit, die auszuschneidenden Rechtecke auch um 90° gedreht auszuschneiden. Lodi, Martello und Vigo [26] verwenden folgende Klassifikation der vier daraus entstehenden Möglichkeiten:

2BP|O|G: Die Elemente haben eine fixe Orientierung (O) und Guillotineschnitt (G) ist verlangt;

2BP|R|G: Die Elemente dürfen um 90° rotiert (R) werden und Guillotineschnitt ist verlangt;

2BP|O|F: Die Elemente haben eine fixe Orientierung und die Schnitte sind frei (F);

2BP|R|F: Die Elemente dürfen um 90° rotiert werden und die Schnitte sind frei.

Beim 2BP-WR handelt es sich um ein Problem der Klasse 2BP|R|G mit zusätzlichen Einschränkungen (siehe Abschnitt 1.2). In diesem Kapitel werden hauptsächlich Verfahren zur Lösung von Problemen der Klasse 2BP|R|G beschrieben.

2.2 Grundlegende Verfahren

Hier werden einige grundlegende Algorithmen zur Lösung von eindimensionalem *Bin Packing* und von *Strip Packing* vorgestellt.

Allen in der Folge beschriebenen Verfahren liegt das Problem in Form einer Liste L vor, welche die zu platzierenden Elemente enthält. Die Maße der Rohlinge sind immer auf eins normiert. Die Elemente aus L werden nacheinander abgearbeitet. Das jeweils zu platzierende Element wird *aktuelles Element* genannt. Nachdem ein Element platziert wurde, kann diese Entscheidung nicht mehr rückgängig gemacht werden, deshalb werden sie auch als *greedy* Algorithmen bezeichnet. Das Ergebnis von Algorithmus A angewandt auf die Liste L wird mit $\mathrm{A}(L)$ bezeichnet, das Optimum mit $\mathrm{OPT}(L)$.

2.2.1 Eindimensionales Bin Packing

Algorithmen zur Lösung von eindimensionalem *Bin Packing* stellen die Grundlage für die Lösung mehrdimensionaler Packungsprobleme dar. In diesem Abschnitt wird kurz auf die wichtigsten Aspekte und Lösungsansätze für eindimensionales *Bin Packing* eingegangen.

Das klassische eindimensionale *Bin Packing* Problem besteht aus einer Liste von Elementen $L = (e_1, e_2, ..., e_n)$, wobei jedes die Größe $s(e_i) \in (0, 1]$ hat. Diese Elemente sollen so in Bins mit Größe eins gepackt werden, dass die Anzahl der verwendeten Bins minimal ist.

Die ersten exakten Verfahren zur Lösung von eindimensionalem *Bin Packing* wurden von Gilmore und Gomory entwickelt ([12] und [13]). Dabei wird das Problem als ganzzahliges Programm formuliert, wobei jedes einzelne Packmuster einer ganzzahligen Variabel entspricht. Die Anzahl dieser Variablen kann aber exponentiell wachsen. Der Ansatz von Gilmore und Gomory war, die einzelnen sinnvollen Packmuster nach und nach durch Lösung eines verwandten Rucksackproblems zu generieren.

Coffman, Garey und Johnson geben in [5] einen Überblick über Approximationsalgorithmen für eindimensionales *Bin Packing*. Eine sehr ausführliche Beschreibung verschiedener grundlegender *Bin Packing* Verfahren gibt Mark Jarvis in seiner Diplomarbeit [19]. Hier werden einige der dort eingehend beschriebenen Verfahren kurz vorgestellt.

NF Bei *Next Fit* wird versucht, das aktuelle Element in einen aktuellen Bin hinzuzufügen. Falls dort nicht mehr genug Platz vorhanden ist, wird ein neuer Bin begonnen (dieser wird zum aktuellen Bin) und das Element dort hinzugefügt. Für alle Instanzen L gilt: $\mathrm{NF}(L) \leq 2 \cdot \mathrm{OPT}(L) - 1$

FF Bei *First Fit* wird versucht, das aktuelle Element dem erstmöglichen Bin hinzuzufügen.

Falls es in keinen bereits angefangenen passt, wird ein neuer Bin begonnen und das Element dort hinzugefügt. Für alle Instanzen L gilt: $\mathrm{FF}(L) \leq \lceil (17/10) \cdot \mathrm{OPT}(L) \rceil$ [11].

BF Bei *Best Fit* wird das aktuelle Element dem Bin hinzugefügt, der es am „besten“ aufnehmen kann, das heißt der Bin mit dem höchsten Füllstand, in den das Element noch passt. Ist in keinem Bin mehr Platz, wird ein neuer Bin begonnen und das Element dort hinzugefügt. Die Approximationsgüte von BF ist dieselbe wie die von FF [20]. Bei speziellen Instanzen können FF und BF aber sehr verschiedene Ergebnisse liefern, so sind in [21] Instanzen L angegeben, für die $\mathrm{BF}(L) = (4/3) \cdot \mathrm{FF}(L)$ gilt, und solche, für die $\mathrm{FF}(L) = (3/2) \cdot \mathrm{BF}(L)$ gilt.

FFD und BFD Bei *First Fit Decreasing* und *Best Fit Decreasing* werden die Elemente zuerst der Größe nach sortiert und dann mittels FF bzw. BF auf die Bins verteilt. Das Sortieren der Elemente führt zu einer Verbesserung des Worst-Case-Verhaltens der beiden Algorithmen. Für alle Instanzen L gilt: $\mathrm{FFD}(L) \leq (11/9) \cdot \mathrm{OPT}(L) + 4$ und $\mathrm{BFD}(L) \leq (11/9) \cdot \mathrm{OPT}(L) + 4$ [20]

2.2.2 Zweidimensionales Strip Packing

Hier werden drei grundlegende Verfahren zur Lösung von 2SP vorgestellt (siehe auch Abbildung 2.1). Es handelt sich um sogenannte *level*-orientierte Verfahren, welche die Elemente reihenweise von links nach rechts platzieren, sodass diese Abschnitte (*levels*) bilden und das erste Element die Abschnittshöhe festlegt. Diese Verfahren erzeugen mit Guillotineschnitt herstellbare Muster. Die Liste L der Elemente wird der Höhe nach absteigend sortiert, da die Verfahren dadurch, ähnlich zum eindimensionalen *Bin Packing*, eine höhere Approximationsgüte erreichen.

NFDH Beim *Next-Fit Decreasing Height* Verfahren wird der erste Abschnitt mit dem ersten Element initialisiert. Danach wird versucht, das nächste Element dem Abschnitt hinzuzufügen. Wenn es nicht hineinpasst, wird ein neuer Abschnitt mit diesem Element initialisiert. Dieser Vorgang wird solange wiederholt, bis alle Elemente platziert wurden. Für alle Instanzen L gilt: $\mathrm{NFDH}(L) \leq 2 \cdot \mathrm{OPT}(L) + 1$ [8]

FFDH Beim *First-Fit Decreasing Height* Verfahren wird der erste Abschnitt mit dem ersten Element initialisiert. Danach wird das aktuelle Element in den ersten Abschnitt gepackt, in den es hineinpasst. Wenn es in keinen Abschnitt passt, wird ein neuer Abschnitt mit diesem Element initialisiert. Für alle Instanzen L gilt: $\mathrm{FFDH}(L) \leq (17/10) \cdot \mathrm{OPT}(L) + 1$ [8]

BFDH Beim *Best-Fit Decreasing Height* Verfahren wird der erste Abschnitt mit dem ersten Element initialisiert. Danach wird das aktuelle Element auf den Abschnitt gepackt, in den es passt und bei dem der nicht benutzte horizontale Bereich minimal ist.

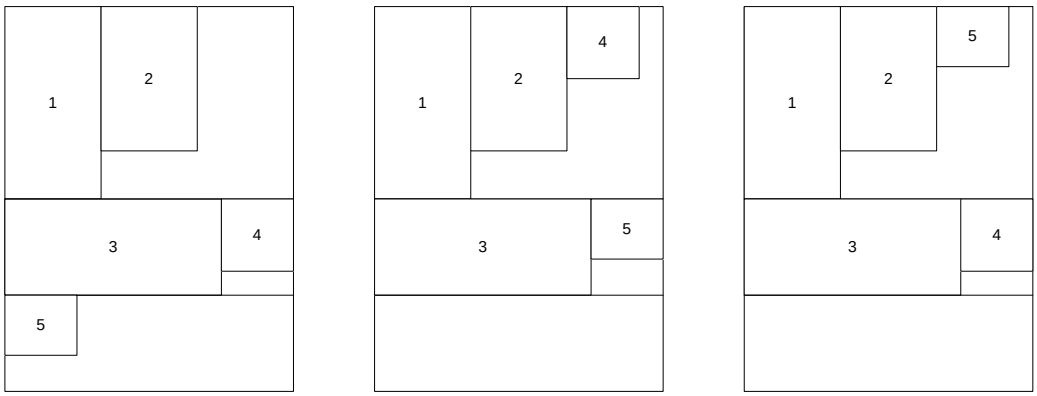


Abbildung 2.1: Einfache *Strip Packing* Verfahren

2.3 Verfahren zur Lösung von zweidimensionalem Bin Packing

Der folgende Abschnitt orientiert sich größtenteils an den von Lodi, Martello und Monaci in [25] und von Lodi, Martello und Vigo in [27] verfassten Übersichten. Es werden die Teile vorgestellt, die für die weiteren Kapitel der Diplomarbeit relevant sind. Für einen umfassenderen Überblick wird auf die beiden Übersichten sowie für einen allgemeinen Überblick über Packungsprobleme auf die kommentierte Bibliographie von Dyckhoff, Scheithauer und Terno [7] verwiesen.

Bei den meisten einfachen Verfahren aus der Literatur handelt es sich um sogenannte *greedy* Verfahren. Die Elemente werden nacheinander abgearbeitet, die Platzierung der Elemente kann im Lauf des Verfahrens nicht rückgängig gemacht werden. Dabei kann man zwei große Familien unterscheiden:

Einphasige Verfahren: Die Elemente werden direkt auf die Rohlinge gepackt

Zweiphasige Verfahren: Mittels 2SP Verfahren werden Abschnitte erzeugt, welche dann auf die Rohlinge verteilt werden

2.3.1 Greedy Heuristiken

Allen in der Folge beschriebenen Verfahren liegt das Problem in Form einer Liste L vor, welche die zu platzierenden Elemente enthält. Die Maße der Rohlinge sind auf eins normiert. Bei den zweiphasigen Verfahren werden zuerst Abschnitte erzeugt, welche danach auf die Bins verteilt werden:

HFF Chung, Garey und Johnson stellen in [4] die *Hybrid First Fit* Heuristik vor. Dabei wird in einer ersten Phase ein *Strip Packing* mittels FFDH erzeugt. Danach werden in einer zweiten Phase, die erzeugten Abschnitte der Höhe nach sortiert und mit FFD (*First*

Fit Decreasing für 1BP) auf die Bins verteilt. Chung, Garey und Johnson zeigen, dass folgende Schranke gilt $\mathrm{HFF}(L) \leq (17/8) \cdot \mathrm{OPT}(L) + 5$. Diese Schranke ist nicht scharf, denn ein *Worst-Case* Beispiel ergibt $\mathrm{HFF}(L) = (91/45) \cdot (\mathrm{OPT}(L) - 1)$

FBS Berkey und Wang stellen in [2] das *Finite Best Strip* Verfahren vor. Es handelt sich dabei um eine Variante von HFF, wobei in der ersten Phase BFDH und in der zweiten Phase BFD (*Best Fit Decreasing* für 1BP) eingesetzt wird.

HNF Frenk und Galambos stellen in [9] das *Hybrid Next Fit* Verfahren vor. In der ersten Phase wird NFDH angewandt, und die erzeugten Abschnitte werden dann mittels NFD (*NextFit Decreasing* für 1BP) auf die Bins verteilt. Frenk und Galambos [9] zeigen, dass folgende Schranke gilt: $\mathrm{HNF}(L) \leq 3,382 \cdot \mathrm{OPT}(L) + 9$

KP Lodi, Martello und Vigo stellen in [26] den *Knapsack Packing* Algorithmus vor, welcher einen Abschnitt nach dem anderen packt, wobei das erste (größte) Element den Abschnitt initialisiert und dessen Höhe vorgibt. Dieser wird dann durch Lösung des sich daraus ergebenden Rucksackproblems aufgefüllt. Die einzelnen Abschnitte werden dann heuristisch oder mittels eines lokal optimalen Algorithmus für das 1BP auf die Rohlinge verteilt.

Berkey und Wang stellen in [2] zwei einphasige Verfahren vor:

FNF *Finite Next-Fit* packt einen Abschnitt mittels NFDH und diesen sofort in den aktuellen Rohling. Wenn kein Platz mehr auf dem Rohling ist, wird ein neuer Rohling begonnen. FNF ist dasselbe Verfahren wie HNF.

FFF *Finite First-Fit* packt das aktuelle Element in den ersten Abschnitt des ersten Rohlings, in den es passt. Wenn es von keinem Abschnitt aufgenommen werden kann, wird auf dem ersten passenden Rohling oder auf einem neuen Rohling – falls auf keinem der anderen Rohlinge Platz war – ein neuer Abschnitt initialisiert. Abbildung 2.2 ist ein Beispiel für das Ergebnis von FFF.

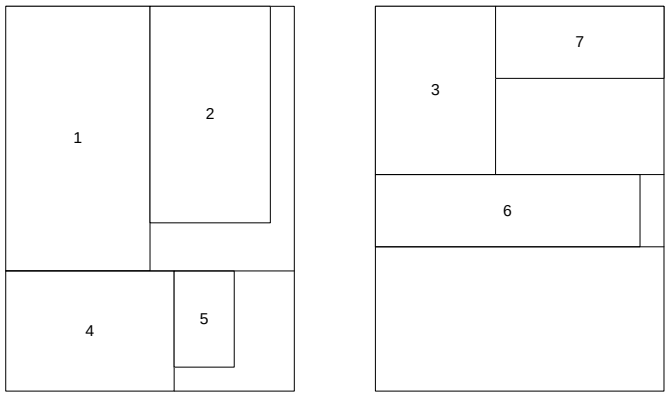


Abbildung 2.2: FFF Verfahren

Die Verfahren FFF, FBS und KP werden von Lodi, Martello und Vigo in [26] verglichen. Zumeist liefert KP bessere Ergebnisse als FBS und FFF. FBS ist fast immer besser oder gleich im Vergleich zu FFF.

2.3.2 Branch and Bound Verfahren für 2BP

Verschiedene *Branch and Bound* Verfahren für 2BP werden in der Literatur vorgestellt. Dabei kann zwischen zwei Ansätzen unterschieden werden: Es wird ein einzelner Rohling oder das Gesamtergebnis über mehrere Rohlinge optimiert. Exemplarisch werden hier zwei Verfahren vorgestellt:

- Martello und Vigo haben in [28] einen B&B Algorithmus für allgemeines zweidimensionales *Bin Packing* beschrieben. Das Verfahren basiert auf einem zweistufigen *branching* Schema, wobei die Elemente mittels eines äußeren Entscheidungsbaumes auf die Rohlinge verteilt werden. Mögliche Platzierungen der Elemente werden dann heuristisch oder mit Hilfe eines inneren Entscheidungsbaumes ermittelt. Martello und Vigo vergleichen in [28] die Ergebnisse des B&B (z) mit den oberen Schranken (OS), welche das bessere Ergebnis der Verfahren FFF und FBS ist. Die Rate OS/z liegt zumeist zwischen 100% und 110% und erreicht fallweise bis zu 130%.
- Morabito und Arenales stellen in [30] ein Verfahren vor, welches zweidimensionales *Bin Packing* mit Guillotineschnitt behandelt. In diesem Verfahren wird eine Art Schnittbaum (AND/OR Graph) als Entscheidungsbaum für das B&B verwendet. Im Gegensatz zum Verfahren von Martello und Vigo [28] wird aber immer nur ein einzelner Rohling optimiert.

2.3.3 Metaheuristiken für 2BP

2.3.3.1 Evolutionäre Algorithmen

Hopper gibt in ihrer Dissertation [17] einen umfassenden Überblick über die Verwendung von evolutionären Algorithmen zur Lösung von verschiedenen Packungsproblemen. Eine detaillierte Einführung zu evolutionären Algorithmen gibt Michalewicz in [29], siehe weiters Bäck, Fogel und Michalewicz [1] für ein umfassendes Nachschlagwerk über evolutionäre Algorithmen.

Die meisten genetischen Algorithmen zur Lösung von zweidimensionalen Packungsproblemen mit Guillotineschnitt basieren auf einer Baumdarstellung der Lösung, wobei verschiedene genetische Operatoren für diese Bäume entwickelt worden sind. Einer davon ist der von Kröger in [23] beschriebene Ansatz, bei dem ein Schnittbaum zur Darstellung verwendet wird. Für diesen Schnittbaum wird ein spezieller Rekombinationsoperator entwickelt, welcher Teilbäume austauscht. Des weiteren wird auch das Konzept der Metarechtecke eingeführt: Dabei werden aus mehreren Objekten bestehende Teilmuster „eingefroren“. Dieses Konzept reduziert die Laufzeit des Algorithmus und verbessert, im Mittel, die Ergebnisse.

Eine andere Möglichkeit sind Verfahren, die auf der Reihenfolge der Objekte (Permutationsdarstellung) basieren. Die Lösungen werden heuristisch durch die Dekodierungsfunktion generiert. Die folgenden beiden Algorithmen beruhen auf dieser Darstellungsform:

- Hwang, Kao und Horng beschreiben in [18] einen genetischen Algorithmus, welcher die Lösung als geordnete Liste von Elementen darstellt. Daraus werden mit einer *First Fit* oder einer *Best Fit* Heuristik die Lösungen abgeleitet und anschließend bewertet. Es werden damit gute Ergebnisse erreicht, welche immer besser sind als die von *Hybrid First Fit* erzielten. Die Laufzeit liegt allerdings hinter der der beiden Heuristiken zurück.
- Corno, Prinetto, Rebaudengo und Reorda beschreiben in [3] einen genetischen Algorithmus, der für ein Unternehmen entwickelt wurde, welches Glasschneidemaschinen herstellt. Die Kodiereung ist eine Permutation der zu erzeugenden Elemente. Die Elemente werden mittels einer speziellen Platzierungsheuristik, die auch problemspezifische Einschränkungen handhabt, auf die Rohlinge verteilt. Die erzielten Ergebnisse wurden mit denen verschiedener kommerzieller Optimierer (auch XOPTS) verglichen und erzielen meist gleiche oder bessere Ergebnisse.

2.3.3.2 Weitere Algorithmen

Andreas Fritsch stellt in seiner Diplomarbeit [10] einen neuen Ansatz vor: Verschnittoptimierung durch iteratives *Matching*. Um einen Rohling herzustellen, werden die zu platzierenden Rechtecke iterativ gepaart. Dadurch entstehen Metarechtecke, welche als Rechtecke angesehen werden und im nächsten Iterationsschritt wiederum gepaart werden können. Die in einem Iterationsschritt vorliegenden Rechtecke werden als Knoten in einem ungerichteten, gewichteten, vollständigen Graph interpretiert. Die Kanten des Graphen geben die möglichen Paarungen an und werden mit einem Benefitwert versehen, welcher die Ausdehnung und den Verschnitt des aus der Paarung resultierenden Rechtecks berücksichtigt. Es wird *Maximum Weight Matching* verwendet, um die bezüglich der Benefitwerte optimalen Paarungen auszuwählen. Die Paarungen werden in einem binären Baum repräsentiert, dessen Blätter die einzelnen Scheiben darstellen. Das Verfahren ist in vier Phasen unterteilt. Zuerst werden die Scheiben zu Abschnitten gepaart. Durch Vertauschen gewisser Teilbäume werden die Abschnitte nachoptimiert. Abschließend werden die Abschnitte den einzelnen Rohlingen mit FFD zugeordnet. Durch Vertauschen der Abschnitte zwischen den Rohlingen wird noch eine bessere Verteilung gesucht, die in möglichst vielen Rohlingen die Länge komplett nutzt. Das Verfahren von Fritsch liefert für die von ihm getesteten Instanzen zumeist (in 5 von 7 Fällen) bessere Ergebnisse als XOPTS.

Noch zu erwähnen ist das von Lodi, Martello und Vigo in [26] entwickelte *Tabu Search Framework*, welches für *de facto* alle Varianten des 2BP eingesetzt werden kann. Das Verfahren beginnt mit einer korrekten Lösung, die heuristisch erzeugt und in jedem Schritt modifiziert wird. Dabei wird die Platzierung einer Untermenge von Elementen geändert, wobei das Ziel ist, einen speziellen Rohling (*target bin*) zu leeren. Die in [26] beschriebenen Ergebnisse zeigen, dass mit diesem *Tabu Search Framework* deutliche Verbesserungen der Startlösungen möglich sind.

2.3.4 Schlussfolgerungen in Hinblick auf das Glasverschnittproblem

Die hier vorgestellten Algorithmen sind – aufgrund der Einschränkung durch die logischen Gruppen – nicht direkt auf das 2BP-WR anwendbar. Sie bilden aber die Grundlage für die in den weiteren Kapiteln entwickelten Verfahren. Die Einteilung der Elemente in logische Gruppen und die Füllstände der Transportwagen müssen bei diesen aber berücksichtigt werden, was zu wesentlichen Einschränkungen führt und die Verfahren verkompliziert. So dürfen zum Beispiel bei zweiphasigen Verfahren die generierten Abschnitte nicht einfach umsortiert werden, sondern es müssen Gruppengrenzen eingehalten werden.

Die meisten hier vorgestellten Verfahren nutzen Symmetrien, und die Abschnitte und Rohlinge werden normiert. Die Reihenfolge der Elemente innerhalb eines Abschnittes sowie die der Abschnitte auf einem Rohling ist unerheblich. So kann die Anzahl der möglichen Muster reduziert werden. Zusätzlich verbessert nachträgliches Umordnen die Ergebnisse. So erreicht zum Beispiel das Verfahren von Fritsch [10] ohne Vertauschen der Abschnitte zwischen den Rohlingen, keine „befriedigenden Ergebnisse“. Beim 2BP-WR hingegen spielt die Reihenfolge der Elemente eine wichtige Rolle, da sie die logischen Gruppen, die Füllstände der Transportwagen und somit auch die Wagenwechsel bestimmt. Die Wagenwechsel bilden starre Grenzen für die gewählten logischen Gruppen. Die für das 2BP-WR entwickelten Verfahren müssen also immer die Reihenfolge der Elemente in Hinblick auf die logischen Gruppen berücksichtigen sowie den aktuellen Status der logischen Gruppen und Transportwagen mitführen.

Kapitel 3

Greedy Heuristiken

In diesem Kapitel werden zwei einfache Verfahren für das 2BP-WR präsentiert, welche eine Art Grundlage oder Referenzlösung für weitere Algorithmen darstellen. Die vorliegenden Testergebnisse werden in Hinblick auf die Entwicklung weiterer komplexerer Algorithmen analysiert.

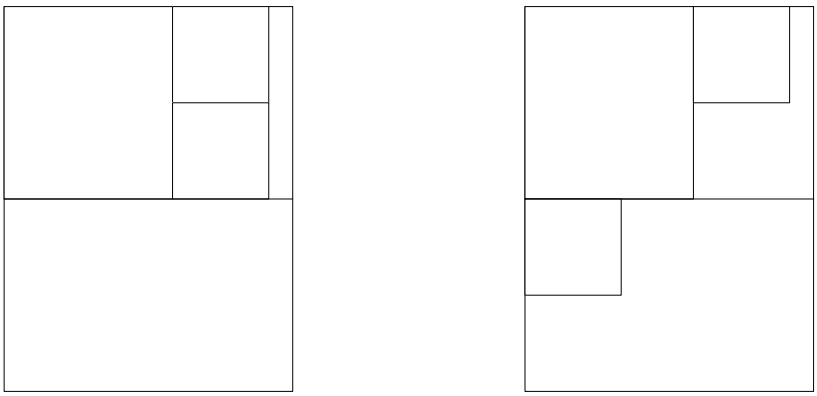
3.1 Grundlegendes

Die in der Folge beschriebenen Verfahren basieren beide auf einem *greedy* Ansatz und sind sogenannte *level*-orientierte Verfahren.

Zusätzlich zu den in Abschnitt 2.3 vorgestellten Ideen kommt noch folgender Ansatz hinzu: Es werden nicht nur Elemente nebeneinander gepackt, um Abschnitte zu bilden, sondern wenn der Abschnitt hoch genug ist, werden gleiche Elemente übereinander gestapelt. Dies kann vor allem dann ein Vorteil sein, wenn sehr „große“ und sehr „kleine“ Elemente in einem Auftrag vorkommen (siehe Abbildung 3.1) und entspricht der Randbedingung, dass der Schnittbaum Maximal höhe drei hat.

Um die Drehbarkeit der Elemente zu berücksichtigen, werden die drehbaren Elementtypen in der Liste, die den Algorithmen übergeben wird, doppelt gespeichert. Sie werden somit wie getrennte Elemente behandelt, wobei aber auf die korrekte Behandlung der Anzahl der zu platzierenden Elemente eines Elementtyps geachtet werden muss.

Beide in der Folge vorgestellten *greedy* Heuristiken müssen im *worst-case* die Liste der Elementtypen für jedes platzierte Element ganz durchlaufen. Daraus ergibt sich eine *worst-case* Laufzeit von $\mathcal{O}(N \cdot K)$, wobei N die Anzahl der Elemente und K die Anzahl der Elementtypen ist.



(a) (b)

Abbildung 3.1: *Level*-orientierte Verfahren mit (a) und ohne (b) Stapeln

3.2 Finite First Fit für das 2BP-WR

Die *Finite First Fit* Heuristik für das 2BP-WR (FFF) ist eine an unser Problem angepasste *Finite First-Fit Decreasing Height* (FFFDH) Heuristik. Diese muss gewährleisten, dass ausschließlich erlaubte Elemente (siehe Abschnitt 1.2.2) verwendet werden. Darum geht der Algorithmus nicht von der Liste der Elemente aus, sondern von den Rohlingen. Der erste Abschnitt auf dem ersten Rohling wird mit dem längsten Element initialisiert und dann mit der Länge nach geordneten erlaubten Elementen aufgefüllt. Der nächste Abschnitt wird mit dem längstmöglichen erlaubten Element initialisiert und wieder befüllt. Dies wiederholt sich, bis kein weiteres erlaubtes Element mehr auf dem Rohling hinzugefügt werden kann. Dann wird ein neuer Rohling initialisiert. Dadurch kann die Einhaltung der Bedingungen für die logischen Gruppen garantiert werden.

Algorithmus 3.1 zeigt FFF als Pseudocode. Die Rohlinge werden der Reihe nach aufgefüllt. Die Restlänge rl eines Rohlings entspricht der Länge des Rohlingteils, der noch nicht mit Abschnitten belegt ist.

Solange nicht bereits platzierte erlaubte Elemente e mit $l(e) \leq rl$ vorhanden sind, wird die Methode fülleAbschnittFFF(rl) (Algorithmus 3.2) aufgerufen. Diese erzeugt einen Abschnitt mit maximaler Länge rl, der durch den Knoten $k_{abschnitt}$ definiert ist und zum aktuellen Rohling r_{akt} hinzugefügt wird. Dabei werden passende Elemente bzw. übereinandergestapelte Elemente desselben Elementtyps zum Abschnitt hinzugefügt, wofür der Faktor t berechnet wird, der die Anzahl der zu stapelnden Elemente festlegt. Danach wird der aktuelle Rohling zur Lösung $\mathcal{L}$ hinzugefügt. Falls noch Elemente vorhanden sind, die nicht platziert wurden, wird ein neuer Rohling initialisiert, rl zurückgesetzt und der Vorgang von neuem gestartet. Die Restbreite $\hat{b}$ eines Abschnittes entspricht der Breite des Abschnittteils, der noch nicht mit Elementen belegt ist. Die mit k_{vert} und k_{hor} bezeichneten Schnitte stellen die vertikalen und horizontalen Schnitte dar, die im Verlauf von fülleAbschnittFFF zu $k_{abschnitt}$ hinzugefügt werden.

Algorithmus 3.1 Pseudocode der *FFF* Heurisitk

initialisiere Lösung $\mathcal{L} = \emptyset$
initialisiere aktuellen Rohling $r_{akt} = \emptyset$
initialisiere die Restlänge $rl = L$
while nicht platzierte Elemente e vorhanden **do**
  if $\exists$ nicht platziertes, erlaubtes Element e mit $l(e) \leq rl$ **then**
    $k_{abschnitt}$ = fülleAbschnittFFF(rl) {Siehe Algorithmus 3.2}
    füge $k_{abschnitt}$ zu r_{akt} hinzu
    $rl = L - c(k_{abschnitt})$
  else
    füge r_{akt} zu $\mathcal{L}$ hinzu
    initialisiere aktuellen Rohling $r_{akt} = \emptyset$
    $rl = L$
  end if
end while
füge r_{akt} zu $\mathcal{L}$ hinzu

3.3 Best Fit Cut

Die *Best Fit Cut* Heuristik (BFC) wählt zu Beginn eines Abschnitts ab das erste Element zufällig. Dann wird versucht, den Abschnitt zu füllen und dabei so wenig Verschnitt wie möglich zu erzeugen. Die Abschnittslänge $l(ab)$ ist die Länge des ersten Elements und wird im Verlauf des Verfahrens aktualisiert, da sie sich durch hinzugefügte Elemente ändern kann. Solange die Restbreite $\hat{b}$ des teilweise befüllten Abschnitts eine gewisse Grenze ($B \cdot WF$, $WF \in [0, 1]$) nicht unterschreitet (d.h. wenn der Abschnitt einen gewissen Füllgrad $(1 - WF)$ noch nicht erreicht hat), wird lediglich der entstehende Verschnitt des bisher befüllten Teils berücksichtigt. Wenn dieser Füllgrad hingegen erreicht oder überschritten wird, wird auch der entstehende Restverschnitt minimiert, was breitere Elemente begünstigt. Dies wird erreicht, indem zum Auffüllen des Abschnitts Elemente e_i verwendet werden, die folgende Funktion minimieren:

$$f = \begin{cases} (l(e_i) - l(ab)) \cdot (B - \hat{b}) + \begin{cases} l(e_i) \cdot (\hat{b} - b(e_i)) & wenn\ \hat{b} \leq B \cdot WF \\ 0 & sonst \end{cases} & wenn\ l(e_i) \geq l(ab) \\ (l(ab) - l(e_i)) \cdot b(e_i) + \begin{cases} l(ab) \cdot (\hat{b} - b(e_i)) & wenn\ \hat{b} \leq B \cdot WF \\ 0 & sonst \end{cases} & sonst \end{cases}$$

wobei die Elemente immer nur aus erlaubten Gruppen kommen dürfen. Elemente können auch gestapelt hinzugefügt werden, um f zu minimieren, wobei dann anstatt der Länge $l(e_i)$ die Länge $l(e_i) \cdot t$ genommen wird, dabei ist t die Anzahl der übereinander gestapelten Elemente. Wenn auf einem Rohling schon Abschnitte vorhanden sind, wird, falls die Restlänge eine gewisse Grenze ($L \cdot HF$, $HF \in [0, 1]$) unterschreitet, das längste Element für den Beginn des neuen Abschnitts gewählt.

Algorithmus 3.2 Pseudocode der *fülleAbschnittFFF* Methode

fülleAbschnittFFF(rl):
initialisiere die Restbreite $\hat{b} = B$
e_{ab} = längstes erlaubtes Element mit $l(e) \leq rl$
{e_{ab} wird hier platziert}
initialisiere $k_{abschnitt}$
setze $c(k_{abschnitt}) = L - rl + l(e_{ab})$
initialisiere vertikalen Schnitt k_{vert}
setze $c(k_{vert}) = b(e_{ab})$
füge k_{vert} zu $k_{abschnitt}$ hinzu
$\hat{b} = \hat{b} - b(e_{ab})$
while $\exists$ nicht platziertes, erlaubtes e mit $l(e) \leq l(e_{ab}) \wedge b(e) \leq \hat{b}$ **do**
e_{akt} = längstes erlaubtes Element mit $l(e) \leq l(e_{ab}) \wedge b(e) \leq \hat{b}$
$t = \min(a(e_{akt}), \lfloor c(k_{abschnitt})/l(e_{akt}) \rfloor)$
{t gibt an, wie oft das Element e_{akt} vom Typ E_{akt} gestapelt werden kann}
{e_{akt} wird hier t-mal platziert}
initialisiere k_{vert}
if $t = 1$ **then**
if $l(e_{akt}) < l(e_{ab})$ **then**
setze $c(k_{vert}) = B - \hat{b} + b(e_{akt})$
initialisiere k_{hor}
setze $c(k_{hor}) = L - rl + l(e_{akt})$
füge k_{hor} zu k_{vert} hinzu
else
setze $c(k_{vert}) = B - \hat{b} + b(e_{akt})$
end if
else
setze $c(k_{vert}) = B - \hat{b} + b(e_{akt})$
setze $tmprl = rl$
for $i = 1$ bis t **do**
initialisiere k_{hor}
setze $c(k_{hor}) = L - tmprl + l(e_{akt})$
füge k_{hor} zu k_{vert} hinzu
$tmprl = tmprl - l(e_{akt})$
end for
end if
$\hat{b} = \hat{b} - b(e_{akt})$
füge k_{vert} zu $k_{abschnitt}$ hinzu
end while
return $k_{abschnitt}$

Algorithmus 3.3 zeigt BFC als Pseudocode. Dieser folgt demselben Prinzip wie der Pseudocode für FFF, der Unterschied liegt in der Methode fülleAbschnittBFC, welche sich von fülleAbschnitFFF ausschließlich in der Auswahl der zu platzierenden Elemente unterscheidet.

Algorithmus 3.3 Pseudocode der *Best Fit Cut* Heurisitk

initialisiere $\mathcal{L} = \emptyset$
initialisiere die Lösung $\mathcal{L} = \emptyset$
initialisiere den aktuellen Rohling $r_{akt} = \emptyset$
initialisiere die Restlänge $rl = L$
while nicht platzierte Elemente e vorhanden **do**
  if $\exists$ nicht platziertes, erlaubtes e mit $l(e) \leq rl$ **then**
    $k_{abschnitt}$ = fülleAbschnittBFC(rl) {Siehe Algorithmus 3.4}
    füge $k_{abschnitt}$ zu r_{akt} hinzu
    $rl = L - c(k_{abschnitt})$
  else
    füge r_{akt} zu $\mathcal{L}$ hinzu
    initialisiert aktuellen Rohling $r_{akt} = \{\emptyset\}$
    $rl = L$
  end if
end while
füge r_{akt} zu $\mathcal{L}$ hinzu

3.4 Ergebnisse und Schlussfolgerungen

In diesem Abschnitt werden die Ergebnisse der vorgestellten Verfahren anhand spezieller Instanzen näher betrachtet und Schlussfolgerungen in Hinblick auf weitere Verfahren gezogen.

Einen Überblick über die verwendeten Testdaten sowie über die Ergebnisse des kommerziellen Optimierers XOPTS gibt Abschnitt 6.1.

Der BFC Algorithmus wurde mit verschiedenen Werten für WF und HF getestet. In Tabelle 6.3 sind die Ergebnisse für FFF und für folgende Varianten von BFC aufgelistet:

BFC22 $WF = 1/2$ und $HF = 1/2$

BFC24 $WF = 1/2$ und $HF = 1/4$

BFC42 $WF = 1/4$ und $HF = 1/2$

BFC44 $WF = 1/4$ und $HF = 1/4$

3.4.1 Einige Ergebnisse im Detail

Die genauen Daten der in diesem Anschnitt behandelten Instanzen real_21 und real_37 können in Appendix C nachgelesen werden. Die beiden Instanzen haben folgende Eckdaten:

Algorithmus 3.4 Pseudocode der *fülleAbschnittBFC* Methode

fülleAbschnittBFC(rl):
initialisiere die Restbreite $\hat{b} = B$
if (erster Abschnitt auf Rohling) $\vee$ ($rl \geq L \cdot HF$) **then**
e_{ab} = zufälliges erlaubtes Element e mit $l(e) < rl$
else
e_{ab} = längstes erlaubtes Element e mit $l(e) < rl$
end if
{e_{ab} wird hier platziert}
initialisiere $k_{abschnitt}$
setze $c(k_{abschnitt}) = L - rl + l(e_{ab})$
initialisiere vertikalen Schnitt k_{vert}
setze $c(k_{vert}) = b(e_{ab})$
füge k_{vert} zu $k_{abschnitt}$ hinzu
$\hat{b} = \hat{b} - b(e_{ab})$
while $\exists$ nicht platziertes, erlaubtes e mit $(l(e) \leq rl) \wedge (b(e) \leq \hat{(}b)$ **do**
{in der Folge gilt: $t = \min(a(e), \lfloor c(k_{abschnitt})/l(e) \rfloor)$}
if $t = a(e)$ **then**
e_{akt} = erlaubtes e, sodass f durch $t \cdot e$ minimiert wird
{$t \cdot e$ bedeutet e wird t-mal gestapelt}
else
e_{akt} = erlaubtes e, sodass f durch $t \cdot e$ minimiert wird
oder
e_{akt} = erlaubtes e, sodass f durch $(t+1) \cdot e$ minimiert wird, $t = t + 1$
{Hier wird versucht, die bisherige Abschnittshöhe zu überschreiten}
end if
{e_{akt} wird hier t-mal platziert}
initialisiere k_{vert}
if $t = 1$ **then**
if $l(e_{akt}) < l(e_{ab})$ **then**
setze $c(k_{vert}) = B - \hat{b} + b(e_{akt})$
initialisiere k_{hor}
setze $c(k_{hor}) = L - rl + l(e_{akt})$
füge k_{hor} zu k_{vert} hinzu
else
setze $c(k_{vert}) = B - \hat{b} + b(e_{akt})$
end if
else
setze $c(k_{vert}) = B - \hat{b} + b(e_{akt})$
setze $tmprl = rl$
for $i = 1$ bis t **do**
initialisiere k_{hor}
setze $c(k_{hor}) = L - tmprl + l(e_{akt})$
füge k_{hor} zu k_{vert} hinzu
$tmprl = tmprl - l(e_{akt})$
end for
end if
$\hat{b} = \hat{b} - b(e_{akt})$
füge k_{vert} zu $k_{abschnitt}$ hinzu
end while
return $k_{abschnitt}$

real_21 6 Gruppen, 13 Elementtypen, 151 Elemente, $KUS = 3,53$, $Z_{XOPTS} = 3,78$

real_37 15 Gruppen, 23 Elementtypen, 60 Elemente, $KUS = 6,88$, $Z_{XOPTS} = 7,89$

Die jeweils aktiven logischen Gruppen werden unterschieden, indem die zu ihnen gehörenden Elemente grün, blau oder rot eingefärbt sind. Elemente, nach denen ein Wagenwechsel erfolgt, sind dünkler eingefärbt. Die Reihenfolge, in der die Elemente aus den abgebildeten Rohlingen herausgeschnitten werden müssen, um die Einhaltung der Nebenbedingungen zu gewährleisten, ist von oben nach unten und von links nach rechts.

Abbildung 3.2 zeigt das von der FFF Heuristik erzielte Ergebnis für die Instanz real_21. Das FFF Verfahren erreicht für real_21 einen Wert von $Z = 4,00$, das entspricht $105,8\%$ des von XOPTS erzielten Ergebnisses und $113,11\%$ der KUS. Aus Abbildung 3.2 ist ersichtlich, dass das FFF Verfahren aufgestellte Elemente (das sind Elemente, deren längere Seite parallel zur längeren Seite des Rohlings verläuft) bevorzugt. Dies liegt in der Natur des Verfahrens, da es die längeren Elemente zuerst platziert.

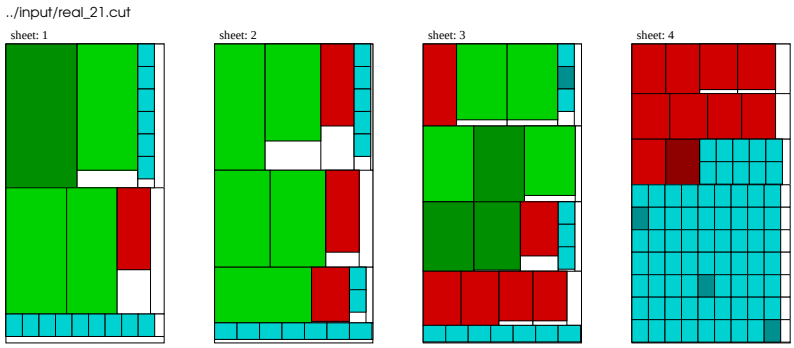


Abbildung 3.2: FFF_real_21

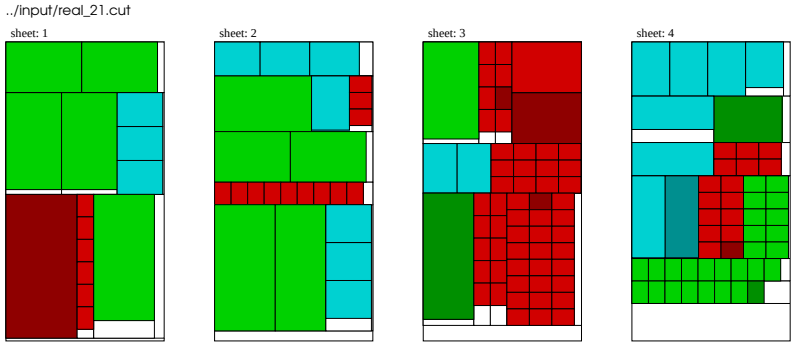


Abbildung 3.3: BFC22_real_21

In den Abbildungen 3.3 und 3.4 ist jeweils ein Ergebnis von BFC22 und BFC44 zu sehen. Diese Ergebnisse sind nicht eindeutig, da BFC ein stochastisches Verfahren ist. Die Mittelwerte, die Standardabweichungen sowie das beste erzielte Ergebnis für jeweils zehn Durchläufe können in Tabelle 6.3 nachgelesen werden.

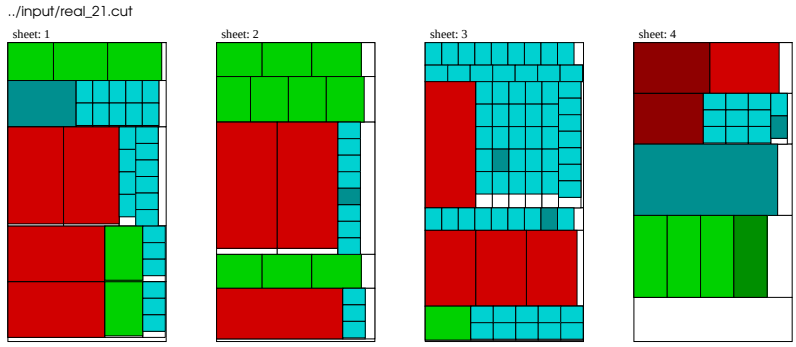


Abbildung 3.4: BFC44_real_21

Das in Abbildung 3.3 dargestellte Ergebnis erreicht einen Wert von $Z = 3,87$, das entspricht $102,38\%$ des von XOPTS erzielten Ergebnisses und $109,63\%$ der KUS. Das in 3.4 dargestellte Ergebnis erreicht einen Wert von $Z = 3,85$, das entspricht $101,85\%$ des von XOPTS erzielten Ergebnisses und $109,07\%$ der KUS. Für diese speziellen Ergebnisse ist also BFC um $0,13$ bzw. $0,15$ Rohlinge besser als FFF. Die durschnittlichen Werte sind nur mehr um $0,09$ Rohlinge besser für BFC22 und sogar um $0,03$ Rohlinge schlechter für BFC44 (siehe Tabelle 6.3).

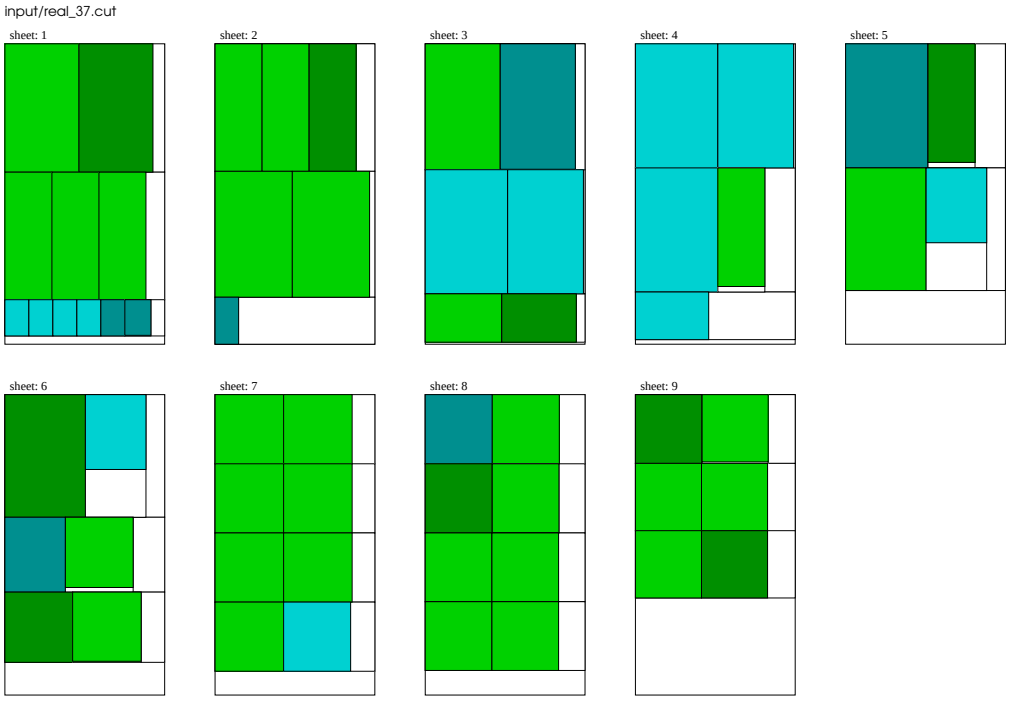


Abbildung 3.5: FFF_real_37

In Abbildung 3.5 ist das Ergebnis für real_37 der FFF Heuristik dargestellt. Das FFF Verfahren erreicht für real_37 einen Wert von $Z = 8,68$, das entspricht $110,01\%$ des von XOPTS erzielten Ergebnisses und $126,16\%$ der KUS.

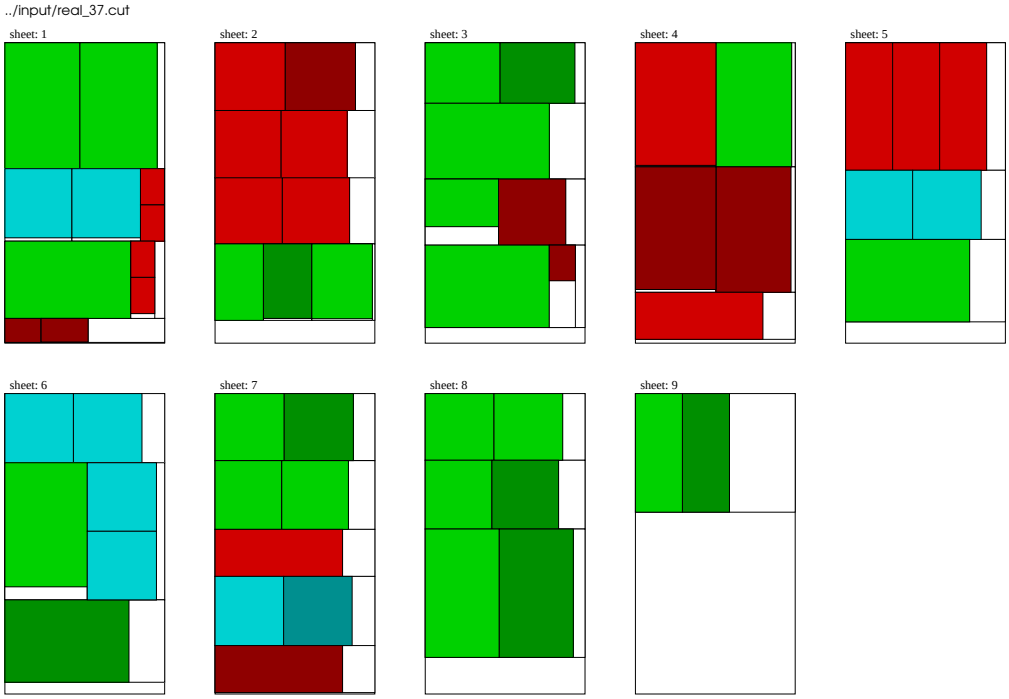


Abbildung 3.6: BFC22_real_37 A

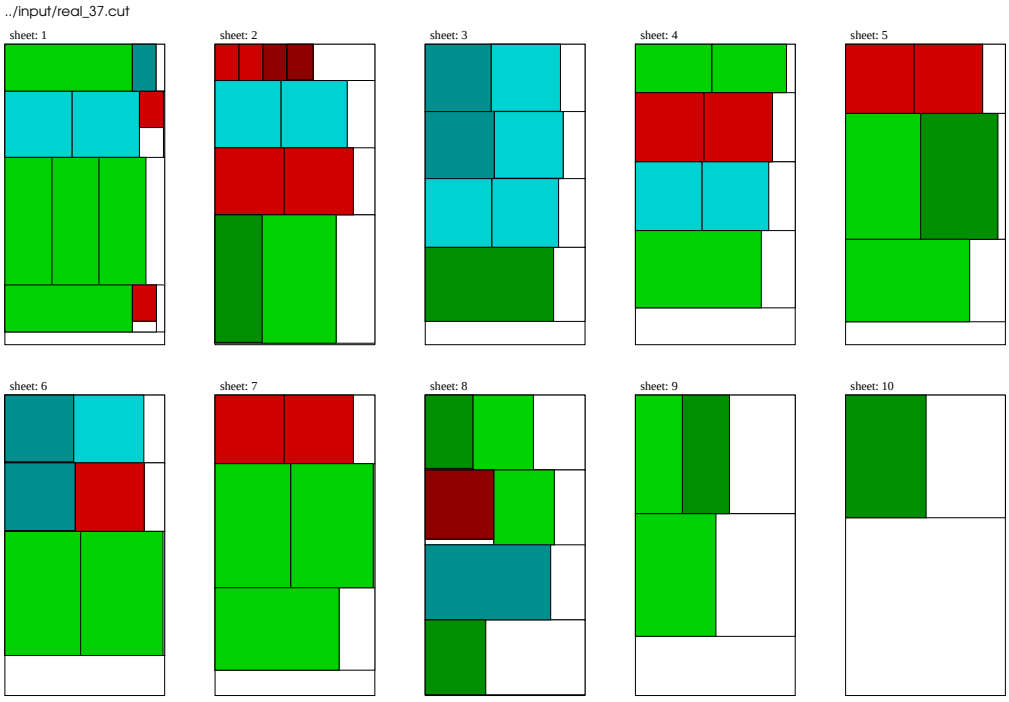


Abbildung 3.7: BFC22_real_37 B

In Abbildung 3.6 ist ein Ergebnis für real_37 der BFC22 Heuristik zu sehen. Das Ergebnis in Abbildung 3.6 erreicht einen Wert von $Z = 8,39$, das entspricht $106,34\%$ des von XOPTS erzielten Ergebnisses und $121,95\%$ der KUS. Dieses spezielle Ergebnis ist um $0,29$ Rohlinge besser als das von FFF erreichte. Der Mittelwert der Ergebnisse ist um $0,08$ Rohlinge schlechter als das von FFF erreichte Resultat (siehe auch Tabelle 6.3).

In Abbildung 3.7 ist auch ein Ergebnis für real_37 der BFC22 Heuristik zu sehen. Das Ergebnis in Abbildung 3.7 erreicht einen Wert von $Z = 9,40$, das entspricht $119,14\%$ des von XOPTS erzielten Ergebnisses und $136,63\%$ der KUS. Die Differenz zwischen den beiden Ergebnissen des BFC Verfahrens liegt bei $1,01$ Rohlingen.

3.4.2 Schlussfolgerungen

Anhand der in Tabelle 6.3 und Abschnitt 3.4.1 beschriebenen Ergebnisse können einige Rückschlüsse über die Schwächen der beiden Heuristiken gezogen und auch Ideen für weitere Algorithmen gewonnen werden.

Die FFF Heuristik liefert in kurzer Zeit akzeptable Ergebnisse. Sie liegen jedoch meist hinter denen von XOPTS zurück, und es ist zu erwarten, dass mit komplexeren Verfahren bessere Resultate erzielt werden können. Die Ergebnisse der FFF Heuristik stellen aber eine gute Referenz und Vergleichsmöglichkeit dar.

Die BFC Heuristik ist im Schnitt etwas schlechter (siehe Tabelle 6.4 und Abbildung 6.1) als die FFF Heuristik, liefert aber teilweise sehr dicht befüllte Abschnitte. Die abgebildeten Ergebnisse lassen noch Spielraum für verbesserte Verfahren erkennen, denn sie sind eindeutig schlechter als jene von XOPTS. Dies liegt daran, dass es sich um *greedy* Algorithmen handelt, die eine Änderung einer „Fehlentscheidung“ unmöglich machen. So könnte beispielsweise der zweite Abschnitt des ersten Rohlings der Abbildung 3.2 mit anderen Elementen als dem letztplatzierten roten Element besser befüllt werden. Das BFC Verfahren kann hingegen sehr gute Abschnitte erzeugen. Beachtenswert ist zum Beispiel der zweite Abschnitt auf dem ersten Rohling von Abbildung 3.3. Der letzte Abschnitt auf dem dritten Rohling derselben Abbildung zeigt, was der Faktor WF bewirkt: Nachdem die Rohlingsmitte überschritten wird, werden breitere Elemente (in diesem Fall dasselbe Element aber gedreht) bevorzugt. Die „Fehlentscheidungen“ können beim BFC Verfahren negative Auswirkungen haben. In Abbildung 3.7 hätte ein Rohling gespart werden können, wenn die letzten beiden Elemente gedreht platziert worden wären.

Bei der Entwicklung weiterer Verfahren sollte einerseits versucht werden, die Abschnitte besser zu füllen oder sogar ganze Rohlinge zu optimieren, indem zwischen verschiedenen Möglichkeiten gewählt werden kann. Dafür eignet sich zum Beispiel *Branch and Bound*. Andererseits könnten aber auch Metaheuristiken erfolgreich sein, die eine Art BFC Heuristik verwenden und die Reihenfolge der Objekte gezielter bestimmen.

Kapitel 4

Branch and Bound Verfahren

In diesem Kapitel werden zwei Verfahren vorgestellt, welche *Branch and Bound* (B&B) verwenden. Das erste Verfahren produziert Abschnitte mittels B&B und fügt diese dann mit FFDH zu Rohlingen zusammen. Das zweite Verfahren erzeugt ganze Rohlinge mittels B&B. Beide Verfahren halten die Bedingungen hinsichtlich der logischen Gruppen und der Schnitttiefe ein.

4.1 Branch and Bound

Branch and Bound ermöglicht eine komplette Aufzählung aller möglichen Lösungen eines Problems, ohne sie alle einzeln berücksichtigen zu müssen. Die folgende Beschreibung des allgemeinen B&B kommt von Korte und Vygen aus [22].

Um B&B auf ein Optimierungsproblem (nehmen wir hier ein Minimierungsproblem an) anwenden zu können, müssen folgende Operationen möglich sein:

„branch“ Eine gegebene Untermenge aller möglichen Lösungen kann in mindestens zwei nichtleere Untermengen partitioniert werden.

„bound“ Für eine durch iteratives *branching* erhaltene Untermenge kann eine untere Schranke für alle in dieser Untermenge vorhandenen Lösungen berechnet werden.

Das allgemeine B&B Verfahren ist in Algorithmus 4.1 beschrieben.

4.2 BBHEU

BBHEU ist ein zweiphasiger Algorithmus, der dem Prinzip der einfachen zweiphasigen Verfahren aus Abschnitt 2.3 folgt. Er besteht aus einer *strip packing* Phase (Phase 1) und einer *bin packing* Phase (Phase 2).

Algorithmus 4.1 Allgemeines B&B Verfahren

initialisiere Subproblemliste $PL = \{S\}$ {S ist die Menge aller möglichen Lösungen}
$O := \infty$ {O obere Schranke, kann auch heuristisch berechnet werden, um einen besseren Wert zu erhalten}

initialisiere Lösung L
loop
  if $PL = \emptyset$ **then**
    return L
  else
    wähle ein Subproblem $X \in PL$
    $PL = PL \setminus \{X\}$
  end if
  „branch“: partitioniere $X = X_1 \cup ... \cup X_k$
  for $i = 1$ bis k **do**
    „bound“: berechne eine untere Schranke U für die Kosten jeder Lösung in X_i
    if $|X_i| == 1$ (mit $X_i = \{x\}$) und $kosten(x) < O$ **then**
      $O := kosten(x)$ und $L := x$
    end if
    if $|X_i| > 1$ und $U < O$ **then**
      $PL = PL \cup \{X_i\}$
    end if
  end for
end loop

Während Phase 1 werden mittels *Branch and Bound* Abschnitte ab auf einem unendlich langen Streifen mit Breite B (entspricht der Rohlingsbreite) erzeugt, deren Länge $l(b)$ kleiner als die Rohlingslänge L ist. Wenn auf einem Abschnitt ein Wechsel einer logischen Gruppe stattfindet, wird er zum Grenzabschnitt.

In Phase 2 werden die durch Phase 1 erzeugten Abschnitte mittels einer angepassten FFD Heuristik auf die Rohlinge verteilt. Zwischen zwei Grenzabschnitten kann die Reihenfolge der Abschnitte geändert werden, um eine bessere Anordnung der Abschnitte auf den Rohlingen zu ermöglichen. Die Grenzabschnitte sind hingegen fixiert.

4.2.1 Phase 1

Es werden solange Abschnitte erzeugt, bis keine zu platzierenden Elemente mehr vorhanden sind. Die Erzeugung der Abschnitte geschieht mittels *Branch and Bound* Verfahren (siehe Algorithmus 4.2). Dabei soll der relative Verschnitt der einzelnen Abschnitte minimiert werden. Der relative Verschnitt rV eines Abschnitts ab entspricht der freien Fläche des Abschnitts (Gesamtfläche weniger der Fläche der im Abschnitt platzierten Elemente) relativ zur Gesamtfläche des Abschnitts:

$$rV(ab) = \frac{l(ab) \cdot B - \sum_{e \in ab} l(e) \cdot b(e)}{l(ab) \cdot B}$$

Zu Beginn der Phase 1 wird die Subproblemliste PL mit allen möglichen Abschnitten, welche ein Element oder mehrere gestapelte Elemente desselben Elementtyps enthalten, initialisiert und das B&B Verfahren gestartet. Dabei werden immer nur Elemente aus erlaubten Gruppen verwendet. Außerdem wird jeder Abschnitt, auf dem ein Wechsel einer logischen Gruppe erfolgt, zu einem Grenzabschnitt. Dies ist notwendig, um die korrekte Behandlung der logischen Gruppen in Phase 2 zu ermöglichen. Die fertiggestellten Abschnitte werden zu einer Abschnittsliste AL hinzugefügt, und der nächste Abschnitt wird erzeugt. Die Anzahl a der noch zu platzierenden Elemente e eines Elementtyps E wird während Phase 1 laufend aktualisiert.

Ein Abschnitt ab enthält Elemente e, und die Abschnittslänge ist durch $l(ab)$ gegeben. Die lokale untere Schranke LUS und die globale obere Schranke GOS für den Verschnitt eines Abschnittes ab werden wie folgt berechnet:

$$\begin{aligned} LUS &= \frac{\sum_{e \in ab} l(ab) \cdot b(e) - \sum_{e \in ab} l(e) \cdot b(e)}{l(ab) \cdot B} \\ GOS &= rV(ab) \end{aligned}$$

Die LUS stellt den Verschnitt, der bis zum letzten Element des Abschnittes entsteht, dar. Die GOS hingegen stellt den gesamten Verschnitt eines Abschnittes dar. Durch die Verwendung des relativen Verschnitts wird eine Bevorzugung von kürzeren Abschnitten vermieden.

Algorithmus 4.2 Pseudocode BBHEU Phase 1

main:
  initialisiere Abschnittsliste AL = $\emptyset$
  while nicht platzierte Elemente vorhanden **do**
    initialisiere Subproblemliste PL
    initialisiere Abschnitt *best* =NULL
    intialisiere $GOS = 1$
    while PL ist nicht leer **do**
      get $ab \in$ PL
      BB(ab)
    end while
    füge *best* zu AL hinzu und markiere Elemente von *best* als platziert
  end while

BB(ab**):**
  berechne LUS von ab
  if LUS > GOS **then**
    sicher schlechtere Lösung, return
  end if
  teile(ab)

teile(ab**):**
  for all nicht platzierte erlaubte Elemente e_i mit $b(e_i) \leq \hat{b}(ab)$ **do**
    $\{\hat{b}(ab)$ ist die noch freie Breite von $ab\}$
    for $i = 1$ bis $\min(\lfloor L/l(e_i) \rfloor, a(e_i))$ **do**
      initialisiere nab=$ab \cup (i \cdot e_i)$ $\{(i \cdot e_i)$ e_i wird i-mal gestapelt$\}$
      füge nab zu PL hinzu
      if relativer Verschnitt(nab) < GOS **then**
        GOS = relativer Verschnitt(nab)
        $best = ab$
      end if
    end for
  end for

4.2.2 Phase 2

In Phase 2 werden die Abschnitte auf den Rohlingen platziert, wobei die Reihenfolge der Grenzabschnitte beibehalten wird. Die Reihenfolge der Abschnitte, die zwischen zwei Grenzabschnitten liegen, kann aber verändert werden, um eine möglichst gute Anordnung der Abschnitte auf den Rohlingen zu ermöglichen.

Zuerst werden die Abschnitte zwischen zwei Grenzabschnitten der Länge nach sortiert. Dann werden diese mittels *First Fit* Heuristik zwischen den benachbarten Grenzabschnitten auf die Rohlinge verteilt.

4.3 BBALG

Der hier beschriebene Algorithmus basiert auf der Idee von Morabito und Arenales in [30], die einzelnen Rohlinge durch Maximierung der befüllten Rohlingsfläche zu optimieren.

4.3.1 Algorithmus

Der Algorithmus ist im Wesentlichen eine Erweiterung des BBHEU Verfahrens von abschnittsweiser Optimierung hin zur Optimierung ganzer Rohlinge. Die unvollständig gefüllten Rohlinge stellen die Subprobleme für das B&B Verfahren dar. Die Partitionsfunktion erzeugt neue Subprobleme, indem zum letzten Abschnitt eines Subproblems neue Elemente hinzugefügt werden oder, wenn dies nicht mehr möglich ist, ein neuer Abschnitt zum Rohling hinzugefügt wird. Die aktuelle Gruppenbelegung sowie der aktuelle Wagenfüllstand werden bei jedem Subproblem mitgeführt, um ein korrektes Ergebnis unter Einhaltung der Nebenbedingungen zu ermöglichen.

Die Optimierung der Rohlinge erfolgt durch Maximierung der befüllten Rohlingsfläche. Wenn zwei Subprobleme dieselbe befüllte Rohlingsfläche aufweisen, wird jener mit der größeren Restlänge (rl) für besser befunden.

Als Globale Untere Schranke für die Befüllung eines Rohlings (GUS) wird die befüllte Fläche (F_b) des aktuell besten Subproblems verwendet. Ein Subproblem s enthält die platzierten Elemente e, somit ergibt sich die Formel:

$$F_b(s) \quad = \quad \sum_{e \in s} b(e) \cdot l(e)$$

Bevor das B&B Verfahren gestartet wird, wird eine erste GUS mittels FFF Heuristik berechnet.

Die Lokale Obere Schranke für die Befüllung eines Subproblems s ($LOS(s)$) ergibt sich aus der bisher befüllten Fläche ($F_b(s)$) und einer Abschätzung der Fläche von s, die noch befüllt werden kann. Die unterhalb des letzten Abschnitts von s liegende Fläche ($F_u(s)$) und die rechts vom letzten Element des letzten Abschnitts liegende Fläche ($F_r(s)$) können noch befüllt werden. In Abbildung 4.1 sind diese Flächen schematisch dargestellt.

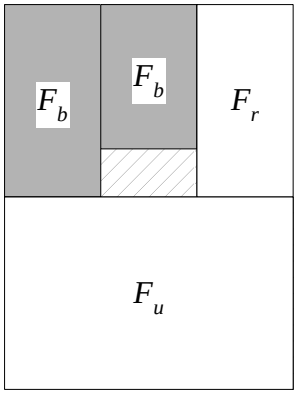


Abbildung 4.1: Schematische Darstellung der Flächen eines zum Teil befüllten Rohlings

$P_u(s)$ und $P_r(s)$ sind die Mengen der Elemente, die in den zugehörigen Flächen noch Platz haben. Somit ergibt sich:

$$LOS(s) \quad = \quad F_b(s) + \min\left(F_u(s), \sum_{e \in P_u(s)} b(e) \cdot l(e)\right) + \min\left(F_r(s), \sum_{e \in P_r(s)} b(e) \cdot l(e)\right)$$

Das BBALG Verfahren wird in Algorithmus 4.3 beschrieben.

4.3.2 Erweiterungen

Einige Ergebnisse des Algorithmus BBALG (siehe auch Abschnitt 4.4) haben gezeigt, dass einerseits sehr gut gepackte Rohlinge erzeugt werden, dies andererseits aber für die Gesamtlösung negative Folgen haben kann. Es werden „kleine“ Elemente im Allgemeinen zu Beginn verbraucht, obwohl sie später mit „größeren“ Elementen kombiniert werden könnten, die „größeren“ untereinander aber nicht. Dies führt dazu, dass zu Beginn sehr volle Rohlinge erzeugt werden, die Rohlinge am Ende der Lösung aber relativ viel Verschnitt aufweisen (siehe z.B. Abbildung 4.8).

Deshalb sind zwei Parameter eingeführt worden, mit welchen die Verwendung der Elemente mit der größtmöglichen Fläche erzwungen wird:

FH Solange die Restlänge des Rohlings L/FH übersteigt, wird die Verwendung des Elements mit der größtmöglichen Fläche erzwungen.

FW Solange die Restbreite des Rohlings W/FW übersteigt, wird die Verwendung des Elements mit der größtmöglichen Fläche erzwungen.

Bei Elementen mit gleicher Fläche werden die längeren bevorzugt eingesetzt. Die Verwendung von bestimmten Elementen kann erzwungen werden, indem im B&B anstatt aller Möglichkeiten, nur die, die das bestimmte Element beinhalten, berücksichtigt werden.

Algorithmus 4.3 Pseudocode BBALG

main:
initialisiere Lösung $\mathcal{L} = \emptyset$
while nicht platzierte Elemente vorhanden **do**
initialisiere Subproblemliste PL
initialisiere Rohling *best* =NULL
berechne *GUS* mittels FFF
while PL is not empty **do**
get $s \in$ PL
BB(s)
end while
füge *best* zur Lösung $\mathcal{L}$ hinzu und markiere Elemente von *best* als platziert
end while

BB(s):
Berechne $LOS(s)$
if $LOS(s) < GUS$ **then**
sicher schlechtere Lösung, return
end if
teile(s)

teile(s):
sei ab der zuletzt befüllte Abschnitt von s
{Wenn möglich werden Elemente zu ab hinzugefügt.}
for all nicht platzierte erlaubte Elemente e_i mit $b(e_i) \leq \hat{b}(ab)$ **do**
{$\hat{b}(ab)$ ist die noch freie Breite von ab}
for $i = 1$ bis $\min(\lfloor (l(ab) + rl(s))/l(e_i) \rfloor, a(e_i))$ **do**
initialisiere Abschnitt nab=$ab \cup (i \cdot e_i)$ {$(i \cdot e_i)$ e_i wird i-mal gestapelt}
initialisiere Rohling ns=$s \cup nab$
füge ns zu PL hinzu
if $F_b(ns) > GUS$ oder ($F_b(ns) = GUS$ und $rl(s) > rl(best)$) **then**
$GUS = F_b(ns)$
$best = ns$
end if
end for
end for
{Ansonsten muss eine neuer Abschnitt hinzugefügt werden.}
if zu ab konnte kein Element hinzugefügt werden **then**
for all nicht platzierte erlaubte Elemente e_i **do**
for $i = 1$ bis $\min(\lfloor rl(s)/l(e_i) \rfloor, a(e_i))$ **do**
initialisiere Abschnitt nab=$i \cdot e_i$ {$(i \cdot e_i)$ e_i wird i-mal gestapelt}
initialisiere Rohling ns=$s \cup nab$
füge ns zu PL hinzu
if $F_b(ns) > GUS$ oder ($F_b(ns) = GUS$ und $rl(s) > rl(best)$) **then**
$GUS = F_b(ns)$
$best = ns$
end if
end for
end for
end if

4.4 Ergebnisse und Schlussfolgerungen

Es wurden BBHEU und folgende Varianten von BBALG getestet:

BBALG1 Hier wird keine Verwendung eines Elements erzwungen

BBALG2 Hier wird als erstes Element des Rohlings das größtmögliche verwendet

BBALG3 FH = 2 Hier wird die Verwendung des Elements mit der größtmöglichen Fläche erzwungen, solange die Restlänge des Rohlings größer als $L/2$ ist

BBALG4 FW = 2 Hier wird die Verwendung des Elements mit der größtmöglichen Fläche erzwungen, solange die Restbreite des Rohlings größer als $B/2$ ist

BBALG5 FH = 2 und FW = 2 Hier wird die Verwendung des Elements mit der größtmöglichen Fläche erzwungen, solange die Restlänge des Rohlings größer als $L/2$ oder die Restbreite größer als $B/2$ ist

Aufgrund der endlichen Kapazität des benutzten Rechners muss das B&B bei einigen Instanzen abgebrochen werden. Der Abbruch erfolgt, wenn im Verlauf des B&B mehr als 3 000 000 Subprobleme in der Problemliste vorhanden sind. Es wird dann der bisher beste erzeugte Rohling verwendet. Auf die nachfolgenden Rohlinge wird wieder das B&B angwendet.

Die gesammelten Ergebnisse von BBHEU und den Varianten von BBALG befinden sich in den Tabellen 6.6 und 6.7. Einen Überblick über die verwendeten Testdaten sowie über die Ergebnisse des kommerziellen Optimierers XOPTS gibt Abschnitt 6.1.

4.4.1 Einige Ergebnisse im Detail

Hier werden Ergebnisse für die Instanzen real_21 und real_37 genauer betrachtet und analysiert. Zur Erinnerung nochmals die Eckdaten dieser Instanzen:

real_21 6 Gruppen, 13 Elementtypen, 151 Elemente, $KUS = 3,53$, $Z_{XOPTS} = 3,78$

real_31 11 Gruppen, 45 Elementtypen, 1149 Elemente, $KUS = 22,85$, $Z_{XOPTS} = 28,62$

real_37 15 Gruppen, 23 Elementtypen, 60 Elemente, $KUS = 6,88$, $Z_{XOPTS} = 7,89$

Die genauen Daten der in diesem Anschnitt behandelten Instanzen real_21 und real_37 können in Appendix C nachgelesen werden.

Abbildung 4.2 zeigt das von der BBHEU Heuristik erzielte Ergebnis für die Instanz real_21. Das BBHEU Verfahren erreicht für real_21 einen Wert von $Z = 4,11$, das entspricht $108,73\%$ des von XOPTS erzielten Ergebnisses und $116,43\%$ der KUS.

Abbildung 4.3 zeigt das von der BBALG1 Heuristik erzielte Ergebnis für die Instanz real_21. Das BBALG1 Verfahren erreicht für real_21 einen Wert von $Z = 3,75$, das entspricht $99,21\%$ des von XOPTS erzielten Ergebnisses und $106,23\%$ der KUS.

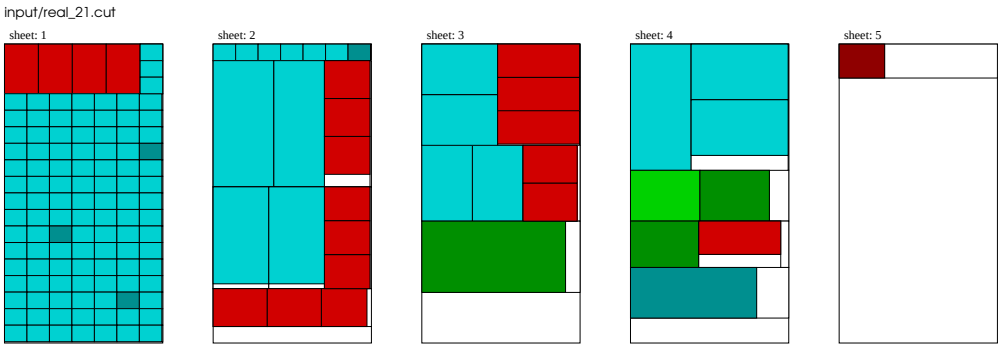


Abbildung 4.2: BBHEU_real_21

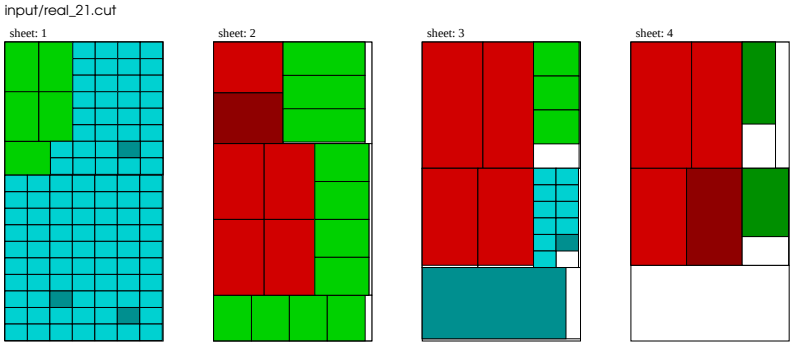


Abbildung 4.3: BBALG1_real_21

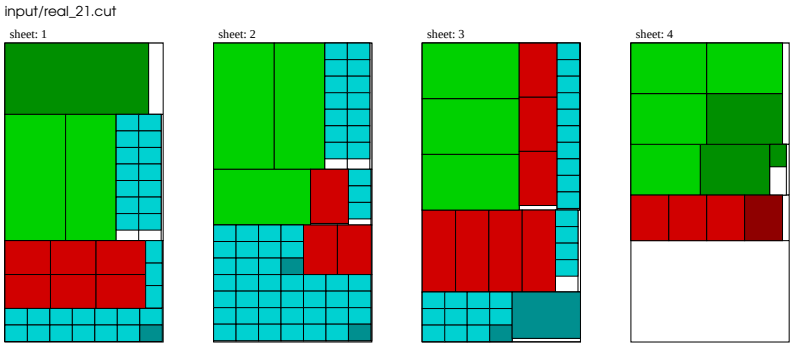


Abbildung 4.4: BBALG5_real_21

Abbildung 4.4 zeigt das von der BBALG5 Heuristik erzielte Ergebnis für die Instanz real_21. Das BBALG5 Verfahren erreicht für real_21 einen Wert von $Z = 3,66$, das entspricht $96,83\%$ des von XOPTS erzielten Ergebnisses und $103,68\%$ der KUS.

Abbildung 4.5 zeigt das von der BBHEU Heuristik erzielte Ergebnis für die Instanz real_37. Das BBHEU Verfahren erreicht für real_37 einen Wert von $Z = 9,22$, das entspricht $116,86\%$ des von XOPTS erzielten Ergebnisses und $134,01\%$ der KUS.

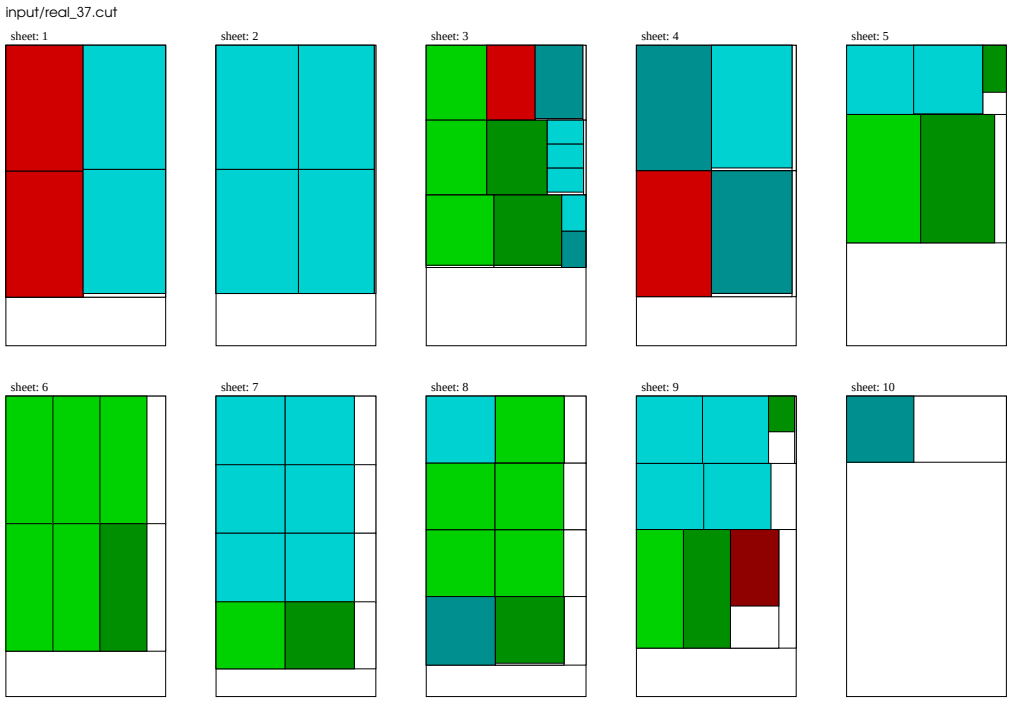


Abbildung 4.5: BBHEU_real_37

Abbildung 4.6 zeigt das von der BBALG1 Heuristik erzielte Ergebnis für die Instanz real_37. Das BBALG1 Verfahren erreicht für real_37 einen Wert von $Z = 7,88$, das entspricht $99,87\%$ des von XOPTS erzielten Ergebnisses und $114,53\%$ der KUS.

Abbildung 4.7 zeigt das von der BBALG5 Heuristik erzielte Ergebnis für die Instanz real_37. Das BBALG5 Verfahren erreicht für real_37 einen Wert von $Z = 8,22$, das entspricht $104,18\%$ des von XOPTS erzielten Ergebnisses und $119,48\%$ der KUS.

Abbildung 4.8 zeigt das von der BBALG1 Heuristik erzielte Ergebnis für die Instanz real_31. Das BBALG1 Verfahren erreicht für real_31 einen Wert von $Z = 36,62$, das entspricht $127,95\%$ des von XOPTS erzielten Ergebnisses und $160,26\%$ der KUS.

Anhand der Abbildungen 4.2 und 4.5 kann man sowohl die Stärken als auch die Schwächen des BBHEU Verfahrens erläutern. Einerseits weisen die von BBHEU erzeugten Abschnitte einen sehr geringen Verschnitt auf, andererseits werden diese schlecht angeordnet, was am strikten Einhalten der Gruppengrenzen in Phase 2 liegt. So könnte zum Beispiel das letzte Element in Abbildung 4.2 auf dem dritten Rohling untergebracht werden, es gehört zwar der roten

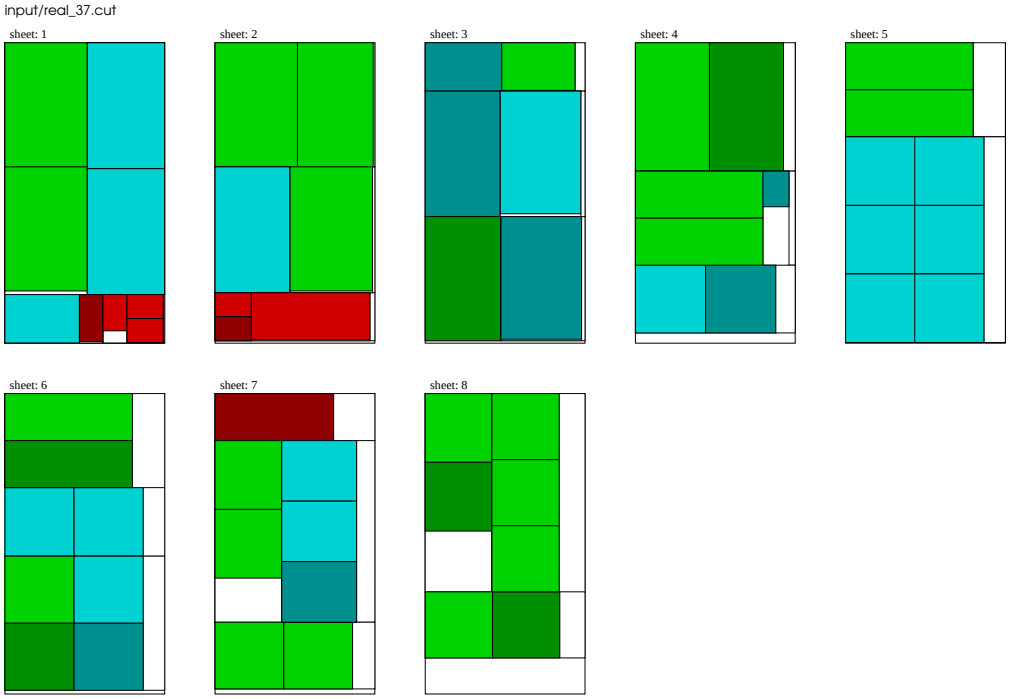


Abbildung 4.6: BBALG1_real_37

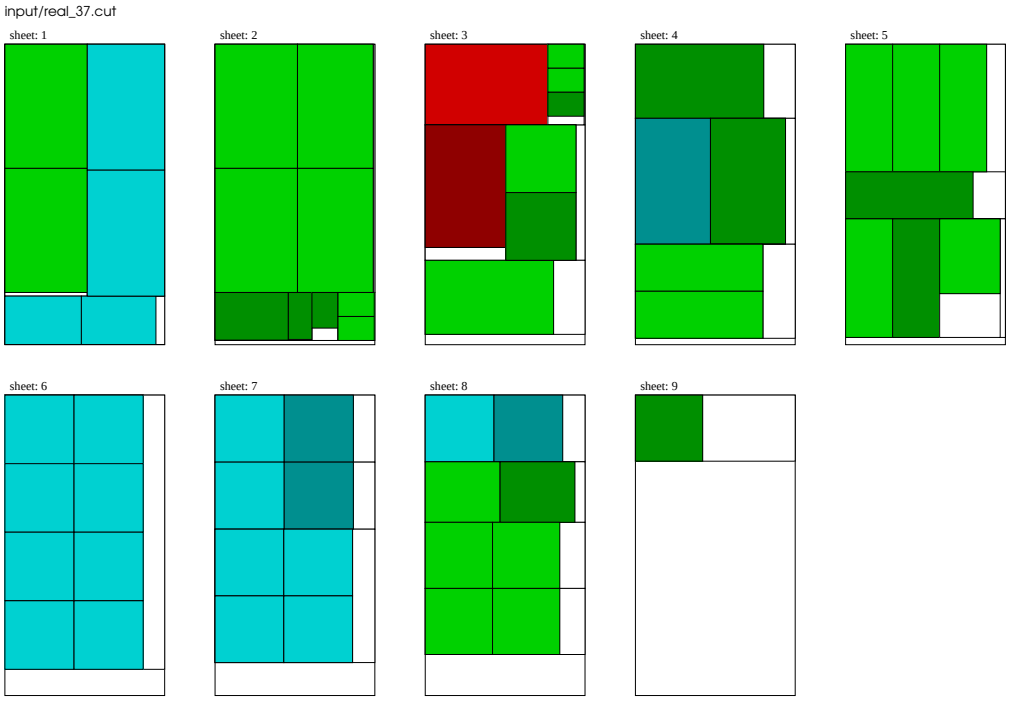


Abbildung 4.7: BBALG5_real_37

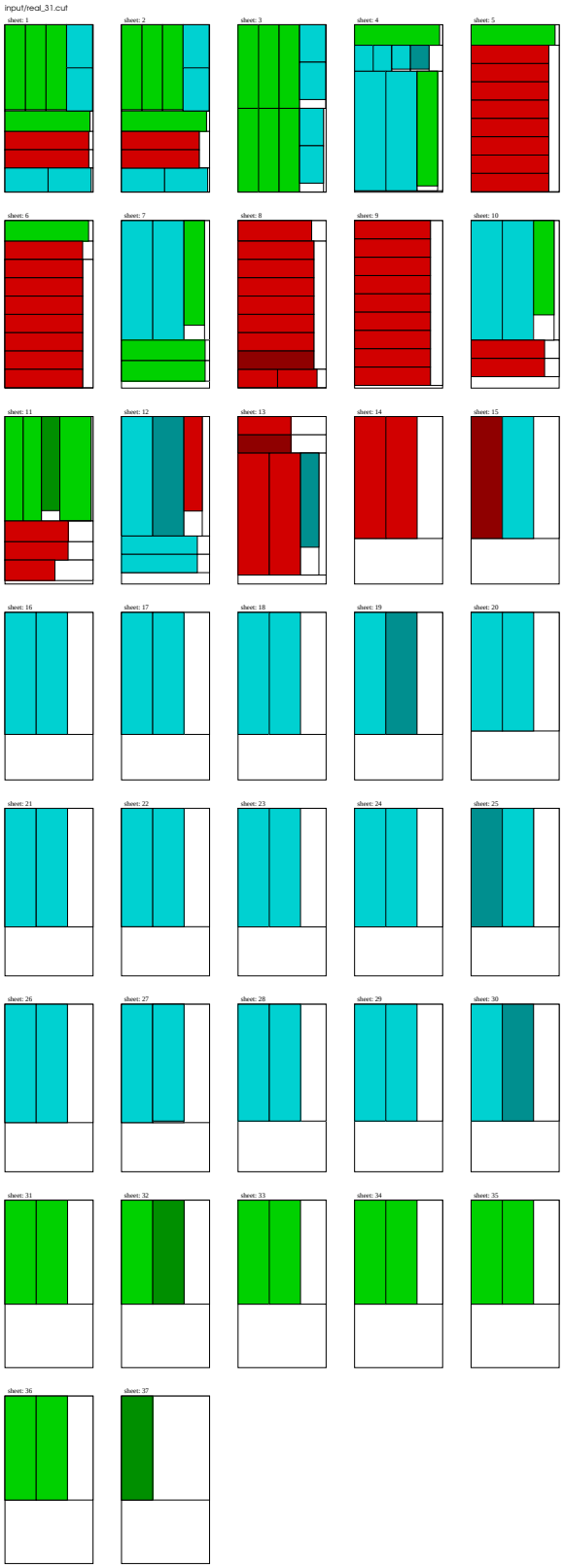


Abbildung 4.8: BBALG1_real_31

Gruppe an, die vorher schon aktiv war, aber die davor erzeugten Grenzabschnitte verhindern, dass der Abschnitt dort platziert wird.

Die Abbildungen 4.3 und 4.4 zeigen, dass BBALG wesentlich bessere Ergebnisse als BBHEU erzielen kann. Weiters wird eines der Hauptprobleme des BBALG Verfahrens deutlich: Die lokal optimale iterative Optimierung einzelner Rohlinge führt nicht unbedingt zur Optimalität des Gesamtergebnisses. So weist der durch BBALG1 erzeugte erste Rohling in Abbildung 4.3 zwar einen sehr geringen Verschnitt auf, die bessere Verteilung „großer“ und „kleiner“ Elemente durch BBALG5 in Abbildung 4.4 führt aber zu einem Gesamtergebnis, das weniger Verschnitt verursacht, obwohl bei dieser Lösung kein Rohling so dicht befüllt ist wie der erste von BBALG1 erzeugte Rohling. Diese Problematik wird in den Abbildungen 4.8 noch deutlicher. Der Vergleich der in den Abbildungen 4.6 und 4.7 dargestellten Ergebnisse zeigt, dass die erzwungene Platzierung größerer Elemente auch von Nachteil sein und das Ergebnis verschlechtern kann.

4.4.2 Schlussfolgerungen

Die Analyse der Ergebnisse von BBHEU zeigt, dass die Längen der einzelnen Abschnitte wichtig für eine erfolgreiche Optimierung eines ganzen Rohlings sind und die Optimierung der einzelnen Abschnitte keinesfalls zu guten Gesamtresultaten führen muss. Wenn man zum Beispiel den ersten Rohling der von BBHEU erzeugten Lösung in Abbildung 4.5 betrachtet, sieht man, dass nach dem letzten Abschnitt noch Platz vorhanden ist, der mit Elementen gefüllt werden könnte. Diese Elemente sind jedoch in längeren Abschnitten, zum Beispiel auf dem dritten Rohling, untergebracht und können somit nicht auf dem ersten Rohling platziert werden.

Außerdem stellt sich heraus, dass die Einschränkung der logischen Gruppen und die Einführung von Gruppengrenzen schlechte Auswirkungen auf dieses zweiphasige Verfahren hat. BBHEU könnte sich aber im Rahmen einer Metaheuristik als interessant erweisen, wenn es zusätzlich fallweise für die Erzeugung einzelner Abschnitte zum Einsatz kommt, da diese dann optimal befüllt werden können.

Mit BBALG können teilweise sehr gute Ergebnisse erzielt werden, es tritt aber ein neues Problem auf: Bei der Erzeugung der Schnittmuster für die ersten Rohlinge werden hauptsächlich die kleineren Elemente kombiniert, woraus Rohlinge mit sehr geringem Verschnitt entstehen. Die großen Elemente bleiben hingegen übrig und können nicht miteinander kombiniert werden. Dadurch entsteht viel Verschnitt: Platz, der für die zuvor verbrauchten Elemente genutzt hätte werden können. Am Beispiel der von BBALG1 erzeugten Lösungen für die Instanz real_31 (Abbildung 4.8) kann man diese Problematik deutlich erkennen. Die Einführung der BBALG Varianten hat zwar einige Verbesserungen gebracht, aber die Bevorzugung großer Elemente kann auch zu Verschlechterungen führen, wie es die im vorhergehenden Abschnitt beschriebenen Ergebnisse für die Instanz real_37 erkennen lassen.

Um die Nachteile der bis jetzt beschriebenen Verfahren zu vermeiden, wurden in weiterer Folge metaheuristische Verfahren entwickelt, die die Anzahl der Rohlinge minimieren, indem die Verteilung der Elemente auf die Rohlinge berücksichtigt und gesteuert wird.

Kapitel 5

Metaheuristiken

In diesem Kapitel werden einige Varianten eines evolutionären Algorithmus zur Lösung des 2BP-WR beschrieben. Zu Beginn werden die in weiterer Folge benötigten Grundlagen vorgestellt. Danach wird näher auf die Verfahren eingegangen, um schließlich einige Testergebnisse zu präsentieren.

5.1 Evolutionäre Algorithmen

Evolutionäre Algorithmen sind an das von Darwin beschriebene Konzept der natürlichen Selektion angelehnte Verfahren. Sie werden zur Lösung verschiedenster Optimierungsaufgaben verwendet. Eine detaillierte Einführung zu evolutionären Algorithmen gibt Michalewicz in [29], siehe weiters Bäck, Fogel und Michalewicz [1] für ein umfassendes Nachschlagwerk über evolutionäre Algorithmen.

5.1.1 Der klassische genetische Algorithmus

Der genetische Algorithmus (GA) – eine konkretere Art evolutionärer Algorithmen – ist ein von John Holland in [16] beschriebenes Modell (siehe Algorithmus 5.1). Eine Population von Individuen (d.h. Lösungen) soll über einen gewissen Zeitraum hinweg überleben und sich reproduzieren, wobei sich die besseren Individuen durchsetzen, da die schlechteren durch Selektion ausgeschieden werden.

Zu Beginn müssen die potenziellen Lösungen des Problems kodiert werden. Eine bestimmte Lösung wird als Genotyp, Individuum oder Chromosom bezeichnet. Ein Individuum besteht aus Genen. Die Repräsentation der potenziellen Lösungen als Individuen ist im Allgemeinen an das Problem angepasst. Ursprünglich wurden Lösungen häufig in Form eines binären Vektors der Länge l kodiert, wobei jedes Bit als Gen bezeichnet wird.

Die einzelnen Schritte beim Ablauf eines GA sind:

Algorithmus 5.1 Der klassische GA

BEGIN GA
$i = 0$
initialisiere($P(i)$) {$P(i)$ ist die Population der Generation i}
evaluiere($P(i)$)
while not Stop-Bedingung **do**
  $i = i + 1$
  $P(i)$ = selektiere($P(i-1)$)
  $P(i)$ = rekombiniere($P(i)$)
  $P(i)$ = mutiere($P(i)$)
  evaluiere($P(i)$)
end while
END GA

- Der erste Schritt ist die Initialisierung: Eine Anfangspopulation wird erzeugt, indem eine bestimmte Anzahl von Individuen zufällig oder auf eine andere geeignete Weise generiert wird.
- Im zweiten Schritt wird die Population mit Hilfe der Fitnessfunktion evaluiert. Die Fitnessfunktion ordnet jedem Individuum einen Fitnesswert zu.
- Im dritten Schritt, der Selektion, werden die Eltern für die nächste Generation ausgewählt. Ein klassischer Ansatz ist fitnessproportionale Selektion (auch *Roulette Wheel Selection* genannt): Jedem Individuum wird eine Selektionswahrscheinlichkeit zugeordnet, welche von seinem Fitnesswert abhängt (je höher die Fitness, desto höher die Selektionswahrscheinlichkeit).
- Mittels Rekombination werden aus den selektierten Eltern neue Individuen erzeugt. Die Rekombination erlaubt es, neue Bereiche des Suchraums zu erschließen. Üblicherweise werden aus zwei Eltern zwei Kinder erzeugt, wobei die Eltern Teilbereiche ihrer Chromsome an die Kinder vererben. Einer der einfachsten Rekombinationsoperatoren für binäre Vektoren ist das *One Point Crossover*. Dabei wird eine Zahl k zwischen 1 und $l - 1$ zufällig gewählt. Am Punkt k werden dann die beiden Eltern zerschnitten. Das erste Kind wird aus dem ersten Teil des ersten Elternteils und dem zweiten Teil des zweiten Elternteils zusammengefügt. Beim zweiten Kind ist es umgekehrt.
- Zusätzlich erfolgt noch die Mutation, bei der ein zufällig ausgewähltes Gen verändert wird. Im Falle eines binären Vektors kann zum Beispiel mit einer sehr geringen Mutationswahrscheinlichkeit ein Gen von 0 auf 1 bzw. von 1 auf 0 geändert werden.
- Die Schritte Evaluation, Selektion, Rekombination und Mutation werden über eine festgelegte Anzahl von Generationen, oder bis ein Endkriterium erfüllt ist, wiederholt.

5.1.2 EA-Varianten

Evolutionäre Algorithmen wurden zur Lösung von vielen praktischen Optimierungsproblemen eingesetzt. Dies hat zu einer Diversifizierung des Verfahrens geführt. Die meisten Varianten

von EAs bestehen aus Änderungen bzw. Erweiterungen der Konzepte des klassischen GA. Die in diesem Abschnitt vorgestellten Varianten einzelner Merkmale von EAs stellen die Grundlage für die in der Folge entwickelten Verfahren dar.

Selektion Fitnessproportionale Selektion kann zur Folge haben, dass zu großer Selektionsdruck entsteht, wenn zum Beispiel einige wenige Individuen sehr viel besser sind als alle anderen in der Population. Es würden diese wenigen dann zu häufig ausgewählt werden und rasch die Population überschwemmen. Umgekehrt kann zu geringer Selektionsdruck dazu führen, dass schlechte Individuen in der Population bleiben und diese nicht in Richtung guter Lösungen konvergiert. Durch Skalierung kann der Selektionsdruck beeinflusst werden. Weiters wurden deshalb auch andere Selektionsmethoden entwickelt:

- Bei der *rank based selection* wird den Individuen ein von der Fitness abhängiger Rang zugewiesen, und die Selektionswahrscheinlichkeit ist eine lineare Funktion des Ranges.
- Bei der *tournament selection* wird eine kleine Anzahl von Individuen zufällig gewählt, von denen dann das beste selektiert wird.

Elitismus Die Idee des Elitismus besteht darin, immer das beste Individuum in der Population zu behalten. Dabei wird üblicherweise ein Individuum der nachfolgenden Generation durch das beste der vorhergehenden Generation ersetzt. Elitismus vermeidet, dass die besten Individuen durch Rekombination und Mutation „verloren“ gehen.

Steady-State EA Bei *steady state* EAs, wie Whitleys GENITOR [34], wird pro Generation immer nur eine neue Lösung erzeugt. Meist wird dann das schlechteste Individuum durch das neue ersetzt. Es kann auch ein zufällig oder auf andere Art gewähltes Individuum ersetzt werden.

Permutationsdarstellung Bei der Lösung von kombinatorischen Optimierungsproblemen, wie zum Beispiel dem *travelling salesman problem* (TSP), bietet sich eine andere Darstellungsform (Whitley et al. [35]) an. Die Reihenfolge der zu besuchenden Städte wird als Permutation kodiert. Mit Hilfe der Permutationsdarstellung lässt sich zum Beispiel auch die Reihenfolge von zu platzierenden Elementen kodieren.
Um die Permutationsdarstellung in einem GA verwenden zu können, werden spezielle Variationsoperatoren benötigt. Als einfacher Mutationsoperator kann zum Beispiel der Zweier-Austausch (Austausch zweier zufällig gewählter Positionen) verwendet werden.

Order 3 Crossover Das *Order 3 Crossover* (OX3) von Davis [6] ist ein Rekombinationsoperator für Probleme, bei denen eine Lösung durch eine Permutation kodiert ist. Dieser Operator vererbt die Reihenfolge der Permutationen teilweise: Seien A und B die durch Selektion ausgewählten Eltern, und p, $q \in [1, l]$ (mit $p < q$, l Länge des Chromosoms) zwei Zufallszahlen. Zuerst werden die Gene von A zwischen p und q auf dieselben Positionen im ersten Kind übertragen, danach werden die übrigen Gene in der Reihenfolge von B gefüllt.

Beim zweiten Kind ist es genau umgekehrt. Ein Beispiel:
$A = (1, 4, 3, 2)$, $B = (3, 4, 2, 1)$, $p = 2$, $q = 3$
Das erste Kind ist zuerst $a = (*, 4, 3, *)$ und wird dann $a = (2, 4, 3, 1)$.
Das zweite Kind ist zuerst $b = (*, 4, 2, *)$ und wird dann $b = (1, 4, 2, 3)$.

Partially Matched Crossover Das *Partially Matched Crossover* (PMX) von Goldberg und Lingle [14] ist ebenfalls ein Rekombinationsoperator für die Permutationsdarstellung, auch dabei wird der Crossoverbereich genau übernommen. Die Positionen der restlichen Gene werden ungefähr erhalten. Seien A und B die durch Selektion ausgewählten Eltern und p, $q \in [1, l]$ (mit $p < q$, l Länge des Chromosoms) zwei Zufallszahlen. A wird vollständig in das erste Kind kopiert, danach werden für alle Gene B_i im Crossoverbereich $i \in [p, q]$ die ihnen entsprechenden Gene A_j (die $B_i = A_j$) bestimmt und A_i mit A_j vertauscht. Ein Beispiel:
$A = (1, 4, 3, 2)$, $B = (3, 4, 2, 1)$, $p = 2$, $q = 3$
Das erste Kind ist zuerst $a = (1, 4, 3, 2)$ und wird dann durch Vertauschen von a_2 mit a_2 (es findet keine Vertauschung statt, da $A_2 = B_2$) und Vertauschen von a_3 mit a_4 zu $a = (1, 4, 2, 3)$.
Das zweite Kind ist zuerst $b = (3, 4, 2, 1)$ und wird dann durch Vertauschen von b_2 mit b_2 und Vertauschen von b_3 mit b_1 zu $b = (2, 4, 3, 1)$.

Hybridisierung Mit Hilfe der Permutationsdarstellung lässt sich ein GA häufig auf einfache Art und Weise hybridisieren. Zur Dekodierung der Chromosome werden Heuristiken verwendet, die aus der vorgegebenen Reihenfolge eine korrekte Lösung erzeugen (zum Beispiel eine Platzierungsheuristik für zweidimensionales *Bin Packing*), welche dann bewertet werden kann. Es können somit auf einfache Weise komplexe Nebenbedingungen (wie zum Beispiel die Einhaltung der logischen Gruppen) berücksichtigt werden.

5.2 Entwickelte Verfahren

Die in diesem Abschnitt beschriebenen Algorithmen basieren alle auf dem Prinzip der Hybridisierung. Es wurden die in den Kapiteln 3 und 4 beschriebenen Algorithmen abgeändert und kombiniert, um als geeignete Dekodierfunktion für permutationsbasierte EAs verwendet werden zu können. Alle hier beschriebenen EAs verwenden zur Berechnung der Fitness die in Abschnitt 1.2.2 vorgestellte Zielfunktion Z.

5.2.1 EA mit Elementtyp-Repräsentation: EAet

Der in der weiteren Folge mit EAet bezeichnete evolutionäre Algorithmus besteht aus einem permutationsbasierten *steady state* EA mit einer angepassten FFF Heuristik als Dekodierfunktion.

Die Chromosome sind geordnete Vektoren, welche Verweise auf die Elementtypen E_i enthalten. Gedrehte Elementtypen werden zweimal getrennt gespeichert. Die zueinander gehörenden Elementtypen referenzieren einander, sodass die korrekte Anzahl der platzierten Elemente garantiert werden kann. Die Initialisierung der Chromosome erfolgt zufällig.

Die Dekodierfunktion platziert die Elemente wie folgt auf die Rohlinge: Unter Berücksichtigung der logischen Gruppen werden die Elemente in der durch das Chromosom vorgegebenen Reihenfolge der Elementtypen platziert. Der erste Abschnitt eines Rohlings wird mit dem Element begonnen, dessen Elementtyp im Chromosom der erste platzierbare ist, d.h. dass er aus einer erlaubten Gruppe kommt und dass noch nicht alle seine Elemente platziert wurden. Es werden dann die nächsten passenden Elemente daneben platziert und, wenn möglich, gestapelt, wobei mit den nächsten Elementen jene gemeint sind, deren Elementtypen im Chromsom die ersten platzierbaren sind.

Im Gegensatz zur FFF Heuristik können, wenn der Gesamtverschnitt des Abschnitts dadurch geringer wird, die gestapelten Elemente auch die bisherige Abschnittshöhe überschreiten, wodurch sich die Abschnittshöhe dementsprechend anpasst.

Der EAet Algorithmus wurde mit verschiedenen Konfigurationen und Operatoren getestet, welche in Abschnitt 5.3 näher erläutert werden.

Die Elementtyp-Repräsentation hat den Nachteil, dass Elemente gleichen Typs immer hintereinander platziert werden, sofern Platz vorhanden ist, was den Lösungsraum erheblich einschränkt.

5.2.2 Element-Repräsentation und neue Variationsoperatoren

5.2.2.1 Element-Repräsentation

Bei der Element-Repräsentation stellt jedes Gen ein Element dar, was die Länge der Chromosome und damit den Lösungsraum im Allgemeinen im Vergleich zur Elementtyp-Repräsentation wesentlich vergrößert. Es werden, wenn erlaubt, auch die gedrehten Varianten der Elemente gespeichert. Es kann also vorkommen, dass in einem Chromosom bis zu doppelt so viele Elemente gespeichert sind als in der Lösung aufscheinen dürfen. Dies muss in der Dekodierfunktion berücksichtigt werden, um korrekte Lösungen zu erzeugen. Die Korrektheit der Lösungen wird folgendermaßen ermöglicht: Die Gene sind Zeiger auf die Elementtypen, und diese wiederum enthalten die Anzahl der zu platzierenden Elemente, welche im Falle der Verwendung des korrespondierenden Elements um eins verringert wird. Drehbare Elementtypen sind miteinander verbunden, sodass die jeweilige Anzahl der noch zu platzierenden Elemente bei beiden (dem gedrehten und dem nichtgedrehten Elementtyp) vermindert wird.

Das Stapeln von Elementen muss im Fall der Element-Repräsentation anders gehandhabt werden als bei der Elementtyp-Repräsentation, bei der so viele Elemente des gleichen Typs wie möglich gestapelt werden. Hier werden Elemente nur dann gestapelt, wenn diese im Chromosom direkte Nachbarn sind und der Platz es zulässt. Somit kann der EA auch das Stapeln beeinflussen. Abbildung 5.1 zeigt die Auswirkung verschiedener Chromosome auf das Stapeln von Elementen durch die verwendete Dekodierfunktion.

Bei Chromosom 1 werden drei Elemente des Typs b neben einem Element vom Typ a gestapelt. Das Element vom Typ c findet noch darunter Platz. Bei Chromosom 2 hingegen werden nur zwei Elemente vom Typ b gestapelt, und Elemente der Typen c und b werden darunter platziert. Bei Chromosom 3 wird zuerst das Element vom Typ c platziert, das Element vom

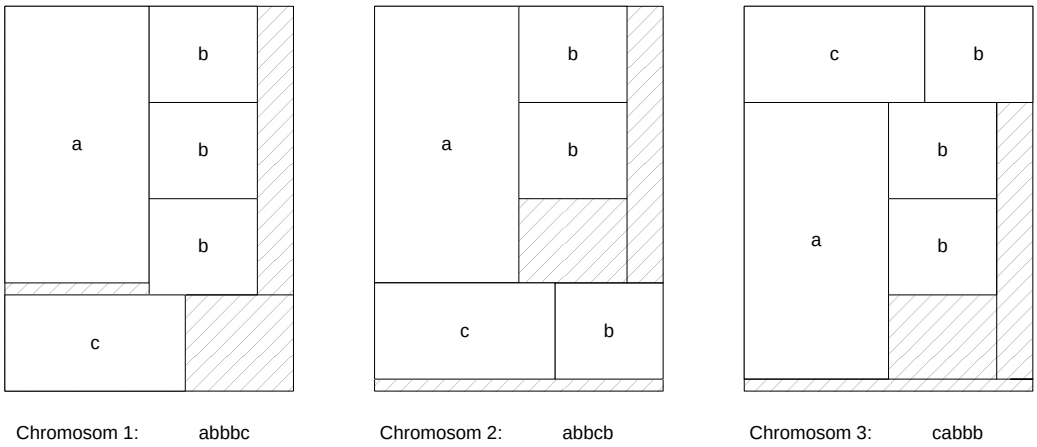


Abbildung 5.1: Auswirkung verschiedener Chromosome auf die erzeugten Schnittmuster

Typ a hat daneben keinen Platz, also wird das Element vom Typ b genommen. Dieses wird nicht gestapelt, da sich sonst der Verschnitt des Abschnitts erhöhen würde. Dann wird ein neuer Abschnitt mit dem Element vom Typ a begonnen, und daneben werden die beiden restlichen Elemente vom Typ b platziert.

5.2.2.2 Rekombinationsoperator GOX

Aufgrund der Art und Weise, wie das Übereinanderstapeln von der Dekodierfunktion gehandhabt wird, kann es von Vorteil sein, wenn Gruppen nebeneinander liegender gleicher Elemente nicht getrennt werden bzw. wieder zusammengeführt werden.

Es wurde ein abgeänderter OX3 Operator (gruppiertes OX3 GOX) entwickelt, der solche Gruppen zusammenhält. Der Pseudocode dieses GOX ist in Algorithmus 5.2 zu finden. Der GOX Operator funktioniert wie OX3. Nur wird der Crossoverbereich vor dem eigentlichen Crossover derart erweitert, dass Gruppen gleicher Elemente, welche durch die Grenzen des Crossoverbereichs getrennt würden, zur Gänze darin aufgenommen werden, indem der Crossoverbereich dementsprechend vergrößert wird.

Wenn GOX verwendet wird, werden Gruppen nebeneinanderliegender gleicher Elemente nur durch Mutation oder andere Eingriffe (siehe Abschnitt 5.2.4) getrennt.

5.2.2.3 Mutationsoperatoren

Zusätzlich zum einfachen Zweier-Austausch wurden zwei weitere Mutationsoperatoren entwickelt: Der k-Block Austausch und der Gruppierungsoperator, wobei zweiterer ein problemspezifischer Operator ist, der miteinbezieht, dass jedes Gen ein Element darstellt.

Algorithmus 5.2 Rekombinationsoperator GOX

gegeben: Eltern $parA$, $parB$ mit Länge l.
zwei Zufallszahlen p und q mit $p < q$ und $p, q < l$
$i = p - 1$
while $(i \geq 0) \wedge (parA[i] = parA[p])$ **do**
  $i = i - 1$
end while
$p = i + 1$
$i = q + 1$
while $(i < l) \wedge (parA[i] = parA[q])$ **do**
  $i = i + 1$
end while
$q = i - 1$

k-Block Austausch Es wird ein zufällig ausgewählter Block $\lfloor k \cdot l \rfloor$ nebeneinander liegender Elemente mit einem anderen, nicht überlappenden, zufällig gewählten Block $\lfloor k \cdot l \rfloor$ nebeneinander liegender Elemente vertauscht. Dabei ist $k \in [0, 1/2]$ und l die Länge des Chromosoms.

Gruppierungsoperator Es wird ein Gen g zufällig gewählt, welches für ein bestimmtes Element e_g vom Elementtyp E_g steht. Weiters wird eine durch Zufall bestimmte Anzahl (k) Gene, die Elemente des Typs E_g repräsentieren, mit Nachbargenen von g, die einen anderen Elementtyp repräsentieren, ausgetauscht. Somit wird eine Gruppe nebeneinanderliegender Gene, die Elemente desselben Elementtyps repräsentieren, erzeugt. Algorithmus 5.3 beschreibt den Gruppierungsoperator in Pseudocode. Es werden abwechselnd links und rechts vom bisher erzeugten Bereich Elemente ausgetauscht, um die gewünschte Bereichsgröße k zu erreichen.

5.2.3 EA mit Element-Repräsentation: EAe

Der Algorithmus EAe ist ein permutationsbasierter *steady state* EA mit der in Abschnitt 5.2.2 beschriebenen Element-Repräsentation. Bei der Initialisierung der Chromosome wird die Reihenfolge der Elementtypen zufällig bestimmt, aber die Gene, die Elemente desselben Elementtyps repräsentieren, werden nebeneinander ins Chromosom geschrieben. Somit werden zu Beginn Gruppen nebeneinanderliegender Gene, die denselben Elementtyp repräsentieren, erzeugt.

Dieser Algorithmus wurde mit verschiedenen Konfigurationen getestet, welche in Abschnitt 5.3 näher beschrieben und dargestellt werden.

5.2.4 EA mit Element-Repräsentation und B&B: EAebb

Der EAebb Algorithmus funktioniert wie EAe mit dem Unterschied, dass noch ein B&B dazugeschaltet wird. Mit einer gewissen Wahrscheinlichkeit werden von der Dekodierfunktion Abschnitte mittels BBHEU anstelle der Heuristik generiert. Nach der Dekodierung wird, falls

Algorithmus 5.3 Pseudocode des Gruppierungsoperators

gegeben Chromosom A der Länge l
initialisiere Zufallszahl $g \in [0, l)$ {Gen g wird ausgewählt}
initialisiere Zufallszahl $k \in [0, a(e_g))$ {Anzahl der zu gruppierenden Gene}
initialisiere Bereichsgrenzen $links = g$ und $rechts = g$
while $k > 0$ **do**
  initialisiere Index $ind = -1$
  for $i = 0$ to $i < l$ **do**
    if $k = links$ **then**
      {Der bisher behandelte Bereich wird übersprungen}
      $i = rechts$
    else if $E_g = E_i$ **then**
      {Gen i repräsentiert denselben Elementtyp wie g}
      $ind = i$
      break
    end if
  end for
  if $links = 0 \vee (k \mathrm{mod} 2 = 0 \wedge rechts < l - 1)$ **then**
    vertausche $A[rechts + 1]$ mit $A[ind]$
  else
    vertausche $A[links - 1]$ mit $A[ind]$
  end if
  $k = k - 1$
end while

BBHEU zum Einsatz kam, die Lösung in das Chromsom zurückgeschrieben. Der Einsatz des B&B Verfahrens wirkt sich somit auf die nachfolgenden Generationen aus. Das Ziel ist, lokal optimale Abschnitte in das Chromosom zu injizieren und somit die Qualität der Lösungen zu erhöhen. Es ist zu erwarten, dass die Ergebnisse in einigen Fällen verbessert werden können. Die Anzahl der mittels B&B generierten Abschnitte sollte aber nicht zu hoch sein, da sonst ähnlich negative Effekte wie bei BBHEU auftreten könnten, und vor allem auch die Laufzeit stark steigen kann.

Der Algorithmus EAebb wurde mit verschiedenen Konfigurationen getestet, welche in Abschnitt 5.3 näher beschrieben werden.

5.3 Ergebnisse und Schlussfolgerungen

Es wurden zahlreiche Varianten der verschiedenen Algorithmen getestet wobei Faktoren wie Populationsgröße, Anzahl der Generationen, Variationsoperatoren, Mutationswahrscheinlichkeiten usw. verändert wurden. Es wurden einige der Parameter fixiert, da das Ziel der verschiedenen Testreihen vor allem die Analyse der Auswirkungen der verschiedene Variationsoperatoren war. Es wurden auch andere Varianten getestet, und dabei haben sich die in weiterer Folge verwendeten Werte (siehe Tabelle 5.1) als besonders geeignet herausgestellt.

popsize	=	1000	Populationsgröße
tgen	=	1000000	Anzahl Generationen bis zum Abbruch
tcgen	=	10000	Anzahl Generationen bis zum Abbruch bei gleichbleibender Fitness
eamod	=	0	Steady-State EA
elit	=	1	Elitismus wird verwendet
repl	=	1	Das schlechteste Chromosom wird ersetzt
bbrate	=	0,001	Wahrscheinlichkeit pro Abschnitt, mit der EAebb B&B aufruft

Tabelle 5.1: Für die EA Testläufe fixierte Parameter

Die gesammelten Ergebnisse der Algorithmen EAet, EAe und EAebb befinden sich in den Tabellen 6.10, 6.11, 6.14, 6.15, 6.18 und 6.19. Einen Überblick über die verwendeten Testdaten sowie über die Ergebnisse des kommerziellen Optimierers XOPTS gibt Abschnitt 6.1.

Um die verschiedenen Varianten zu beschreiben, wurde folgende Nomenklatur festgelegt Name/CrossoverOperator/pmut/mtype (zum Beispiel EAe/OX3/0,01/1), dabei ist:

Name eines der drei Verfahren EAet, EAe oder EAebb

CrossoverOperator einer der Operatoren OX3, PMX und für EAe und EAebb auch GOX

pmut die Mutationswahrscheinlichkeit

mtype die Art der Mutation, wird nur für EAe und EAebb verwendet. Bei mtype = 0 ist es der Zweier-Austausch, bei mtype $\in (0, 1/2]$ ist es der k-Block Austausch mit k = mtype, bei mtype $= -1$ ist es der Gruppierungsoperator und bei mtype $\in [-1/2, 0)$ wird bei jeder Mutation zufällig entschieden, welcher Mutationstyp gewählt wird, wobei für den k-Block Austausch $k = |\text{mtype}|$ verwendet wird.

Es wurden weiters Tests für EAe und EAebb durchgeführt, bei welchen Lösungen der in den vorherigen Kapiteln beschriebenen Verfahren zur Anfangspopulation hinzugefügt wurden. Dies hatte jedoch keine besonderen Auswirkungen auf die erzielten Ergebnisse.

5.3.1 Einige Ergebnisse im Detail

Hier werden Ergebnisse für die Instanzen real_21 und real_37 genauer betrachtet und analysiert. Zur Erinnerung nochmals die Eckdaten dieser Instanzen:

real_21 6 Gruppen, 13 Elementtypen, 151 Elemente, $KUS = 3,53$, $Z_{XOPTS} = 3,78$

real_37 15 Gruppen, 23 Elementtypen, 60 Elemente, $KUS = 6,88$, $Z_{XOPTS} = 7,89$

Die genauen Daten der in diesem Abschnitt behandelten Instanzen real_21 und real_37 können in Appendix C nachgelesen werden. Da es sich bei den hier präsentierten Verfahren um stochastische Verfahren handelt, sind die hier gezeigten typischen Lösungen nicht eindeutig.

Abbildung 5.2 zeigt das von EAet/OX3/0,01 erzielte Ergebnis für die Instanz real_21. Das Verfahren erreicht einen Wert von $Z = 3,70$, das entspricht $97,88\%$ des von XOPTS erzielten Ergebnisses und $104,82\%$ der KUS.

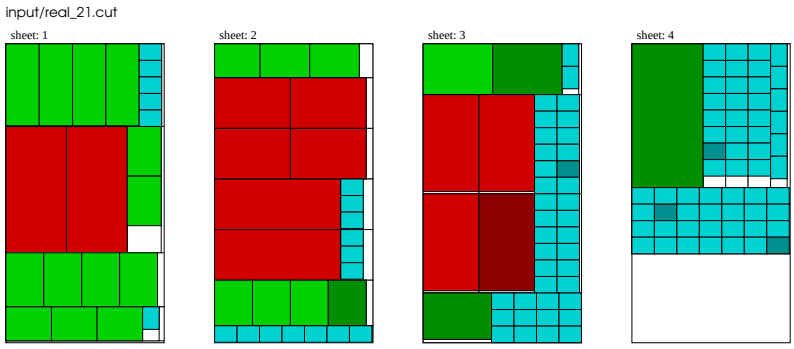


Abbildung 5.2: EAet/OX3/0,01_real_21

Abbildung 5.3 zeigt das von EAet/PMX/0,1 Heuristik erzielte Ergebnis für die Instanz real_21. Das Verfahren erreicht einen Wert von $Z = 3,68$, das entspricht $97,35\%$ des von XOPTS erzielten Ergebnisses und $104,25\%$ der KUS.

Abbildung 5.4 zeigt das von EAe/OX3/0,01/-1 erzielte Ergebnis für die Instanz real_21. Das Verfahren erreicht einen Wert von $Z = 3,70$, das entspricht $97,88\%$ des von XOPTS erzielten Ergebnisses und $104,82\%$ der KUS.

Abbildung 5.5 zeigt das von EAebb/GOX/0,01/-1 erzielte Ergebnis für die Instanz real_21. Das Verfahren erreicht einen Wert von $Z = 3,72$, das entspricht $98,41\%$ des von XOPTS erzielten Ergebnisses und $105,38\%$ der KUS.

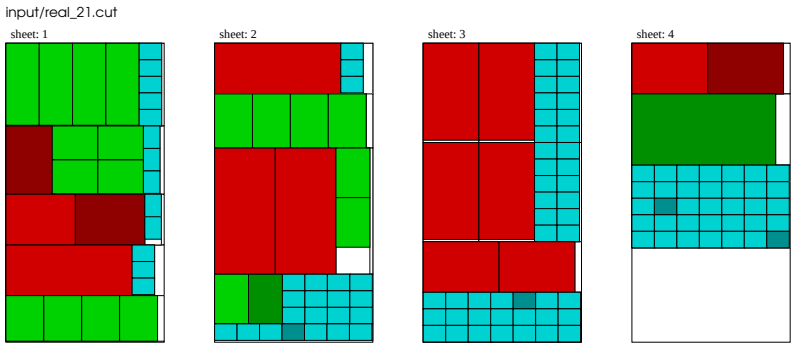


Abbildung 5.3: EAet/PMX/0,1_real_21

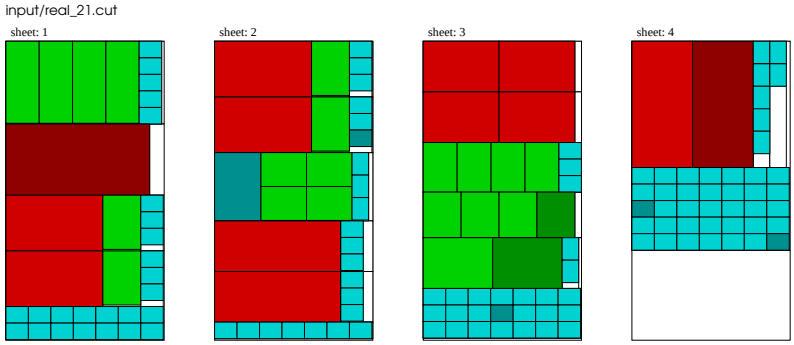


Abbildung 5.4: EAe/OX3/0,01/-1_real_21

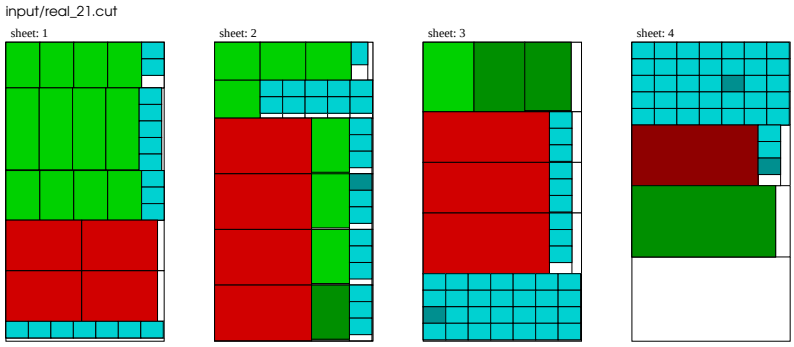


Abbildung 5.5: EAebb/GOX/0,01/-1_real_21

Die Mittelwerte der Ergebnisse von zehn Durchläufen der Varianten von EAet, EAe und EAebb liegen bei $\overline{Z} = 3,70$ bzw. $\overline{Z} = 3,71$. Auch die anderen Varianten erreichen ungefähr dieselben Ergebnisse (siehe Abschnitt 6.2.3).

Die Ergebnisse für real_21 liegen nahe zusammen, sodass man keine Aussage darüber treffen kann, welche der Varianten bessere Ergebnisse liefert. Am letzten Rohling in Abbildung 5.4 kann man die Auswirkung der in Abschnitt 5.2.2.1 beschriebenen Vorgehensweise beim Stapeln von Elementen deutlich erkennen: Die blauen Elemente werden nicht so oft wie möglich gestapelt, sondern nur so oft, wie Elemente dieses Typs nebeneinander im Chromosom stehen.

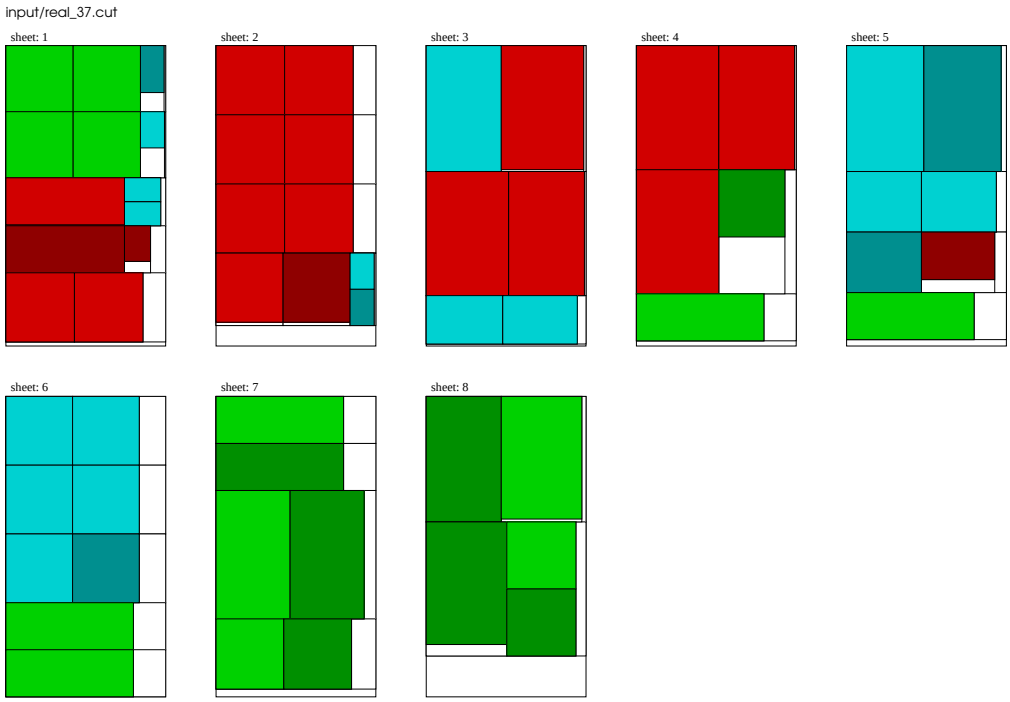


Abbildung 5.6: EAet/OX3/0,01_real_37

Abbildung 5.6 zeigt das von EAet/OX3/0,01 erzielte Ergebnis für die Instanz real_37. Das Verfahren erreicht einen Wert von $Z = 7,86$, das entspricht $99,62\%$ des von XOPTS erzielten Ergebnisses und $114,24\%$ der KUS. Im Durchschnitt über zehn Durchläufe wird ein Wert von $\overline{Z} = 7,92$ erreicht.

Abbildung 5.7 zeigt das von EAet/PMX/0,1 erzielte Ergebnis für die Instanz real_37. Das Verfahren erreicht einen Wert von $Z = 8,16$, das entspricht $103,42\%$ des von XOPTS erzielten Ergebnisses und $118,60\%$ der KUS. Im Durchschnitt über zehn Durchläufe wird ein Wert von $\overline{Z} = 8,08$ erreicht.

Die Verwendung von unterschiedlichen Rekombinationsoperatoren wirkt sich im Fall der Instanz real_37 auf die Ergebnisse von EAet aus. So liegt die Differenz der beiden abgebildeten Lösungen bei $0,3$ Rohlingen.

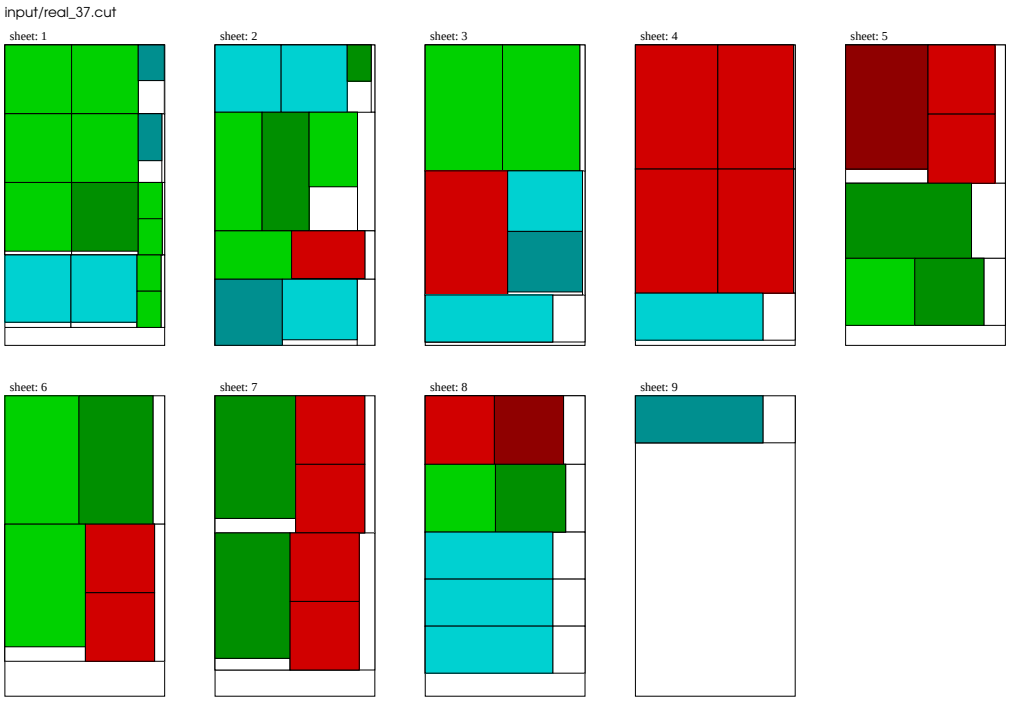


Abbildung 5.7: EAet/PMX/0,1_real_37

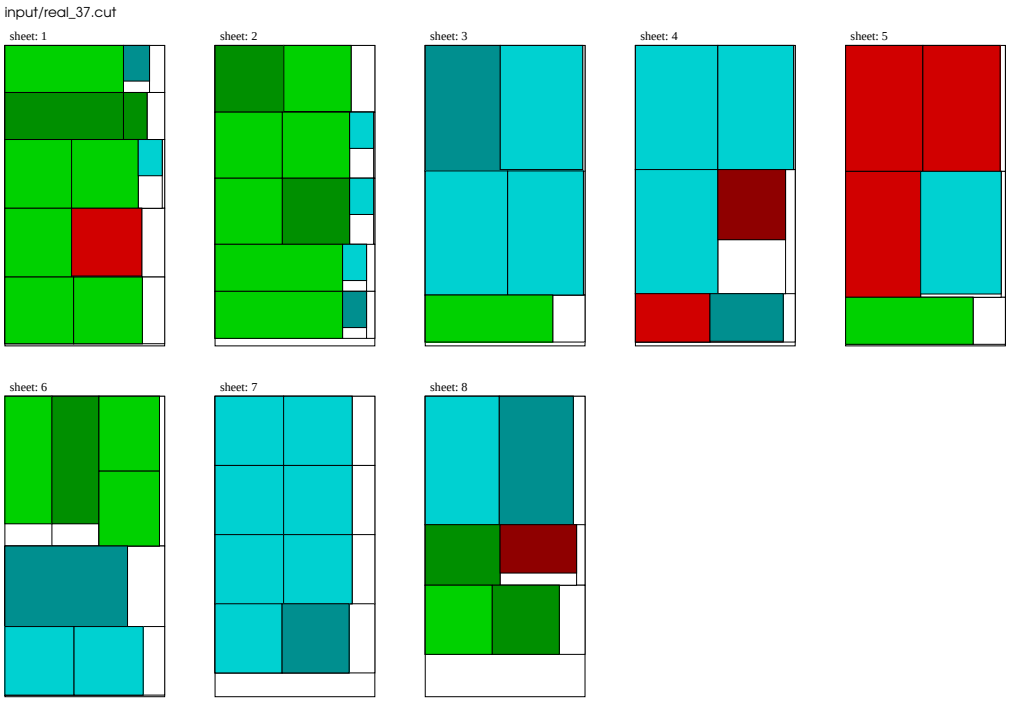


Abbildung 5.8: EAebb/OX3/0,01/-0,01_real_37

Abbildung 5.8 zeigt das von der EAebb/OX3/0,01/-0,01 Heuristik erzielte Ergebnis für die Instanz real_37. Das Verfahren erreicht hier einen Wert von $Z = 7,86$, das entspricht $99,62\%$ des von XOPTS erzielten Ergebnisses und $114,24\%$ der KUS. Im Durchschnitt über zehn Durchläufe wird ein Wert von $\overline{Z} = 8,11$ erreicht.

Im vierten Rohling der Abbildung 5.8 zeigt sich deutlich die Auswirkung der Nebenbedingungen. Obwohl im zweiten Abschnitt ein großes Element Platz finden könnte, wird das kleine rote Element verwendet, da die später verwendeten roten und blauen Elemente erst nach einem Wechsel der logischen Gruppen ausgeschnitten werden können.

Anhand der einzelnen Beispiele sind kaum Schlussfolgerungen über die Güte der verschiedenen Varianten möglich. In Abschnitt 6.2.3 werden detaillierte Ergebnisse beschrieben und ausgewertet, wobei sich die Varianten EAebb/GOX/0,01/-1 und EAebb/GOX/0,01/-0,01 als die besten herausstellen. In Appendix D sind die besten Ergebnisse für ausgewählte Instanzen dargestellt.

5.3.2 Schlussfolgerungen

Die in Abschnitt 6.2.3 dargestellten Daten zeigen, dass alle drei Algorithmen gute Ergebnisse erzielen können. Mit einigen Konfigurationen werden zumeist bessere Ergebnisse als mit XOPTS erreicht.

Der Rekombinationsoperator PMX hat sich für die im Rahmen dieser Diplomarbeit entwickelten EAs für das 2BP-WR als nicht zielführend herausgestellt, die besten Ergebnisse werden mit den Rekombinationsoperatoren OX3 und GOX erreicht. Die verschiedenen Varianten werden in Kapitel 6 ausführlich verglichen.

Überraschenderweise erreicht die einfache Variante EAet sehr gute Ergebnisse, die in den meisten Fällen an die des kommerziellen Optimierers heranreichen. Die Darstellungsform der Elementtyp-Repräsentation ist aber zu grobkörnig, um in allen Fällen sehr gute Lösungen zu produzieren.

Die Ergebnisse, die mit der wesentlich feinkörnigeren Element-Repräsentation erreicht werden, sind besser, da der Lösungsraum größer ist. Somit können Packmuster entstehen, die mit der Elementtyp-Repräsentation nicht erreichbar sind. Die erreichten Ergebnisse liegen jedoch weiter auseinander als jene von EAet, da diese durch den vergrößerten Lösungsraum mehr streuen, weiters verlängern sich im Allgemeinen auch die Laufzeiten im Vergleich zu EAet.

Die zusätzliche Verwendung von B&B verbessert die Ergebnisse weiter, wobei aber auch die Laufzeit zunimmt. Es wurde auch eine erhöhte (0,01 im Gegensatz zu 0,001) B&B Rate für einige Instanzen getestet. Dabei hat sich herausgestellt, dass sich die Ergebnisse nicht verbessern und die Laufzeit weiter ansteigt. Eine niedrige B&B Rate ist also ausreichend, um die erwünschten Verbesserungen zu erreichen.

Mit evolutionären Algorithmen können somit im Allgemeinen bessere Ergebnisse als mit dem kommerziellen Optimierer XOPTS erreicht werden, obwohl die Ergebnisse von XOPTS die speziellen Nebenbedingungen nicht berücksichtigen. Die Mittelwerte der Ergebnisse von zehn Durchläufen der besten EA Variante EAebb/GOX/0,01/-1 sind um 1,1% bessser als jene

von XOPTS. Die Ergebnisse sind nur in zwei Fällen immer schlechter als jene von XOPTS (EAebb/GOX/0,01/-1 ist bei real_43 um 0,09 Rohlinge schlechter und bei real_50 um 0,05 Rohlinge schlechter). Das könnte am heuristischen Charakter der Dekodierfunktion liegen, da in einem Fall das Ergebnis von XOPTS durch BBALG1 (Instanz real_43) erreicht wird. Verbesserungen der Dekodierfunktion oder eine andere Repräsentation könnten die Ergebnisse evolutionärer Algorithmen für das 2BP-WR eventuell noch weiter verbessern.

Kapitel 6

Ergebnisse und Vergleich der Verfahren

In diesem Kapitel werden die Verfahren anhand gegebener realer Instanzen experimentell verglichen. Die Ergebnisse werden auch in Bezug zu den von XOPTS erzielten Ergebnissen betrachtet.

6.1 Zur Verfügung stehende Daten

Die in der vorliegenden Arbeit vorgestellten Algorithmen wurden anhand von 31 realen Instanzen getestet. Zu diesen Instanzen wurden uns für Vergleichszwecke von der Firma SOGLATEC auch Lösungen zur Verfügung gestellt, die mit dem kommerziellen Optimierer XOPTS von Albat & Wirsam erzeugt wurden. Es standen leider keine näheren Angaben zur Funktionsweise von XOPTS zur Verfügung, weshalb eine stichprobenartige Überprüfung der zur Verfügung gestellten Lösungen (bei den Instanzen real_22 und real_35) duchgeführt wurde. Diese hat ergeben, dass die Nebenbedingungen der logischen Gruppen und Transportwagen von XOPTS nicht eingehalten werden. XOPTS scheint somit das reine 2BP mit Guillotineschnitten und maximaler Schnittiefe drei zu lösen. Dadurch kann XOPTS potenziell bessere, aber in Hinblick auf das 2BP-WR ungültige Schnittmuster finden. Dies sollte bei allen folgenden Vergleichen berücksichtigt werden.

Tabelle 6.1 gibt einen Überblick über die Daten sowie die von XOPTS erzielten Ergebnisse.

6.2 Ergebnisse

Die in diesem Abschnitt vorgestellten Ergebnisse wurden aus organisatorischen Gründen auf zwei verschiedenen Computern berechnet:
Alle Ergebnisse, bis auf die der EAs ohne B&B (EAet und EAe), wurden auf einem Intel Pentium 4 mit 2,8 GHz und 2GB RAM berechnet. Die Ergebnisse von EAet und EAe wurden auf

Name	Gruppen	Elementtypen	Elemente	KUS	Z_{XOPTS}
real_21	6	13	151	3,53	3,78
real_22	10	15	77	15,42	18,84
real_23	3	3	4	0,83	0,98
real_24	3	5	22	2,50	5,56
real_25	1	2	7	0,69	0,80
real_26	2	7	11	0,34	0,40
real_27	2	3	9	2,60	2,87
real_28	3	14	14	1,61	1,91
real_29	8	15	151	9,68	10,43
real_30	3	3	4	0,90	0,95
real_31	11	45	149	22,85	28,62
real_32	7	9	69	7,63	8,22
real_33	12	12	33	3,50	3,98
real_34	29	31	72	5,28	5,68
real_35	15	78	140	23,90	30,87
real_36	9	9	53	7,86	10,56
real_37	15	23	60	6,88	7,89
real_38	7	7	651	21,54	23,88
real_39	5	9	37	2,80	2,96
real_40	3	24	34	6,72	8,77
real_41	3	7	19	2,08	2,44
real_42	1	1	70	5,56	6,97
real_43	2	2	91	4,98	5,40
real_44	3	3	5	0,58	0,74
real_45	3	4	147	8,00	8,89
real_46	4	4	78	1,78	1,99
real_47	1	16	33	3,65	4,70
real_48	12	12	15	0,76	0,84
real_49	3	3	6	2,17	2,88
real_50	31	31	72	5,28	5,65
real_51	11	11	61	4,40	4,58
mittel				6,01	7,19

Tabelle 6.1: Testdaten, untere Schranke KUS und Ergebnisse von XOPTS (Z_{XOPTS})

einem Intel Pentium 3 mit 800 MHz und 1GB RAM berechnet. Dies muss bei der Betrachtung der Laufzeiten von EAet und EAe berücksichtigt werden, da diese durch die unterschiedliche Rechenleistung der Computer ungefähr um einen Faktor 3,5 länger sind.

Die Verfahren wurden in C++ implementiert mit dem GCC 3.2 Compiler unter LINUX 2.4.19 kompiliert. Die Grundlage der evolutionären Algorithmen ist die von Günther Raidl in der Abteilung Algorithmen und Datenstrukuren am Institut für Computergraphik der Technischen Universität Wien entwickelte Bibliothek für evolutionäre Algorithmen: EAlib [32].

Bei den stochastischen Verfahren (BFC und die evolutionären Algorithmen), wurden jeweils zehn Durchläufe ausgeführt, wobei in Folge Mittelwert, Standardabweichung und Minimum der Ergebnisse angegeben werden.

6.2.1 Ergebnisse von FFF und BFC

In Tabelle 6.3 sind die Ergebnisse von FFF und BFC dargestellt. In Tabelle 6.4 und Abbildung 6.1 werden diese in Relation zu den von XOPTS erzielten Ergebnissen gebracht.

Die von den *greedy* Heuristiken erzielten Ergebnisse sind zumeist schlechter als jene von XOPTS. Sie werden jedoch immer in einer im Zehntelsekundenbereich liegenden Zeit berechnet.

Die Ergebnisse von FFF sind im Schnitt um 10,54% schlechter als jene von XOPTS.

Die besten Ergebnisse der BFC Heuristik werden von Variante BFC24 erzielt. Die Mittelwerte der Ergebnisse von zehn BFC24 Durchläufen sind im Schnitt um 15% schlechter als jene von XOPTS (Tabelle 6.2). Wenn man hingegen die jeweils besten Ergebnisse der zehn Durchläufe betrachtet, sind diese im Schnitt um 7,29% schlechter als die von XOPTS.

In Tabelle 6.2 sind die Ergebnisse von FFF und BFC24 überblicksmäßig dargestellt. Dort wird unter anderem angegeben, wie oft diese Verfahren bessere, gleich gute oder schlechtere Ergebnisse als XOPTS erzielen.

Verfahren	besser	gleich	schlechter	Z/Z_{XOPTS}	Z/KUS
FFF	3	2	26	1,1054	1,3191
Mittel BFC24	1	1	29	1,1500	1,3702
Minimum BFC24	5	5	21	1,0729	1,2805

Tabelle 6.2: Überblick zu den Ergebnissen von FFF und BFC24

NAME	SC		BFC22				BFC24				BFC42				BFC44			
	Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek
real_21	4,00	0,02	3,98	0,15	3,87	0,02	4,07	0,22	3,84	0,02	4,11	0,17	3,89	0,02	4,03	0,18	3,84	0,02
real_22	26,49	0,01	25,19	0,87	23,49	0,02	25,19	0,72	23,98	0,02	24,79	0,68	23,49	0,02	25,09	0,67	23,49	0,02
real_23	1,37	0,01	1,33	0,19	0,98	0,02	1,24	0,22	0,98	0,02	1,28	0,20	0,98	0,02	1,26	0,23	0,98	0,02
real_24	5,56	0,02	5,56	0,00	5,56	0,01	5,56	0,00	5,56	0,02	5,56	0,00	5,56	0,02	5,56	0,00	5,56	0,02
real_25	0,99	0,02	0,96	0,07	0,81	0,02	0,86	0,09	0,81	0,02	0,95	0,07	0,81	0,02	0,96	0,07	0,81	0,02
real_26	0,46	0,01	0,46	0,05	0,40	0,02	0,46	0,04	0,38	0,02	0,48	0,06	0,38	0,02	0,50	0,08	0,38	0,02
real_27	2,96	0,01	3,51	0,27	2,96	0,02	3,55	0,28	2,99	0,02	3,42	0,35	2,96	0,02	3,53	0,28	2,96	0,02
real_28	2,48	0,01	2,55	0,21	1,94	0,02	2,60	0,10	2,35	0,02	2,56	0,05	2,53	0,02	2,58	0,12	2,25	0,02
real_29	10,93	0,03	11,67	0,17	11,39	0,03	11,87	0,21	11,59	0,02	11,55	0,26	11,20	0,03	11,74	0,17	11,39	0,03
real_30	0,97	0,02	0,97	0,02	0,95	0,02	0,98	0,02	0,95	0,01	0,97	0,02	0,95	0,02	0,97	0,02	0,95	0,02
real_31	32,62	0,02	37,65	1,00	35,62	0,03	37,07	1,20	34,70	0,02	36,45	1,25	34,62	0,02	36,95	1,99	31,71	0,03
real_32	8,94	0,01	9,95	0,68	9,32	0,02	9,74	0,41	9,32	0,02	9,65	0,25	9,00	0,02	9,80	0,28	9,39	0,02
real_33	4,67	0,00	4,59	0,12	4,45	0,02	4,61	0,11	4,45	0,02	4,61	0,11	4,43	0,02	4,71	0,12	4,45	0,02
real_34	5,96	0,02	6,60	0,13	6,25	0,02	6,66	0,13	6,48	0,02	6,78	0,36	6,42	0,02	6,73	0,37	6,25	0,02
real_35	32,69	0,02	37,82	0,87	35,61	0,03	37,42	1,16	35,61	0,03	38,12	1,81	33,61	0,03	37,32	0,89	35,61	0,03
real_36	10,78	0,02	10,61	0,13	10,53	0,02	10,77	0,42	10,26	0,02	10,58	0,16	10,26	0,02	10,64	0,13	10,53	0,02
real_37	8,68	0,01	8,76	0,21	8,23	0,02	8,88	0,30	8,39	0,02	8,70	0,17	8,42	0,02	8,82	0,24	8,45	0,02
real_38	23,79	0,05	25,38	0,32	24,84	0,03	25,97	0,39	25,56	0,03	25,05	0,25	24,56	0,03	25,96	0,23	25,56	0,03
real_39	3,48	0,01	4,02	0,47	3,48	0,02	3,77	0,41	3,48	0,02	3,91	0,56	3,12	0,02	3,69	0,33	3,48	0,02
real_40	8,39	0,01	8,82	0,27	8,47	0,02	8,89	0,23	8,39	0,02	8,89	0,32	8,39	0,02	8,87	0,23	8,39	0,02
real_41	2,47	0,00	2,56	0,15	2,47	0,02	2,70	0,21	2,47	0,02	2,79	0,19	2,47	0,01	2,64	0,20	2,47	0,02
real_42	8,68	0,02	8,68	0,00	8,68	0,02	8,68	0,00	8,68	0,02	8,68	0,00	8,68	0,02	8,68	0,00	8,68	0,02
real_43	5,89	0,02	6,49	0,00	6,49	0,02	6,49	0,00	6,49	0,02	5,70	0,09	5,62	0,02	5,78	0,17	5,49	0,02
real_44	0,83	0,01	0,76	0,04	0,74	0,02	0,76	0,03	0,74	0,02	0,81	0,04	0,74	0,02	0,77	0,04	0,74	0,02
real_45	9,16	0,01	9,53	0,18	9,30	0,02	9,62	0,16	9,30	0,02	9,45	0,15	9,30	0,02	9,50	0,19	9,30	0,02
real_46	2,09	0,02	2,07	0,04	2,00	0,02	2,11	0,06	2,00	0,02	2,04	0,05	2,00	0,02	2,10	0,06	2,00	0,02
real_47	4,47	0,01	4,57	0,15	4,48	0,02	4,62	0,08	4,49	0,02	4,71	0,19	4,48	0,01	4,60	0,14	4,48	0,02
real_48	0,99	0,03	1,15	0,13	0,97	0,02	1,06	0,17	0,83	0,02	1,12	0,09	0,99	0,02	1,00	0,13	0,85	0,02
real_49	2,88	0,02	3,48	0,49	2,88	0,02	3,38	0,50	2,88	0,02	3,58	0,46	2,88	0,02	3,68	0,40	2,88	0,02
real_50	5,96	0,00	6,53	0,09	6,42	0,02	6,60	0,13	6,41	0,02	6,62	0,17	6,23	0,02	6,56	0,08	6,49	0,02
real_51	5,15	0,03	5,17	0,26	5,00	0,02	5,15	0,28	4,77	0,02	5,22	0,28	4,96	0,02	5,10	0,23	4,92	0,02
mittel	7,90		8,43		8,02		8,43		8,04		8,36		7,87		8,39		7,89	

Tabelle 6.3: Ergebnisse von FFF und BFC

Name	$\frac{Z_{FFF}}{Z_{XOPTS}}$	$\frac{Z_{BFC22}}{Z_{XOPTS}}$	$\frac{Z_{BFC24}}{Z_{XOPTS}}$	$\frac{Z_{BFC42}}{Z_{XOPTS}}$	$\frac{Z_{BFC44}}{Z_{XOPTS}}$
real_21	1,0582	**1,0529**	1,0767	1,0873	1,0661
real_22	1,4061	1,3370	1,3370	**1,3158**	1,3317
real_23	1,3980	1,3571	**1,2653**	1,3061	1,2857
real_24	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_25	1,2375	1,2000	**1,0750**	1,1875	1,2000
real_26	**1,1500**	**1,1500**	**1,1500**	1,2000	1,2500
real_27	**1,0314**	1,2230	1,2369	1,1916	1,2300
real_28	**1,2984**	1,3351	1,3613	1,3403	1,3508
real_29	**1,0479**	1,1189	1,1381	1,1074	1,1256
real_30	**1,0211**	**1,0211**	1,0316	**1,0211**	**1,0211**
real_31	**1,1398**	1,3155	1,2952	1,2736	1,2911
real_32	**1,0876**	1,2105	1,1849	1,1740	1,1922
real_33	1,1734	**1,1533**	1,1583	1,1583	1,1834
real_34	**1,0493**	1,1620	1,1725	1,1937	1,1849
real_35	**1,0590**	1,2251	1,2122	1,2349	1,2089
real_36	1,0208	1,0047	1,0199	**1,0019**	1,0076
real_37	**1,1001**	1,1103	1,1255	1,1027	1,1179
real_38	**0,9962**	1,0628	1,0875	1,0490	1,0871
real_39	**1,1757**	1,3581	1,2736	1,3209	1,2466
real_40	**0,9567**	1,0057	1,0137	1,0137	1,0114
real_41	**1,0123**	1,0492	1,1066	1,1434	1,0820
real_42	**1,2453**	**1,2453**	**1,2453**	**1,2453**	**1,2453**
real_43	1,0907	1,2019	1,2019	**1,0556**	1,0704
real_44	1,1216	**1,0270**	**1,0270**	1,0946	1,0405
real_45	**1,0304**	1,0720	1,0821	1,0630	1,0686
real_46	1,0503	1,0402	1,0603	**1,0251**	1,0553
real_47	**0,9511**	0,9723	0,9830	1,0021	0,9787
real_48	**1,1786**	1,3690	1,2619	1,3333	1,1905
real_49	**1,0000**	1,2083	1,1736	1,2431	1,2778
real_50	**1,0549**	1,1558	1,1681	1,1717	1,1611
real_51	1,1245	1,1288	1,1245	1,1397	**1,1135**
mittel	**1,1054**	1,1572	1,1500	1,1547	1,1508

Tabelle 6.4: Ergebnisse von FFF und BFC in Relation zu XOPTS

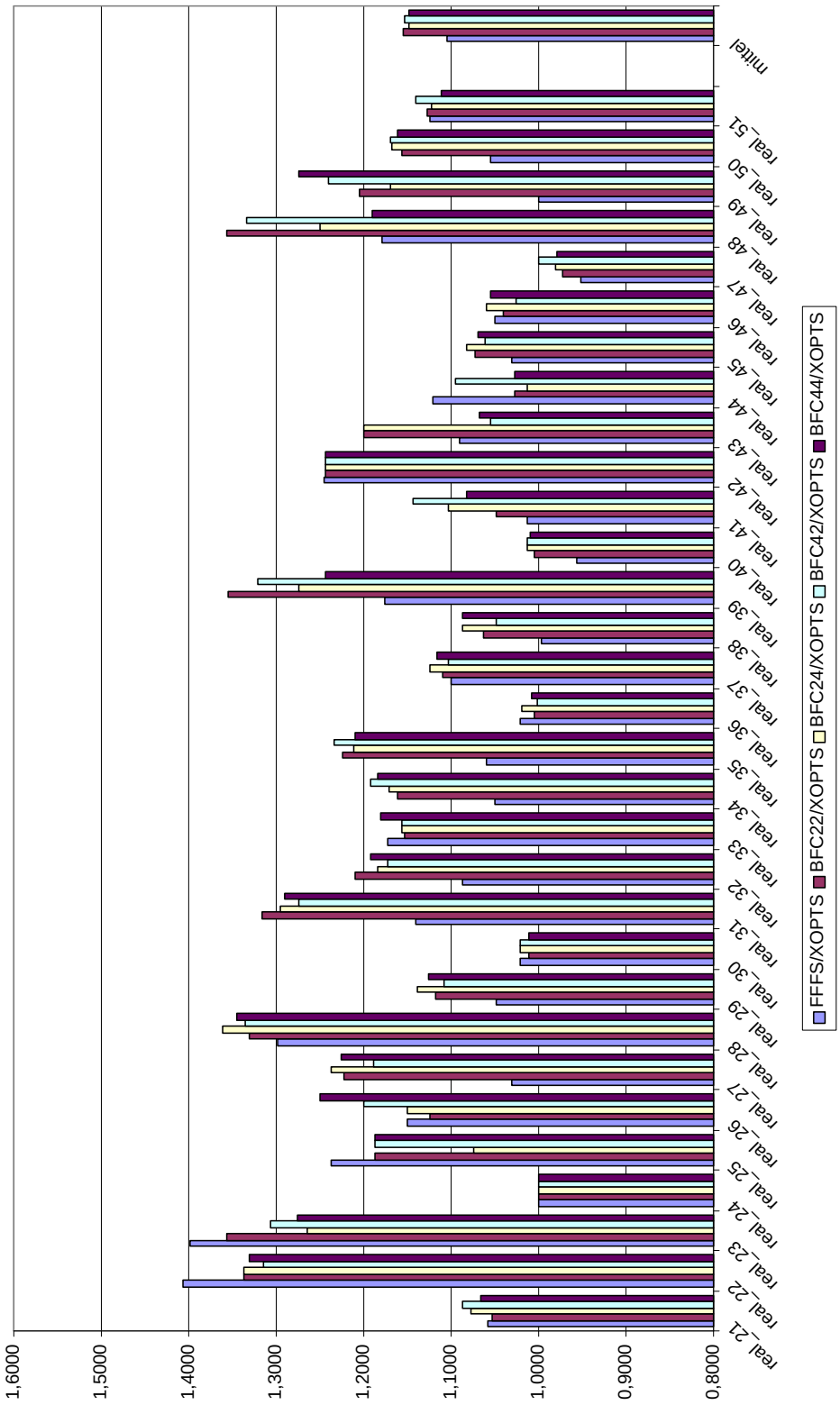


Abbildung 6.1: Graphische Darstellung der von FFF und BFC erzielten Ergebnisse in Relation zu XOPTS

6.2.2 Ergebnisse von BBHEU und BBALG

In Tabellen 6.6 und 6.7 sind die Ergebnisse von BBHEU und BBALG dargestellt. Bei den mit a gekennzeichneten Ergebnissen musste das B&B Verfahren für mindestens einen Rohling abgebrochen werden, da mehr als 3 000 000 Einträge in der Problemliste vorhanden waren (siehe auch Abschnitt 4.4). In Tabelle 6.8 und Abbildung 6.2 werden diese in Relation zu den von XOPTS erzielten Ergebnissen gebracht.

Die Ergebnisse des BBHEU Verfahrens sind zumeist schlechter als jene von XOPTS, es kommen vor allem einige Ausreißer vor, die den Mittelwert der Ergebnisse verschlechtern. Die Laufzeiten von BBHEU liegen im Sekundenbereich und erreichen einmal die Minutengrenze. Die Ergebnisse der BBALG Varianten befinden sich oft im Bereich der Ergebnisse von XOPTS, jedoch gibt es auch bei BBALG einige Ausreißer, die sich auf den Mittelwert auswirken. Die Laufzeiten der BBALG Varianten reichen von einigen Sekunden bis zu über einer Stunde.

Die Ergebnisse von BBHEU sind im Schnitt um 12,21% schlechter als jene von XOPTS (Tabelle 6.5).

Die Ergebnisse von BBALG1 sind im Schnitt um 3,03%, die von BBALG4 um 2% schlechter als jene von XOPTS (Tabelle 6.5). Bei BBALG4 wird im Gegensatz zu BBALG1 die Verwendung „großer“ Elemente erzwungen (siehe Abschnitt 4.3.2).

In Tabelle 6.5 sind die Ergebnisse von BBHEU, BBALG1 und BBALG4 überblicksmäßig dargestellt. Dort wird unter anderem angegeben, wie oft diese Verfahren bessere, gleich gute oder schlechtere Ergebnisse als XOPTS erzielen.

Verfahren	besser	gleich	schlechter	Z/Z_{XOPTS}	Z/KUS
BBHEU	3	27	21	1,1221	1,3358
BBA1	7	11	13	1,0303	1,2320
BBA4	8	10	13	1,0200	1,2177

Tabelle 6.5: Überblick zu den Ergebnissen von BBHEU, BBALG1 und BBALG4

Das Hauptproblem des BBALG Verfahrens liegt bei einigen wenigen Instanzen, für die ein schlechtes Gesamtergebnis erzielt wird (z.B.: Instanz real_35). Die Varianten von BBALG, bei denen „große“ Elemente bevorzugt werden, erreichen im Mittel zwar etwas bessere (ungefähr 1%) Ergebnisse als BBALG1, aber es können Instanzen auftreten, bei denen auch schlechtere Ergebnisse erzielt werden (siehe z.B. Abbildung 6.2 Instanzen real_28 oder real_39). Die BBALG Varianten erzielen somit Lösungen von sehr unterschiedlicher Güte.

Name	BBHEU		BBALG1			BBALG2		
	Z	Sek	Z	Sek		Z	Sek	
real_21	4,11	1,11	3,75	398,56	a	3,66	784,21	a
real_22	20,44	0,09	19,28	621,53	a	19,23	0,39	
real_23	0,98	0,01	0,98	0,01		0,98	0,01	
real_24	5,56	0,02	5,56	12,11	a	5,56	12,16	a
real_25	0,80	0,01	0,80	0,01		0,80	0,01	
real_26	0,41	0,03	0,46	15,38	a	0,46	16,94	a
real_27	3,54	0,02	2,87	0,01		2,87	0,01	
real_28	2,59	0,03	1,88	8,72		1,88	0,40	
real_29	14,55	0,26	11,34	387,25	a	11,17	415,66	a
real_30	0,95	0,01	0,95	0,01		0,95	0,01	
real_31	38,62	8,40	36,62	979,34	a	33,62	42,82	
real_32	9,54	0,04	8,54	16,66		8,61	0,83	
real_33	4,20	0,04	3,94	28,22		3,94	1,05	
real_34	6,66	1,00	5,89	209,92	a	5,96	392,58	a
real_35	37,70	70,32	36,56	127,96		36,62	2,84	
real_36	12,56	0,03	11,56	2,43		10,49	0,05	
real_37	9,22	0,10	7,88	257,48		7,88	4,57	
real_38	25,84	0,19	24,28	2173,64	a	24,28	4393,62	a
real_39	4,58	0,04	2,96	31,35		3,16	1,44	
real_40	9,39	0,04	7,66	4,47		7,78	0,08	
real_41	2,44	0,02	2,44	64,67	a	2,44	115,06	a
real_42	6,93	0,01	6,93	0,03		6,93	0,03	
real_43	6,13	0,01	5,40	2,88		5,40	0,53	
real_44	0,74	0,01	0,74	0,01		0,74	0,01	
real_45	9,59	0,02	8,89	382,09	a	8,89	262,08	a
real_46	2,00	0,18	2,00	44,96	a	2,00	57,53	a
real_47	4,66	0,15	4,24	141,53	a	4,16	6,24	
real_48	0,83	0,02	0,99	25,21	a	0,99	24,23	a
real_49	2,88	0,02	2,88	0,01		2,88	0,01	
real_50	6,66	1,00	5,89	215,79	a	6,12	365,38	a
real_51	5,57	0,08	4,88	2689,97	a	4,52	61,14	
mittel	8,41		7,71			7,58		

Tabelle 6.6: Ergebnisse von BBHEU, BBALG1 und BBALG2. Bei den mit a gekennzeichneten Ergebnissen musste das B&B Verfahren für mindestens einen Rohling abgebrochen werden.

NAME	BBALG3			BBALG4			BBALG5		
	Z	Sek		Z	Sek		Z	Sek	
real_21	3,73	124,83	a	3,71	789,65	a	3,66	10,43	
real_22	19,23	0,39		19,23	0,13		19,23	0,12	
real_23	0,98	0,01		0,98	0,01		0,98	0,01	
real_24	5,56	12,29	a	5,56	19,35	a	5,56	19,31	a
real_25	0,80	0,01		0,80	0,01		0,80	0,01	
real_26	0,38	339,59	a	0,39	14,68		0,39	1,33	
real_27	2,87	0,01		2,87	0,01		2,87	0,01	
real_28	1,88	0,10		1,96	0,02		1,99	0,01	
real_29	11,17	9,18		10,61	140,38		10,51	5,05	
real_30	0,95	0,01		0,95	0,01		0,95	0,01	
real_31	33,62	20,43		30,62	5,44		30,62	0,30	
real_32	8,75	0,29		8,85	0,51		8,75	0,19	
real_33	3,96	0,25		4,19	0,32		4,19	0,11	
real_34	5,96	228,14	a	6,12	289,38	a	6,11	71,20	a
real_35	35,62	0,71		32,69	0,02		31,62	0,03	
real_36	10,49	0,03		10,49	0,02		10,49	0,01	
real_37	7,88	0,79		7,88	1,90		8,22	0,15	
real_38	24,28	2197,71	a	24,25	1775,80	a	24,28	1122,33	a
real_39	3,58	0,25		3,58	0,20		3,58	0,01	
real_40	7,78	0,02		7,78	0,06		7,78	0,01	
real_41	2,44	28,10		2,44	153,31		2,44	5,24	
real_42	6,93	0,03		6,93	0,02		6,93	0,02	
real_43	5,40	0,08		5,40	0,37		5,40	0,05	
real_44	0,74	0,01		0,74	0,01		0,74	0,01	
real_45	8,89	244,67		8,89	55,79		8,89	5,27	
real_46	2,00	67,90	a	2,00	48,18	a	2,00	72,11	a
real_47	4,16	0,32		4,24	0,32		4,24	0,01	
real_48	0,99	26,82	a	0,99	24,14	a	0,99	52,46	a
real_49	2,88	0,01		2,88	0,01		2,88	0,01	
real_50	6,16	254,04	a	6,12	301,26	a	6,11	40,87	a
real_51	4,52	8,27		4,56	42,48		4,72	5,85	
mittel	7,57			7,38			7,35		

Tabelle 6.7: Ergebnisse von BBALG3, BBALG4 und BBALG5. Bei den mit a gekennzeichneten Ergebnissen musste das B&B Verfahren für mindestens einen Rohling abgebrochen werden.

NAME	$\frac{Z_{BBHEU}}{Z_{XOPTS}}$	$\frac{Z_{BBALG1}}{Z_{XOPTS}}$	$\frac{Z_{BBALG2}}{Z_{XOPTS}}$	$\frac{Z_{BBALG3}}{Z_{XOPTS}}$	$\frac{Z_{BBALG4}}{Z_{XOPTS}}$	$\frac{Z_{BBALG5}}{Z_{XOPTS}}$
real_21	1,0873	0,9921	**0,9683**	0,9868	0,9815	**0,9683**
real_22	1,0849	1,0234	**1,0207**	**1,0207**	**1,0207**	**1,0207**
real_23	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_24	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_25	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_26	1,0250	1,1500	1,1500	**0,9500**	0,9750	0,9750
real_27	1,2334	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_28	1,3560	**0,9843**	**0,9843**	**0,9843**	1,0262	1,0419
real_29	1,3950	1,0872	1,0709	1,0709	1,0173	**1,0077**
real_30	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_31	1,3494	1,2795	1,1747	1,1747	**1,0699**	**1,0699**
real_32	1,1606	**1,0389**	1,0474	1,0645	1,0766	1,0645
real_33	1,0553	**0,9899**	**0,9899**	0,9950	1,0528	1,0528
real_34	1,1725	**1,0370**	1,0493	1,0493	1,0775	1,0757
real_35	1,2213	1,1843	1,1863	1,1539	1,0590	**1,0243**
real_36	1,1894	1,0947	**0,9934**	**0,9934**	**0,9934**	**0,9934**
real_37	1,1686	**0,9987**	**0,9987**	**0,9987**	**0,9987**	1,0418
real_38	1,0821	1,0168	1,0168	1,0168	1,0155	**1,0168**
real_39	1,5473	**1,0000**	1,0676	1,2095	1,2095	1,2095
real_40	1,0707	**0,8734**	0,8871	0,8871	0,8871	0,8871
real_41	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_42	**0,9943**	**0,9943**	**0,9943**	**0,9943**	**0,9943**	**0,9943**
real_43	1,1352	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_44	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_45	1,0787	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_46	**1,0050**	**1,0050**	**1,0050**	**1,0050**	**1,0050**	**1,0050**
real_47	0,9915	**0,9021**	**0,9021**	**0,9021**	**0,9021**	**0,9021**
real_48	**0,9881**	1,1786	1,1786	1,1786	1,1786	1,1786
real_49	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_50	1,1788	**1,0425**	1,0832	1,0903	1,0832	1,0814
real_51	1,2162	1,0655	**0,9869**	**0,9869**	0,9956	1,0306
mittel	1,1221	1,0303	1,0238	1,0224	**1,0200**	1,0207

Tabelle 6.8: Ergebnisse von BBHEU und BBALG in Relation zu XOPTS

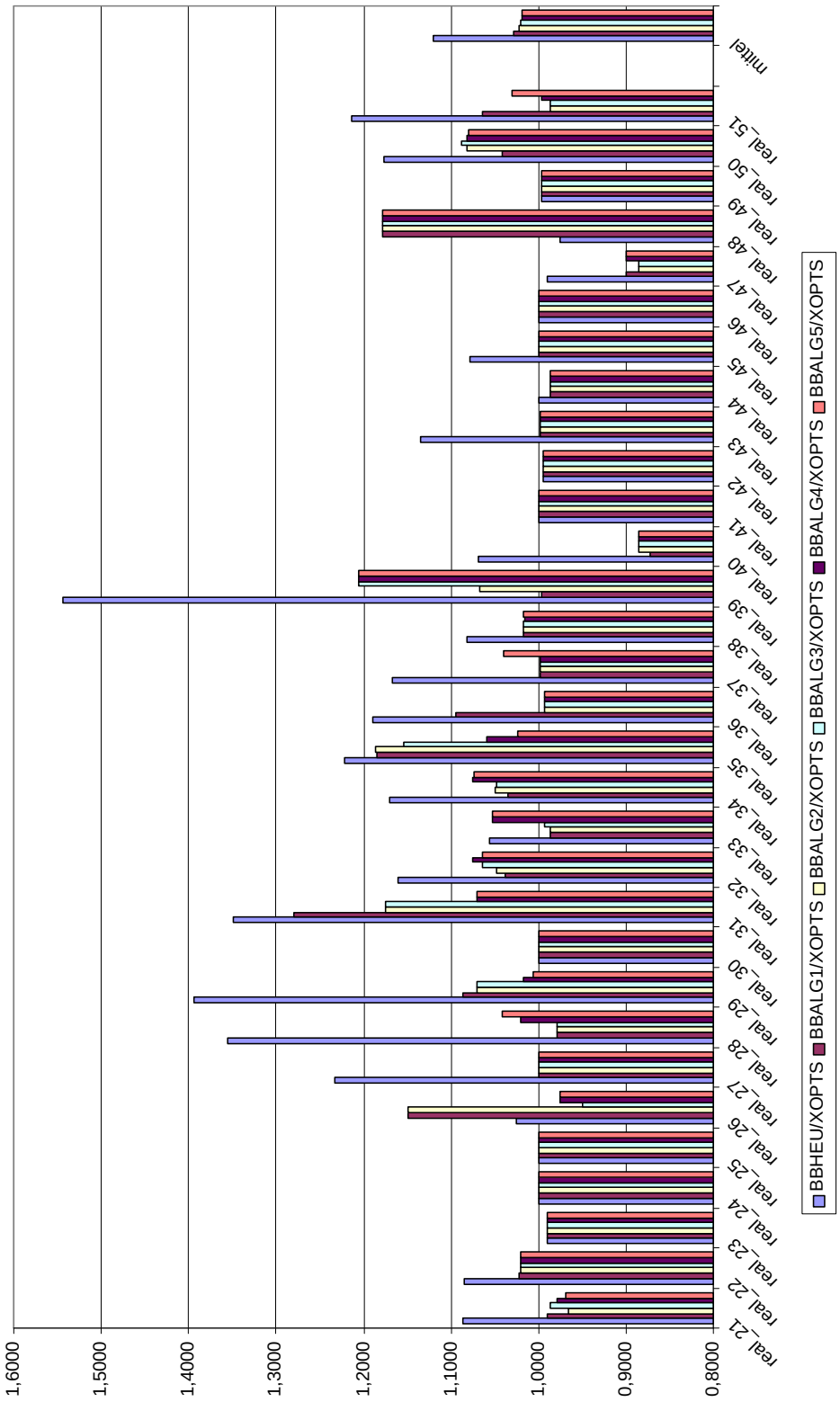


Abbildung 6.2: Graphische Darstellung der von BBHEU und BBALG erzielten Ergebnisse in Relation zu XOPTS

6.2.3 Ergebnisse von EAet, EAe und EAebb

6.2.3.1 EA mit Elementtyp-Repräsentation: EAet

In Tabellen 6.10 und 6.11 sind die Ergebnisse von EAet dargestellt. In Tabelle 6.12 und Abbildung 6.3 werden diese in Relation zu den von XOPTS erzielten Ergebnissen gebracht.

Die Ergebnisse der EAet Varianten übertreffen im Mittel die von XOPTS erzielten Ergebnisse. Die Laufzeiten liegen im Sekundenbereich.

Die besten Ergebnisse der EAet Varianten werden mit EAet/OX3/0,01 erzielt. Die durchschnittlichen Ergebnisse von zehn EAet/OX3/0,01 Durchläufen sind um 0,69% besser als jene von XOPTS (Tabelle 6.9). Wenn man hingegen die jeweils besten Ergebnisse der zehn Durchläufe betrachtet, erzeugen diese sogar um 1,12% weniger Verschnitt als die von XOPTS.

In Tabelle 6.9 sind die Ergebnisse von EAet überblicksmäßig dargestellt. Dort wird unter anderem angegeben, wie oft dieses Verfahren bessere, gleich gute oder schlechtere Ergebnisse als XOPTS erzielt.

Verfahren	besser	gleich	schlechter	Z/Z_{XOPTS}	Z/KUS
Mittel EAet/PMX/0,01	11	7	13	0,9957	1,1903
Min EAet/PMX/0,01	14	8	9	0,9882	1,1814
Mittel EAet/OX3/0,01	11	7	13	0,9931	1,1872
Min EAet/OX3/0,01	15	8	8	0,9867	1,1765

Tabelle 6.9: Überblick zu den Ergebnissen von EAet

Die mit dem PMX Rekombinationsoperator erzielten Ergebnisse, sind eindeutig schlechter als jene, die mit OX3 erzielt werden. Auch eine zu hohe Mutationsrate kann die Ergebnisse negativ beeinflussen. Alle Varianten von EAet übertreffen im Mittel die bisher getesteten Verfahren sowie XOPTS. Die Ergebnisse der verschiedenen Varianten liegen zumeist sehr eng zusammen (Abbildung 6.3). Bei einigen „schwierigen“ Instanzen zeigen sich aber die Vorteile von OX3 (z.B.: Instanzen real_35 oder real_50).

NAME	EAet/OX3/0,01				EAet/OX3/0,1				EAet/OX3/1			
	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek
real_21	3,70	0,01	3,70	10,78	3,70	0,01	3,68	9,77	3,71	0,01	3,70	11,32
real_22	18,53	0,00	18,53	8,33	18,53	0,00	18,53	8,18	18,53	0,00	18,53	7,63
real_23	0,98	0,00	0,98	1,45	0,98	0,00	0,98	1,45	0,98	0,00	0,98	1,45
real_24	5,56	0,00	5,56	6,77	5,56	0,00	5,56	6,53	5,56	0,00	5,56	7,18
real_25	0,81	0,00	0,81	1,62	0,81	0,00	0,81	1,61	0,81	0,00	0,81	1,65
real_26	0,38	0,00	0,38	2,05	0,38	0,00	0,38	1,91	0,38	0,00	0,38	1,44
real_27	2,87	0,00	2,87	1,40	2,87	0,00	2,87	1,42	2,87	0,00	2,87	1,50
real_28	1,85	0,01	1,84	4,20	1,86	0,02	1,84	4,26	1,85	0,01	1,84	2,55
real_29	10,64	0,03	10,59	13,43	10,63	0,03	10,59	15,66	10,63	0,02	10,59	17,05
real_30	0,95	0,00	0,95	1,37	0,95	0,00	0,95	1,35	0,95	0,00	0,95	1,32
real_31	28,63	0,02	28,62	24,80	28,63	0,02	28,62	23,00	28,63	0,02	28,62	25,06
real_32	8,39	0,00	8,39	7,56	8,39	0,00	8,39	7,66	8,40	0,02	8,39	6,28
real_33	3,99	0,07	3,96	7,29	4,02	0,08	3,96	5,15	4,01	0,08	3,96	5,08
real_34	5,75	0,07	5,65	22,23	5,72	0,03	5,67	21,44	5,70	0,03	5,65	24,59
real_35	31,60	1,09	29,61	50,55	31,59	0,64	30,55	53,21	31,82	0,40	31,61	47,88
real_36	10,24	0,00	10,24	5,44	10,24	0,00	10,24	5,23	10,24	0,00	10,24	4,54
real_37	7,92	0,13	7,81	17,09	8,05	0,14	7,84	13,38	8,10	0,14	7,85	10,62
real_38	23,56	0,00	23,56	23,07	23,56	0,00	23,56	22,47	23,56	0,00	23,56	21,40
real_39	2,97	0,00	2,97	5,41	2,97	0,00	2,97	5,42	2,97	0,00	2,97	3,69
real_40	7,70	0,06	7,66	9,79	7,66	0,00	7,66	12,47	7,68	0,05	7,66	9,47
real_41	2,44	0,00	2,44	4,48	2,44	0,00	2,44	4,45	2,44	0,00	2,44	3,53
real_42	6,93	0,00	6,93	4,14	6,93	0,00	6,93	4,14	6,93	0,00	6,93	4,13
real_43	5,66	0,00	5,66	3,39	5,66	0,00	5,66	3,38	5,66	0,00	5,66	3,40
real_44	0,74	0,00	0,74	1,21	0,74	0,00	0,74	1,20	0,74	0,00	0,74	1,21
real_45	9,15	0,00	9,15	8,46	9,15	0,00	9,15	8,25	9,15	0,00	9,15	7,19
real_46	1,98	0,00	1,98	5,76	1,98	0,00	1,98	5,59	1,98	0,00	1,98	5,42
real_47	4,16	0,00	4,16	5,76	4,17	0,01	4,16	4,44	4,17	0,01	4,16	4,42
real_48	0,83	0,00	0,83	1,78	0,83	0,00	0,83	1,81	0,83	0,00	0,83	1,38
real_49	2,88	0,00	2,88	2,36	2,88	0,00	2,88	2,36	2,88	0,00	2,88	2,45
real_50	5,76	0,08	5,68	19,41	5,71	0,03	5,65	23,55	5,74	0,03	5,70	16,59
real_51	4,64	0,00	4,64	5,80	4,64	0,00	4,64	4,94	4,64	0,01	4,62	4,64
mittel	7,17		7,09		7,17		7,12		7,18		7,16	

Tabelle 6.10: Ergebnisse von EAet mit OX3

NAME	EAet/PMX/0,01				EAet/PMX/0,1				EAet/PMX/1			
	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek
real_21	3,71	0,01	3,68	11,44	3,70	0,01	3,68	11,45	3,71	0,01	3,68	9,88
real_22	18,53	0,00	18,53	7,67	18,53	0,00	18,53	8,02	18,53	0,00	18,53	7,78
real_23	0,98	0,00	0,98	1,46	0,98	0,00	0,98	1,45	0,98	0,00	0,98	1,45
real_24	5,56	0,00	5,56	7,17	5,56	0,00	5,56	6,84	5,56	0,00	5,56	7,01
real_25	0,81	0,00	0,81	1,61	0,81	0,00	0,81	1,62	0,81	0,00	0,81	1,64
real_26	0,38	0,00	0,38	1,64	0,38	0,00	0,38	1,77	0,38	0,00	0,38	1,32
real_27	2,87	0,00	2,87	1,41	2,87	0,00	2,87	1,42	2,87	0,00	2,87	1,50
real_28	1,84	0,00	1,84	3,38	1,85	0,02	1,84	2,97	1,84	0,00	1,84	2,48
real_29	10,64	0,02	10,62	18,28	10,64	0,02	10,61	15,87	10,65	0,02	10,62	14,65
real_30	0,95	0,00	0,95	1,33	0,95	0,00	0,95	1,36	0,95	0,00	0,95	1,33
real_31	28,75	0,29	28,62	28,48	28,64	0,03	28,62	28,23	28,65	0,04	28,62	31,18
real_32	8,40	0,02	8,39	6,81	8,39	0,00	8,39	6,87	8,39	0,00	8,39	6,15
real_33	4,03	0,08	3,96	5,20	4,01	0,07	3,96	4,67	4,06	0,08	3,96	3,34
real_34	5,85	0,06	5,74	13,81	5,83	0,08	5,71	16,84	5,85	0,09	5,65	13,25
real_35	32,42	1,18	30,52	49,96	32,71	0,94	31,61	38,07	33,01	0,48	32,55	39,48
real_36	10,24	0,00	10,24	4,62	10,24	0,00	10,24	4,59	10,24	0,00	10,24	4,04
real_37	8,02	0,16	7,85	15,17	8,08	0,15	7,85	10,38	8,15	0,09	7,96	9,71
real_38	23,56	0,00	23,56	21,58	23,56	0,00	23,56	21,58	23,56	0,00	23,56	21,40
real_39	2,97	0,00	2,97	4,36	2,97	0,00	2,97	4,42	2,97	0,00	2,97	3,60
real_40	7,72	0,05	7,66	9,08	7,72	0,06	7,66	8,06	7,69	0,04	7,66	9,84
real_41	2,44	0,00	2,44	3,83	2,44	0,00	2,44	3,78	2,44	0,00	2,44	3,08
real_42	6,93	0,00	6,93	4,10	6,93	0,00	6,93	4,13	6,93	0,00	6,93	4,12
real_43	5,66	0,00	5,66	3,37	5,66	0,00	5,66	3,38	5,66	0,00	5,66	3,39
real_44	0,74	0,00	0,74	1,21	0,74	0,00	0,74	1,18	0,74	0,00	0,74	1,21
real_45	9,15	0,00	9,15	7,78	9,15	0,00	9,15	7,70	9,15	0,00	9,15	7,11
real_46	1,98	0,00	1,98	6,00	1,98	0,00	1,98	5,89	1,98	0,00	1,98	5,53
real_47	4,19	0,03	4,16	4,70	4,17	0,01	4,16	5,31	4,18	0,02	4,16	4,21
real_48	0,83	0,00	0,83	1,56	0,83	0,00	0,83	1,43	0,83	0,00	0,83	1,20
real_49	2,88	0,00	2,88	2,36	2,88	0,00	2,88	2,41	2,88	0,00	2,88	2,47
real_50	5,77	0,07	5,70	21,59	5,84	0,05	5,76	13,74	5,86	0,05	5,74	12,20
real_51	4,64	0,01	4,62	6,57	4,64	0,00	4,64	7,02	4,64	0,01	4,60	5,16
mittel	7,21		7,12		7,22		7,16		7,23		7,19	

Tabelle 6.11: Ergebnisse von EAet mit PMX

NAME	$\frac{Z_{EAet/OX3/0,01}}{Z_{XOPTS}}$	$\frac{Z_{EAet/OX3/0,1}}{Z_{XOPTS}}$	$\frac{Z_{EAet/OX3/1}}{Z_{XOPTS}}$	$\frac{Z_{EAet/PMX/0,01}}{Z_{XOPTS}}$	$\frac{Z_{EAet/PMX/0,1}}{Z_{XOPTS}}$	$\frac{Z_{EAet/PMX/1}}{Z_{XOPTS}}$
real_21	**0,9788**	**0,9788**	0,9815	0,9815	**0,9788**	0,9815
real_22	**0,9835**	**0,9835**	**0,9835**	**0,9835**	**0,9835**	**0,9835**
real_23	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_24	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_25	**1,0125**	**1,0125**	**1,0125**	**1,0125**	**1,0125**	**1,0125**
real_26	**0,9500**	**0,9500**	**0,9500**	**0,9500**	**0,9500**	**0,9500**
real_27	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_28	0,9686	0,9738	0,9686	**0,9634**	0,9686	**0,9634**
real_29	1,0201	**1,0192**	**1,0192**	1,0201	1,0201	1,0211
real_30	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_31	**1,0003**	**1,0003**	**1,0003**	1,0045	1,0007	1,0010
real_32	**1,0207**	**1,0207**	1,0219	1,0219	**1,0207**	**1,0207**
real_33	**1,0025**	1,0101	1,0075	1,0126	1,0075	1,0201
real_34	1,0123	1,0070	**1,0035**	1,0299	1,0264	1,0299
real_35	1,0236	**1,0233**	1,0308	1,0502	1,0596	1,0693
real_36	**0,9697**	**0,9697**	**0,9697**	**0,9697**	**0,9697**	**0,9697**
real_37	**1,0038**	1,0203	1,0266	1,0165	1,0241	1,0330
real_38	**0,9866**	**0,9866**	**0,9866**	**0,9866**	**0,9866**	**0,9866**
real_39	**1,0034**	**1,0034**	**1,0034**	**1,0034**	**1,0034**	**1,0034**
real_40	0,8780	**0,8734**	0,8757	0,8803	0,8803	0,8769
real_41	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_42	**0,9943**	**0,9943**	**0,9943**	**0,9943**	**0,9943**	**0,9943**
real_43	**1,0481**	**1,0481**	**1,0481**	**1,0481**	**1,0481**	**1,0481**
real_44	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_45	**1,0292**	**1,0292**	**1,0292**	**1,0292**	**1,0292**	**1,0292**
real_46	**0,9950**	**0,9950**	**0,9950**	**0,9950**	**0,9950**	**0,9950**
real_47	**0,8851**	0,8872	0,8872	0,8915	0,8872	0,8894
real_48	**0,9881**	**0,9881**	**0,9881**	**0,9881**	**0,9881**	**0,9881**
real_49	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_50	1,0195	**1,0106**	1,0159	1,0212	1,0336	1,0372
real_51	**1,0131**	**1,0131**	**1,0131**	**1,0131**	**1,0131**	**1,0131**
mittel	**0,9931**	0,9935	0,9939	0,9957	0,9962	0,9973

Tabelle 6.12: Ergebnisse von EAet in Relation zu XOPTS

Abbildung 6.3: Graphische Darstellung der von EAet erzielten Ergebnisse in Relation zu XOPTS

6.2.3.2 EA mit Element-Repräsentation: EAe

In Tabellen 6.14 und 6.15 sind die Ergebnisse von EAe dargestellt. In Tabelle 6.16 und Abbildung 6.4 werden diese in Relation zu den von XOPTS erzielten Ergebnissen gebracht.

Die Ergebnisse der EAe Varianten übertreffen im Mittel die von XOPTS und EAet erzielten Resultate. Die Laufzeiten liegen im Sekunden- bis Minutenbereich.

Die besten Ergebnisse der EAe Varianten werden mit EAe/GOX/0,01/0 erzielt. Die durchschnittlichen Ergebnisse von zehn EAe/GOX/0,01/0 Durchläufen sind um 0,91% besser als jene von XOPTS (Tabelle 6.13). Wenn man hingegen die jeweils besten Ergebnisse der zehn Durchläufe betrachtet, erzeugen diese sogar 1,42% weniger Verschnitt als die von XOPTS.

In Tabelle 6.13 sind die Ergebnisse von EAe überblicksmäßig dargestellt. Dort wird unter anderem angegeben, wie oft dieses Verfahren bessere, gleich gute oder schlechtere Ergebnisse als XOPTS erzielt.

Verfahren	besser	gleich	schlechter	Z/Z_{XOPTS}	Z/KUS
Mittel EAe/OX3/0,01/0	12	8	11	0,9927	1,1870
Min EAe/OX3/0,01/0	15	11	5	0,9876	1,1811
Mittel EAe/GOX/0,01/0	12	9	10	0,9909	1,1848
Min EAe/GOX/0,01/0	17	10	4	0,9858	1,1790

Tabelle 6.13: Überblick zu den Ergebnissen von EAe

Die mit GOX erzielten Ergebnisse sind besser als jene, die mit OX3 erreicht werden. Die Idee, Blöcke von Genen, die Elemente desselben Elementtyps repräsentieren, zusammenzuhalten, erweist sich als sinnvoll. Die Unterschiede zwischen den GOX Varianten sind gering und der Unterschied zwischen Zweier-Austausch und Gruppierungsoperator liegt bei 0,1%. Wenn jedoch der OX3 Operator verwendet wird, liegen die Ergebnisse mit dem Gruppierungsoperator als Mutation besser, wodurch der Eindruck bestätigt wird, dass das Zusammenhalten von Gen-Blöcken Vorteile bringt.

NAME	EAe/OX3/0,01/0				EAe/OX3/0,01/-1				EAe/OX3/0,01/-0,01			
	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek
real_21	3,72	0,01	3,69	13,01	3,71	0,01	3,68	13,03	3,72	0,01	3,70	11,85
real_22	18,53	0,00	18,53	7,74	18,53	0,00	18,53	7,15	18,53	0,00	18,53	7,25
real_23	0,98	0,00	0,98	1,98	0,98	0,00	0,98	2,13	0,98	0,00	0,98	2,11
real_24	5,56	0,00	5,56	4,73	5,56	0,00	5,56	4,77	5,56	0,00	5,56	4,75
real_25	0,80	0,00	0,80	1,24	0,80	0,00	0,80	1,26	0,80	0,00	0,80	1,30
real_26	0,38	0,00	0,38	1,22	0,38	0,00	0,38	1,28	0,38	0,00	0,38	1,25
real_27	2,87	0,00	2,87	1,35	2,87	0,00	2,87	1,34	2,87	0,00	2,87	1,33
real_28	1,85	0,01	1,84	1,52	1,87	0,02	1,84	2,07	1,86	0,02	1,84	2,67
real_29	10,52	0,04	10,47	29,56	10,53	0,03	10,49	24,00	10,52	0,05	10,47	25,94
real_30	0,95	0,00	0,95	2,03	0,95	0,00	0,95	2,23	0,95	0,00	0,95	2,05
real_31	28,64	0,03	28,62	22,81	28,63	0,02	28,62	22,68	28,63	0,02	28,62	21,25
real_32	8,26	0,02	8,22	7,08	8,27	0,05	8,22	7,04	8,26	0,03	8,22	7,16
real_33	4,01	0,08	3,95	3,70	3,98	0,05	3,96	3,27	3,96	0,01	3,94	4,93
real_34	5,82	0,04	5,74	13,07	5,81	0,08	5,71	12,88	5,81	0,07	5,72	13,16
real_35	32,41	0,90	31,53	26,35	32,34	0,63	31,62	23,93	32,44	0,83	30,86	24,92
real_36	10,24	0,00	10,24	3,38	10,24	0,00	10,24	3,41	10,24	0,00	10,24	3,43
real_37	8,11	0,14	7,81	6,81	8,03	0,14	7,82	9,05	8,10	0,10	7,90	5,56
real_38	23,45	0,01	23,44	132,78	23,44	0,01	23,44	135,89	23,45	0,01	23,44	151,98
real_39	2,95	0,00	2,95	3,08	2,95	0,00	2,95	3,15	2,95	0,00	2,95	2,57
real_40	7,69	0,05	7,66	6,20	7,71	0,06	7,66	5,48	7,69	0,05	7,66	5,84
real_41	2,44	0,00	2,44	2,66	2,44	0,00	2,44	2,79	2,44	0,00	2,44	2,68
real_42	6,93	0,00	6,93	5,23	6,93	0,00	6,93	4,96	6,93	0,00	6,93	5,25
real_43	5,49	0,00	5,49	5,42	5,49	0,00	5,49	6,42	5,49	0,00	5,49	6,32
real_44	0,74	0,00	0,74	0,86	0,74	0,00	0,74	0,88	0,74	0,00	0,74	0,91
real_45	8,98	0,03	8,89	10,78	8,94	0,04	8,89	13,92	8,97	0,04	8,89	12,97
real_46	1,98	0,00	1,98	4,54	1,98	0,00	1,98	4,52	1,98	0,00	1,98	4,47
real_47	4,17	0,01	4,16	2,56	4,18	0,01	4,16	2,66	4,17	0,01	4,16	2,44
real_48	0,83	0,00	0,83	0,98	0,83	0,00	0,83	0,89	0,83	0,00	0,83	0,96
real_49	2,88	0,00	2,88	1,33	2,88	0,00	2,88	1,38	2,88	0,00	2,88	1,34
real_50	5,88	0,07	5,78	9,04	5,85	0,07	5,77	9,88	5,84	0,07	5,71	10,81
real_51	4,61	0,02	4,58	4,40	4,60	0,02	4,55	6,15	4,61	0,02	4,60	5,92
mittel	7,18		7,13		7,18		7,13		7,18		7,11	

Tabelle 6.14: Ergebnisse von EAe mit OX3

NAME	EAe/GOX/0,01/0				EAe/GOX/0,01/-1				EAe/GOX/0,01/-0,01			
	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek
real_21	3,72	0,01	3,71	12,56	3,72	0,01	3,70	40,12	3,72	0,01	3,70	42,43
real_22	18,53	0,00	18,53	6,84	18,53	0,00	18,53	20,33	18,53	0,00	18,53	19,18
real_23	0,98	0,00	0,98	1,84	0,98	0,00	0,98	5,18	0,98	0,00	0,98	4,91
real_24	5,56	0,00	5,56	4,72	5,56	0,00	5,56	11,28	5,56	0,00	5,56	11,16
real_25	0,80	0,00	0,80	1,22	0,80	0,00	0,80	3,38	0,80	0,00	0,80	3,39
real_26	0,38	0,00	0,38	1,07	0,38	0,00	0,38	2,76	0,38	0,00	0,38	2,70
real_27	2,87	0,00	2,87	1,06	2,87	0,00	2,87	2,70	2,87	0,00	2,87	2,74
real_28	1,86	0,02	1,84	1,91	1,85	0,01	1,84	4,56	1,86	0,02	1,84	5,02
real_29	10,54	0,04	10,48	24,51	10,51	0,03	10,47	88,60	10,55	0,04	10,49	79,45
real_30	0,95	0,00	0,95	1,88	0,95	0,00	0,95	5,14	0,95	0,00	0,95	5,21
real_31	28,63	0,02	28,62	19,51	28,62	0,00	28,62	57,20	28,62	0,00	28,62	55,65
real_32	8,29	0,06	8,22	8,65	8,28	0,02	8,27	24,68	8,29	0,04	8,27	22,41
real_33	3,98	0,05	3,94	4,36	4,01	0,07	3,95	10,49	3,98	0,05	3,96	9,66
real_34	5,75	0,05	5,67	12,61	5,74	0,06	5,67	38,88	5,76	0,07	5,68	34,44
real_35	31,69	0,71	30,56	26,63	31,92	0,62	30,69	66,70	32,11	0,81	30,61	72,31
real_36	10,24	0,00	10,24	3,58	10,24	0,00	10,24	9,38	10,24	0,00	10,24	9,38
real_37	8,08	0,13	7,82	7,82	8,06	0,16	7,80	24,06	8,03	0,14	7,81	22,88
real_38	23,44	0,01	23,44	147,55	23,44	0,01	23,44	509,49	23,44	0,00	23,44	481,05
real_39	2,95	0,00	2,95	2,76	2,95	0,00	2,95	8,23	2,95	0,00	2,95	7,78
real_40	7,67	0,04	7,66	5,88	7,68	0,05	7,66	12,89	7,66	0,01	7,66	19,93
real_41	2,44	0,00	2,44	2,76	2,44	0,00	2,44	7,50	2,44	0,00	2,44	7,53
real_42	6,93	0,00	6,93	4,61	6,93	0,00	6,93	12,96	6,93	0,00	6,93	13,74
real_43	5,52	0,02	5,49	7,08	5,51	0,02	5,49	23,86	5,52	0,01	5,49	21,40
real_44	0,74	0,00	0,74	0,65	0,74	0,00	0,74	1,60	0,74	0,00	0,74	1,62
real_45	8,98	0,01	8,96	14,22	8,98	0,01	8,96	47,46	8,96	0,04	8,89	57,03
real_46	1,98	0,00	1,98	4,82	1,98	0,00	1,98	13,11	1,98	0,00	1,98	13,10
real_47	4,18	0,02	4,16	3,10	4,17	0,01	4,16	8,08	4,17	0,01	4,16	8,06
real_48	0,83	0,00	0,83	1,05	0,83	0,00	0,83	1,84	0,83	0,00	0,83	2,18
real_49	2,88	0,00	2,88	1,27	2,88	0,00	2,88	3,46	2,88	0,00	2,88	3,45
real_50	5,75	0,05	5,68	14,87	5,75	0,06	5,67	43,18	5,79	0,06	5,71	33,16
real_51	4,60	0,02	4,55	6,69	4,62	0,02	4,60	18,31	4,61	0,02	4,59	22,11
mittel	7,15		7,09		7,16		7,10		7,17		7,10	

Tabelle 6.15: Ergebnisse von EAe mit GOX

NAME	$\frac{Z_{EAe/OX3/0,01/0}}{Z_{XOPTS}}$	$\frac{Z_{EAe/OX3/0,01/-1}}{Z_{XOPTS}}$	$\frac{Z_{EAe/OX3/0,01/-0,01}}{Z_{XOPTS}}$	$\frac{Z_{EAe/GOX/0,01/0}}{Z_{XOPTS}}$	$\frac{Z_{EAe/GOX/0,01/-1}}{Z_{XOPTS}}$	$\frac{Z_{EAe/GOX/0,01/-0,01}}{Z_{XOPTS}}$
real_21	0,9841	**0,9815**	0,9841	0,9841	0,9841	0,9841
real_22	**0,9835**	**0,9835**	**0,9835**	**0,9835**	**0,9835**	**0,9835**
real_23	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_24	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_25	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_26	**0,9500**	**0,9500**	**0,9500**	**0,9500**	**0,9500**	**0,9500**
real_27	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_28	**0,9686**	0,9791	0,9738	0,9738	**0,9686**	0,9738
real_29	1,0086	1,0096	1,0086	1,0105	**1,0077**	1,0115
real_30	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_31	1,0007	1,0003	1,0003	1,0003	**1,0000**	**1,0000**
real_32	**1,0049**	1,0061	**1,0049**	1,0085	1,0073	1,0085
real_33	1,0075	1,0000	**0,9950**	1,0000	1,0075	1,0000
real_34	1,0246	1,0229	1,0229	1,0123	**1,0106**	1,0141
real_35	1,0499	1,0476	1,0509	**1,0266**	1,0340	1,0402
real_36	**0,9697**	**0,9697**	**0,9697**	**0,9697**	**0,9697**	**0,9697**
real_37	1,0279	**1,0177**	1,0266	1,0241	1,0215	1,0177
real_38	0,9820	**0,9816**	0,9820	**0,9816**	**0,9816**	**0,9816**
real_39	**0,9966**	**0,9966**	**0,9966**	**0,9966**	**0,9966**	**0,9966**
real_40	0,8769	0,8791	0,8769	0,8746	0,8757	**0,8734**
real_41	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_42	**0,9943**	**0,9943**	**0,9943**	**0,9943**	**0,9943**	**0,9943**
real_43	**1,0167**	**1,0167**	**1,0167**	1,0222	1,0204	1,0222
real_44	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_45	1,0101	1,0056	**1,0090**	1,0101	1,0101	1,0079
real_46	**0,9950**	**0,9950**	**0,9950**	**0,9950**	**0,9950**	**0,9950**
real_47	**0,8872**	0,8894	**0,8872**	0,8894	**0,8872**	**0,8872**
real_48	**0,9881**	**0,9881**	**0,9881**	**0,9881**	**0,9881**	**0,9881**
real_49	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_50	1,0407	1,0354	1,0336	**1,0177**	**1,0177**	1,0248
real_51	1,0066	**1,0044**	1,0066	**1,0044**	1,0087	1,0066
mittel	0,9927	0,9921	0,9921	**0,9909**	0,9910	0,9913

Tabelle 6.16: Ergebnisse von EAe in Relation zu XOPTS

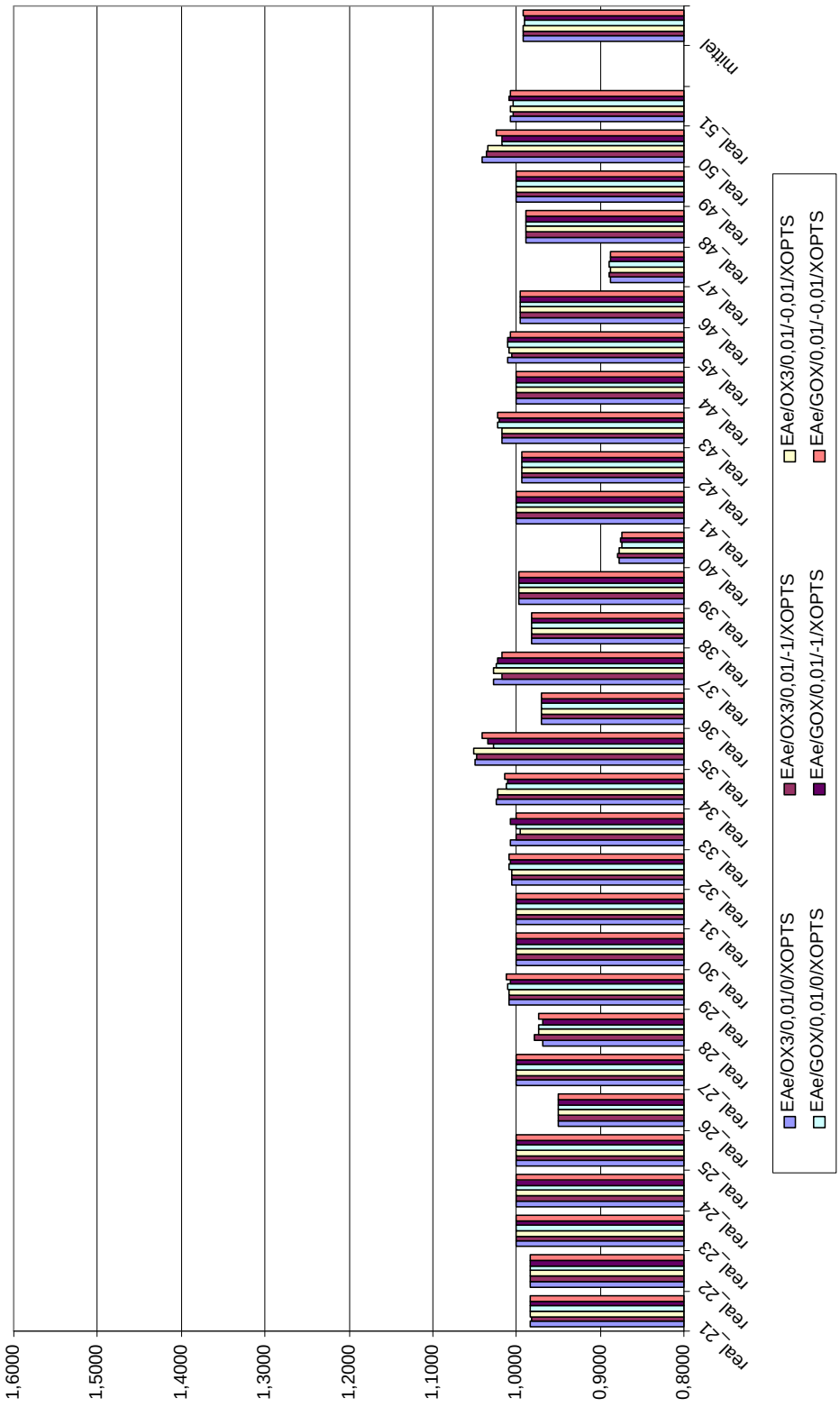


Abbildung 6.4: Graphische Darstellung der von EAe erzielten Ergebnisse in Relation zu XOPTS

6.2.3.3 EA mit Element-Rpräsentation und B&B: EAebb

In Tabellen 6.18 und 6.19 sind die Ergebnisse von EAebb dargestellt. In Tabelle 6.20 und Abbildung 6.5 werden diese in Relation zu den von XOPTS erzielten Ergebnissen gebracht.

Die Ergebnisse der EAebb Varianten übertreffen im Mittel die von XOPTS und EAe erzielten Ergebnisse. Die Laufzeiten liegen im Sekunden- bis Minutenbereich, außer bei Instanz real_35, bei der die Laufzeiten im Stundenbereich liegen.

Die besten Ergebnisse der EAebb Varianten werden mit EAebb/GOX/0,01/-1 und EAebb/GOX/0,01/-0,01 erzielt. Die durchschnittlichen Ergebnisse von zehn EAebb/GOX/0,01/-1 Durchläufen sind um 1,1% besser als jene von XOPTS (Tabelle 6.17). Wenn man hingegen die jeweils besten Ergebnisse der zehn Durchläufe betrachtet, erzeugen diese sogar um 1,56% weniger Verschnitt als die von XOPTS.

In Tabelle 6.17 sind die Ergebnisse von EAebb überblicksmäßig dargestellt. Dort wird unter anderem angegeben, wie oft dieses Verfahren bessere, gleich gute oder schlechtere Ergebnisse als XOPTS erzielt.

Verfahren	besser	gleich	schlechter	Z/Z_{XOPTS}	Z/KUS
Mittel EAebb/OX3/0,01/-1	13	8	10	0,9907	1,1844
Min EAebb/OX3/0,01/-1	16	11	4	0,9847	1,1775
Mittel EAebb/GOX/0,01/-1	14	8	9	0,9890	1,1826
Min EAebb/GOX/0,01/-1	16	13	2	0,9844	1,1772

Tabelle 6.17: Überblick zu den Ergebnissen von EAebb

Die EAebb Varianten mit dem GOX Operator sind besser als jene mit OX3. Das BBHEU Verfahren verbessert die Ergebnisse im Vergleich zu EAe im Mittel um etwa 0,2%. Die Mutationsvarianten mit Gruppierungsoperator erzielen in Kombination mit BBHEU und GOX die besseren Ergebnisse als der Zweier-Austausch, welcher höchstwahrscheinlich die von BBHEU erzeugten Abschnitte zum Teil wieder zerstört.

Bei den zwei Instanzen, wo EAebb/GOX/0,01/-1 schlechter als XOPTS ist, sind die Ergebnisse um nur $0,09$ bzw. $0,05$ Rohlinge schlechter als die Ergebnisse von XOPTS. Im Gegensatz dazu kann der EA bei zwei Instanzen jeweils einen ganzen Rohling gegenüber XOPTS einsparen. Wenn man die vom EA verbrauchten ganzen Rohlinge mit den minimal benötigten ganzen Rohlingen (KUS ganzzahlig aufgerundet) vergleicht, stellt man fest, dass in 17 Fällen genau so viele Rohlinge, in 8 Fällen ein Rohling mehr und nur in 6 Fällen mehr als ein Rohling mehr verbraucht werden. Die 31 Instanzen verbrauchen mindestens (KUS ganzzahlig aufgerundet) 198 Rohlinge, der EA 229 Rohlinge und XOPTS ohne Einhaltung der Nebenbedingungen 231 Rohlinge.

NAME	EAebb/OX3/0,01/0				EAebb/OX3/0,01/-1				EAebb/OX3/0,01/-0,01			
	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek
rea_21	3,71	0,01	3,70	23,04	3,72	0,01	3,70	22,11	3,71	0,01	3,70	23,54
rea_22	18,53	0,00	18,53	7,96	18,53	0,00	18,53	8,34	18,54	0,01	18,53	7,89
rea_23	0,98	0,00	0,98	2,23	0,98	0,00	0,98	2,24	0,98	0,00	0,98	2,27
rea_24	5,56	0,00	5,56	5,04	5,56	0,00	5,56	5,15	5,56	0,00	5,56	5,08
rea_25	0,80	0,00	0,80	1,42	0,80	0,00	0,80	1,44	0,80	0,00	0,80	1,41
rea_26	0,38	0,00	0,38	1,73	0,38	0,00	0,38	1,63	0,38	0,00	0,38	1,82
rea_27	2,87	0,00	2,87	1,44	2,87	0,00	2,87	1,48	2,87	0,00	2,87	1,48
rea_28	1,86	0,02	1,84	2,47	1,85	0,02	1,84	2,50	1,85	0,01	1,84	2,46
rea_29	10,56	0,05	10,49	32,36	10,54	0,04	10,47	31,86	10,52	0,04	10,47	37,78
rea_30	0,95	0,00	0,95	2,29	0,95	0,00	0,95	2,32	0,95	0,00	0,95	2,38
rea_31	28,63	0,02	28,62	118,98	28,63	0,02	28,62	107,13	28,64	0,03	28,62	114,81
rea_32	8,27	0,03	8,22	8,24	8,27	0,02	8,22	7,83	8,25	0,03	8,22	10,21
rea_33	3,97	0,05	3,93	4,33	3,96	0,01	3,94	4,79	3,96	0,01	3,95	4,58
rea_34	5,86	0,09	5,71	217,34	5,81	0,07	5,74	261,54	5,85	0,06	5,75	255,53
rea_35	30,45	0,44	29,94	3929,57	31,12	1,27	29,48	3332,19	30,60	0,68	29,50	3695,82
rea_36	10,24	0,00	10,24	3,46	10,24	0,00	10,24	3,58	10,24	0,00	10,24	3,46
rea_37	8,12	0,14	7,80	10,20	8,10	0,13	7,82	11,20	8,18	0,03	8,16	6,94
rea_38	23,45	0,01	23,44	163,02	23,45	0,03	23,38	150,72	23,46	0,01	23,44	142,96
rea_39	2,95	0,00	2,95	3,47	2,95	0,00	2,95	3,54	2,95	0,00	2,95	3,46
rea_40	7,67	0,04	7,66	5,91	7,66	0,00	7,66	7,47	7,66	0,00	7,66	6,08
rea_41	2,44	0,00	2,44	2,83	2,44	0,00	2,44	2,85	2,44	0,00	2,44	2,87
rea_42	6,93	0,00	6,93	5,59	6,93	0,00	6,93	5,40	6,93	0,00	6,93	5,56
rea_43	5,49	0,00	5,49	6,25	5,49	0,00	5,49	6,53	5,49	0,00	5,49	6,10
rea_44	0,74	0,00	0,74	0,94	0,74	0,00	0,74	0,97	0,74	0,00	0,74	0,95
rea_45	8,93	0,03	8,89	14,94	8,93	0,04	8,89	16,33	8,93	0,05	8,89	15,61
rea_46	1,98	0,00	1,98	6,44	1,98	0,00	1,98	6,44	1,98	0,00	1,98	6,40
rea_47	4,18	0,01	4,16	4,87	4,18	0,01	4,16	4,33	4,18	0,02	4,12	4,11
rea_48	0,83	0,00	0,83	2,65	0,83	0,00	0,83	2,27	0,83	0,00	0,83	2,40
rea_49	2,88	0,00	2,88	1,45	2,88	0,00	2,88	1,46	2,88	0,00	2,88	1,46
rea_50	5,88	0,07	5,75	265,46	5,87	0,07	5,70	276,52	5,88	0,07	5,76	237,13
rea_51	4,58	0,03	4,55	8,92	4,61	0,02	4,55	6,31	4,61	0,02	4,55	6,38
mittel	7,12		7,07		7,14		7,06		7,12		7,07	

Tabelle 6.18: Ergebnisse von EAebb mit OX3

NAME	EAebb/GOX/0,01/0				EAebb/GOX/0,01/-1				EAebb/GOX/0,01/-0,01			
	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek	$\overline{Z}$	dev	min Z	Sek
real_21	3,72	0,01	3,71	22,77	3,71	0,01	3,69	24,47	3,72	0,01	3,70	22,38
real_22	18,53	0,00	18,53	7,16	18,53	0,00	18,53	8,38	18,53	0,00	18,53	7,70
real_23	0,98	0,00	0,98	2,09	0,98	0,00	0,98	2,07	0,98	0,00	0,98	1,99
real_24	5,56	0,00	5,56	4,81	5,56	0,00	5,56	4,87	5,56	0,00	5,56	4,78
real_25	0,80	0,00	0,80	1,35	0,80	0,00	0,80	1,33	0,80	0,00	0,80	1,30
real_26	0,38	0,00	0,38	1,80	0,38	0,00	0,38	1,66	0,38	0,00	0,38	1,55
real_27	2,87	0,00	2,87	1,16	2,87	0,00	2,87	1,11	2,87	0,00	2,87	1,14
real_28	1,87	0,02	1,84	2,77	1,85	0,01	1,84	2,30	1,85	0,01	1,84	2,74
real_29	10,53	0,03	10,47	40,54	10,50	0,05	10,43	39,95	10,51	0,05	10,47	36,99
real_30	0,95	0,00	0,95	2,15	0,95	0,00	0,95	2,12	0,95	0,00	0,95	2,12
real_31	28,62	0,00	28,62	97,03	28,63	0,02	28,62	99,50	28,62	0,00	28,62	105,26
real_32	8,26	0,02	8,22	9,04	8,27	0,05	8,22	10,49	8,30	0,05	8,22	8,54
real_33	3,98	0,05	3,96	4,68	3,96	0,01	3,93	5,05	3,96	0,01	3,93	4,48
real_34	5,79	0,07	5,70	284,67	5,78	0,07	5,68	339,75	5,74	0,05	5,66	301,55
real_35	31,04	1,42	29,61	2755,83	30,86	1,10	29,61	3459,32	31,06	1,21	29,79	2640,08
real_36	10,24	0,00	10,24	3,68	10,24	0,00	10,24	3,74	10,24	0,00	10,24	3,68
real_37	8,06	0,13	7,87	10,43	7,99	0,15	7,80	13,77	7,90	0,10	7,82	13,86
real_38	23,45	0,01	23,44	153,24	23,44	0,00	23,44	150,85	23,44	0,01	23,44	150,32
real_39	2,95	0,00	2,95	3,77	2,95	0,00	2,95	3,63	2,95	0,00	2,95	4,23
real_40	7,66	0,00	7,66	6,56	7,66	0,00	7,66	6,05	7,66	0,00	7,66	7,05
real_41	2,44	0,00	2,44	2,95	2,44	0,00	2,44	2,96	2,44	0,00	2,44	2,91
real_42	6,93	0,00	6,93	5,01	6,93	0,00	6,93	4,58	6,93	0,00	6,93	4,81
real_43	5,52	0,01	5,49	7,25	5,52	0,02	5,49	6,61	5,52	0,02	5,49	6,77
real_44	0,74	0,00	0,74	0,66	0,74	0,00	0,74	0,67	0,74	0,00	0,74	0,66
real_45	8,93	0,05	8,89	19,83	8,92	0,04	8,89	19,95	8,95	0,04	8,89	15,05
real_46	1,98	0,00	1,98	6,47	1,98	0,00	1,98	6,45	1,98	0,00	1,98	6,47
real_47	4,17	0,01	4,16	5,43	4,17	0,01	4,16	5,01	4,18	0,01	4,16	4,17
real_48	0,83	0,00	0,83	2,29	0,83	0,00	0,83	2,15	0,83	0,00	0,83	2,29
real_49	2,88	0,00	2,88	1,39	2,88	0,00	2,88	1,39	2,88	0,00	2,88	1,42
real_50	5,77	0,06	5,71	326,77	5,75	0,04	5,70	432,05	5,77	0,07	5,68	364,00
real_51	4,61	0,02	4,57	9,73	4,62	0,03	4,57	8,14	4,60	0,03	4,55	8,98
mittel	7,13		7,06		7,12		7,06		7,12		7,06	

Tabelle 6.19: Ergebnisse von EAebb mit GOX

NAME	$\frac{Z_{EAebb/OX3/0,01/0}}{Z_{XOPTS}}$	$\frac{Z_{EAebb/OX3/0,01/-1}}{Z_{XOPTS}}$	$\frac{Z_{EAebb/OX3/0,01/-0,01}}{Z_{XOPTS}}$	$\frac{Z_{EAebb/GOX/0,01/0}}{Z_{XOPTS}}$	$\frac{Z_{EAebb/GOX/0,01/-1}}{Z_{XOPTS}}$	$\frac{Z_{EAebb/GOX/0,01/-0,01}}{Z_{XOPTS}}$
real_21	**0,9815**	0,9841	**0,9815**	0,9841	**0,9815**	0,9841
real_22	**0,9835**	**0,9835**	0,9841	**0,9835**	**0,9835**	**0,9835**
real_23	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_24	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_25	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_26	**0,9500**	**0,9500**	**0,9500**	**0,9500**	**0,9500**	**0,9500**
real_27	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_28	0,9738	**0,9686**	**0,9686**	0,9791	**0,9686**	**0,9686**
real_29	1,0125	1,0105	1,0086	1,0096	**1,0067**	1,0077
real_30	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_31	1,0003	1,0003	1,0007	**1,0000**	1,0003	**1,0000**
real_32	1,0061	1,0061	**1,0036**	1,0049	1,0061	1,0097
real_33	0,9975	**0,9950**	**0,9950**	1,0000	**0,9950**	**0,9950**
real_34	1,0317	1,0229	1,0299	1,0194	1,0176	**1,0106**
real_35	**0,9864**	1,0081	0,9913	1,0055	0,9997	1,0062
real_36	**0,9697**	**0,9697**	**0,9697**	**0,9697**	**0,9697**	**0,9697**
real_37	1,0292	1,0266	1,0368	1,0215	1,0127	**1,0013**
real_38	0,9820	0,9820	0,9824	0,9820	**0,9816**	**0,9816**
real_39	**0,9966**	**0,9966**	**0,9966**	**0,9966**	**0,9966**	**0,9966**
real_40	0,8746	**0,8734**	**0,8734**	**0,8734**	**0,8734**	**0,8734**
real_41	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_42	**0,9943**	**0,9943**	**0,9943**	**0,9943**	**0,9943**	**0,9943**
real_43	**1,0167**	**1,0167**	**1,0167**	1,0222	1,0222	1,0222
real_44	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_45	1,0045	1,0045	1,0045	1,0045	**1,0034**	1,0067
real_46	**0,9950**	**0,9950**	**0,9950**	**0,9950**	**0,9950**	**0,9950**
real_47	0,8894	0,8894	0,8894	**0,8872**	**0,8872**	0,8894
real_48	**0,9881**	**0,9881**	**0,9881**	**0,9881**	**0,9881**	**0,9881**
real_49	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**	**1,0000**
real_50	1,0407	1,0389	1,0407	1,0212	**1,0177**	1,0212
real_51	**1,0000**	1,0066	1,0066	1,0066	1,0087	1,0044
mittel	0,9904	0,9907	0,9906	0,9903	**0,9890**	**0,9890**

Tabelle 6.20: Ergebnisse von EAebb in Relation zu XOPTS

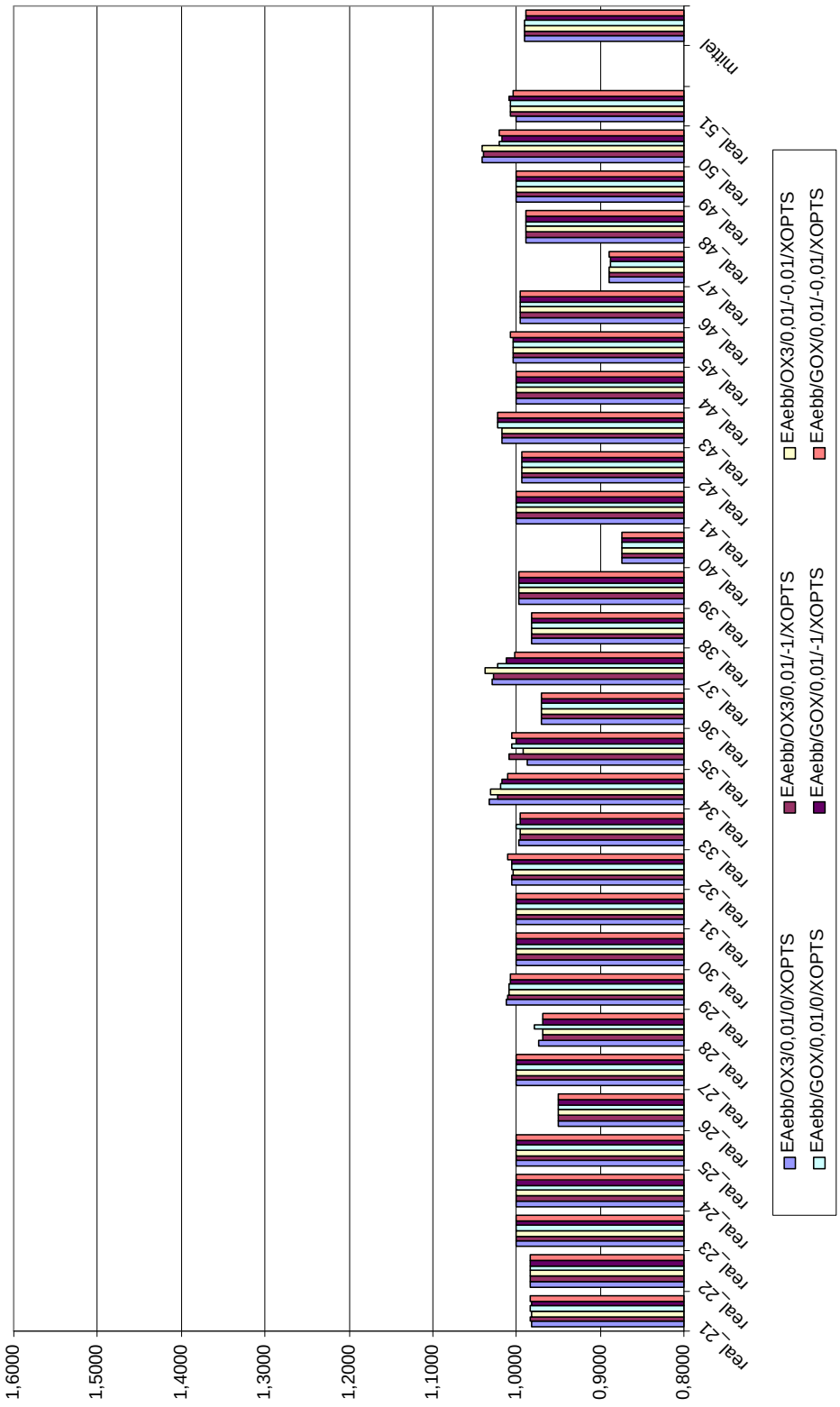


Abbildung 6.5: Graphische Darstellung der von EAebb erzielten Ergebnisse in Relation zu XOPTS

Kapitel 7

Zusammenfassung

Ausgehend von der Problembeschreibung des 2BP-WR und aus der Literatur bekannten Verfahren zur Lösung von zweidimensionalen *Bin Packing* Problemen wurden im Rahmen dieser Diplomarbeit verschiedene Verfahren zur Lösung des 2BP-WR entwickelt. Die Verfahren wurden ausführlich getestet und mit Ergebnissen eines kommerziellen Optimierers (XOPTS) verglichen.

Die komplexen Nebenbedingungen, die sich durch die logischen Gruppen und Transportwagen ergeben, erschweren die Suche nach guten Lösungen für das 2BP-WR. Die aus der Literatur bekannten und existierenden kommerziellen Verfahren können nur zum Teil verwendet werden, da sie nicht auf die Reihenfolge, in der Fertigteile aus den Rohlingen herausgeschnitten werden, achten. Verfahren zur Lösung des 2BP-WR hingegen müssen diese Reihenfolge berücksichtigen. Deshalb können diese weder Symmetrien nützen, noch die Reihenfolge im Nachhinein verändern. Die zum Vergleich der Verfahren benutzten Lösungen des Optimierers XOPTS halten die Nebenbedingungen nicht ein. Dadurch kann XOPTS potenziell bessere aber in Hinblick auf das 2BP-WR ungültige Schnittmuster finden, was bei allen in dieser Arbeit angestellten Vergleichen beachtet werden sollte.

Zu Beginn der Arbeit wurden zwei *greedy* Verfahren (FFF und BFC) entwickelt. Diese haben sich zwar als sehr schnell herausgestellt, sie erreichen aber nicht die Ergebnisse von XOPTS, sondern sind hinsichtlich des Verschnitts um 10% bzw. 15% schlechter.

Im Anschluss daran wurde ein zweiphasiges Verfahren (BBHEU) entwickelt, welches in der ersten Phase lokal optimale Abschnitte mittels B&B erzeugt, die dann in der zweiten Phase heuristisch und unter Einhaltung der Nebenbedingungen auf die Rohlinge verteilt werden. Es hat sich herausgestellt, dass ein zweiphasiger Lösungsansatz nicht zielführend ist, da die Schwierigkeiten, die sich aus der Einhaltung der Nebenbedingungen ergeben, in der zweiten Phase zu sehr schlechten Ergebnissen führen können. Im Mittel sind die Ergebnisse um etwa 12% schlechter als jene von XOPTS. Aber die Erzeugung lokal optimaler Abschnitte in der ersten Phase könnte in anderen Verfahren erfolgreicher zum Einsatz kommen.

In weiterer Folge wurde ein Verfahren (BBALG) entwickelt, welches einzelne Rohlinge mittels B&B optimiert. Es werden bessere Ergebnisse erzielt (2-3% schlechter als XOPTS) als mit den bisher beschriebenen Verfahren, aber die Güte der Ergebnisse kann für verschiedene

Instanzen sehr unterschiedlich sein, da einzelne sehr gut befüllte Rohlinge nicht unbedingt eine gute Gesamtlösung zur Folge haben.

Zum Schluss wurden drei Varianten eines evolutionären Algorithmus entwickelt, der auf dem Prinzip der Hybridisierung basiert. Eine heuristische Dekodierfunktion erzeugt korrekte Lösungen aus den Chromosomen. In der ersten Variante (EAet) stellen diese die Reihenfolge der Elementtypen dar, in den anderen beiden Varianten (EAe und EAebb) wird die Reihenfolge der einzelnen Elemente dargestellt. Für EAe und EAebb wurden der GOX Rekombinationsoperator sowie der als Mutation eingesetzte Gruppierungsoperator entwickelt, welche die gewählte Repräsentation und Dekodierfunktion berücksichtigen. Der EAebb Algorithmus verwendet zusätzlich das BBHEU Verfahren, um während des Dekodiervorgangs mit einer geringen Wahrscheinlichkeit lokal optimale Abschnitte in die Lösung zu injizieren. Die Ergebnisse der drei evolutionären Algorithmen sind besser (EAet um 0,69%, EAe um 0,91% und EAebb um 1,1%) als jene von XOPTS, obwohl die Nebenbedingungen strikt eingehalten werden. Wenn die besten Varianten von EAe und EAebb ohne Einschränkung durch die Nebenbedingungen ausgeführt werden, liefern diese im Schnitt um 1,33% bessere Lösungen als XOPTS, und es können bei einer Instanz (real_35) sogar bis zu zwei ganze Rohlinge eingespart werden. Die kontinuierlichen unteren Schranken *KUS* zeigen, dass in vielen Fällen keine allzu großen weiteren Einsparungen, d.h. bessere Lösungen mehr möglich sind, so liegt die Differenz zwischen den Ergebnissen und der *KUS* in den meisten Fällen (22 von 31) unter einem Rohling.

Die Resultate der hybriden EAs sind vielversprechend. Es könnte durchaus von Interesse sein diese Idee weiterzuverfolgen. Zum Beispiel könnte eine genauere Repräsentation der Schnittmuster entwickelt werden, welche eine breitere Abdeckung des Lösungsraums ermöglicht sowie die Platzierung der Elemente am Rohling exakt vorgibt und nicht der Dekodierfunktion überlässt.

Literaturverzeichnis

[1] Bäck, T., D. B. Fogel und Z. Michalewicz: *Handbook of Evolutionary Computation*. Oxford University Press, New York, 1997.

[2] Berkey, J. O. und P. Y. Wang: *Two-dimensional finite bin packing algorithms*. Journal of the Operational Research Society, 38:423–429, 1987.

[3] Bisotto, S., F. Corno, P. Prinetto, M. Rebaudengo und M. S. Reorda: *Optimizing Area Loss in Flat Glass Cutting*. In: *GALESIA97, IEE/IEEE International Conference on Genetic ALgorithms in Engineering Systems: Innovations and Applications*, Glasgow (UK), september 1997.

[4] Chung, F., M. Garey und D. Johnson: *On packing two-dimensional bins*. SIAM Journal of Algebraic and Discrete Methods, 3:66–76, 1982.

[5] Coffman, Jr., E., M. Garey und D. Johnson: *Approximation Algorithms for Bin Packing: A Survey*. In: Hochbaum, D. (Hrsg.): *Approximation Algorithms for NP-Hard Problems*, S. 46–93. PWS Publishing, Boston, 1996.

[6] Davis, L. (Hrsg.): *A Handbook Of Genetic Algorithms*. International Thomson Computer Press, 1991.

[7] Dyckhoff, H., G. Scheithauer und J. Terno: *Cutting and Packing: An Annotated Bibliography*. In: Dell'Amico, M., F. Maffioli und S. Martello (Hrsg.): *Annotated Bibliographies in Combinatorial Optimization*, S. 393–412. Wiley, 1997.

[8] E.G. Coffman, J., M. Garey, D. Johnson und R. Tarjan: *Performance bounds for level-oriented two-dimensional packing algorithms*. SIAM Journal on Computing, 9:808–826, 1980.

[9] Frenk, J. und G. Galambos: *Hybrid Next-Fit Algorithm for the Two-Dimensional Rectangular Bin-Packing Problem*. Computing, 39:201–217, 1987.

[10] Fritsch, A.: *Verschnittoptimierung durch iteriertes Matching*. Diplomarbeit, Universität Osnabrück, Deutschland, 1994.

[11] Garey, M., R. Graham, D. Johnson und A. Yao: *Resource constrained scheduling as generalized bin packing*. Journal of Combinatorial Theory (A), 21:257–298, 1976.

[12] Gilmore, P. C. und R. E. Gomory: *A Linear Programming Approach to the Cutting-Stock Problem (Part I)*. Operations Research, 9:849–859, 1961.

[13] Gilmore, P. C. und R. E. Gomory: *A Linear Programming Approach to the Cutting-Stock Problem (Part II)*. Operations Research, 11:363–888, 1963.

[14] Goldberg, D. und R. Lingle: *Alleles, Loci, and the Travelling Salesman Problem*. In: Grefenstette, J. J. (Hrsg.): *Proceedings of the First International Conference on Genetic Algorithms*, S. 154–159. Lawrence Erlbaum, 1985.

[15] Grabner, J. und J. Roschitz: *Neue Baustoffe, Glas 1 (1999), Vorlesung TU GRAZ*, 2001. http://www.gra-pa.at/projects/NeueBaustoffe/index2.html.

[16] Holland, J.: *Adaptation In Natural and ArtificialSystems*. University of Michigan Press, 1975.

[17] Hopper, E.: *Two-Dimensional Packing Utilising Evolutionary Algorithms and Other Meta-Heuristic Methods*. Doktorarbeit, University of Wales, Cardiff, U.K., 2000.

[18] Hwang, S.-M., C.-Y. Kao und J.-T. Horng: *On Solving Rectangle Bin Packing Problems Using GAs*. In: *Proceedings of the 1994 IEEE International Conference on Systems, Man, and Cybernetics*, S. 1583–1590. IEEE Press, 1997.

[19] Jarvis, M. G.: *Fast Algorithms for the two-dimensional bin packing problem*. Diplomarbeit, Dalhousie university Halifax, Nova Scotia, Canada, 2001.

[20] Johnson, D., A. Demers, J. Uhlman, M. Garey und R. Graham: *Worst-case performance bounds for simple one-dimensional packing algorithms*. SIAM Journal on Computing, 3:299–325, 1974.

[21] Johnson, D. S.: *Near-Optimal Bin Packing Algorithms*. Doktorarbeit, Massachusetts Institute of Technology, Cambridge, 1973.

[22] Korte, B. und J. Vygen: *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, Berlin Heidelberg, 2000.

[23] Kröger, B.: *Guillotineable Bin Packing: A Genetic Approach*. European Journal of Operational Research, 84:545–661, 1995.

[24] Lodi, A.: *Algorithms for Two-Dimensional Bin Packing and Assignment Problems*. Doktorarbeit, DEIS, Università di Bologna, 1999.

[25] Lodi, A., S. Martello und M. Monaci: *Two-dimensional packing problems: A survey*. European Journal of Operational Research, 141:241–252, 2002.

[26] Lodi, A., S. Martello und D. Vigo: *Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems*. INFORMS Journal on Computing, 11:345–357, 1999.

[27] Lodi, A., S. Martello und D. Vigo: *Recent Advances on Two-Dimensional Bin Packing Problems*. Discrete Applied Mathematics, 123:373–390, 2002.

[28] Martello, S. und D. Vigo: *Exact Solutions of the Two-Dimensional Finite Bin Packing Problem*. Management Science, 44:388–399, 1998.

[29] Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1996.

[30] Morabito, R. und M. N. Arenales: *Staged and Constrained Two-Dimensional Guillotine Cutting Problems: An AND/OR-Graph Approach*. European Journal of Operational Research, 94(3):548–560, 1996.

[31] Petzold, A., H. Marusch und B. Schramm: *Der Baustoff Glas*. Verlag für Bauwesen, Berlin, 1990.

[32] Raidl, G.: *EAlib v1.7 – A Generic Library for Metaheuristics*. Technische Universität Wien, 2002.

[33] Schaffer, J. D. (Hrsg.): *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[34] Whitley, D.: *The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trial is best*. In: Schaffer, J. D. [33], S. 116–121.

[35] Whitley, D., T. Starkweather und D. Fuquay: *Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator*. In: Schaffer, J. D. [33], S. 133–140.

Anhang A

Die Cutter Anwendung

A.1 Bedienungsanleitung

Das Programm **cutter** wurde in C++ implementiert und mit dem GCC 3.2 Compiler unter LINUX 2.4.19 kompiliert. Grundlage der evolutionären Algorithmen ist die von Günther Raidl in der Abteilung Algorithmen und Datenstrukuren am Institut für Computergraphik der Technischen Universität Wien entwickelte Bibliothek für evolutionäre Algorithmen: EAlib [32].

cutter kann mit den in Tabelle A.1 beschriebenen Parametern gestartet werden. Die *default* Werte sind in Klammern, die erlaubten Werte in geschwungenen Klammern oder Itervallen angegebenangegeben.

`cut_heu`	`(FFF) {BFC BBHEU BBALG EAet EAe}`	Heuristik
`cut_pfile`	`{pfile}`	Datei mit Probleminstanz
`cut_ifile`	`{ifile}`	Datei für Schnittanweisung
`cut_imfile`	`{imfile}`	Datei für graphische Darstellung
`cut_HFACTOR`	`(0.5) [0,1]`	*HF* für BFC
`cut_WFACTOR`	`(0.5) [0,1]`	*WF* für BFC
`cut_hforce`	`(0) [0,10]`	*FH* für BBALG
`cut_wforce`	`(0) [0,10]`	*FW* für BBALG
`cut_xtype`	`(OX3) {PMX, GOX}`	Rekombinationstxp
`cut_mtype`	`(0) [-1,0.5]`	Mutationstyp (siehe Abschnitt 5.3)
`cut_useBB`	`(0) {0 1}`	Verwendung von B&B in EAe
`cut_pBB`	`(0.001) [0,1]`	Wahrscheinlichkeit für EAe mit B&B

Tabelle A.1: Parameter von **cutter**

cutter muss auf jeden Fall mit den Optionen `cut_pfile` und `cut_ifile` gestartet werden:
`cutter cut_pfile <pfile> cut_ifile <ifile>`

Die Schnittanweisung wird in die Datei `<ifile>` geschrieben, und die graphische Darstellung des Ergebnisses wird, wenn erwünscht, in die Datei `<imfile>` geschrieben. Die graphische Ausgabe kann mit dem Programm xfig unter LINUX bzw. dem Programm jfig unter allen Betriebssystemen, die Java unterstützen, angezeigt werden.

Für die getesteten BBALG Varianten (siehe Abschnitt 4.4) wurden folgende Werte für `cut_hforce` und `cut_wforce` verwendet:

BBALG1 `cut_hforce = 0` und `cut_wforce = 0`

BBALG2 `cut_hforce = 1` und `cut_wforce = 1`

BBALG3 `cut_hforce = 1` und `cut_wforce = 2`

BBALG4 `cut_hforce = 2` und `cut_wforce = 1`

BBALG5 `cut_hforce = 2` und `cut_wforce = 2`

Um **cutter** für EAebb/GOX/0,01/-1 mit graphischer Ausgabe zu starten, kann z.B. folgende Kommandozeile verwendet werden:

```
cutter cut_imfile real_21.fig cut_pfile real_21.cut cut_ifile real_21.txt
cut_heu EAe cut_useBB 1 cut_xtype GOX cut_mtype -1 pmut -0.01
```

Falls einer der EAs zum Einsatz kommt, können auch Optionen der **ealib** angegeben werden. Diese sind in der Dokumentation zur **ealib** genau beschrieben. Einige Optionen werden von **cutter** gesetzt, wie z.B. `eamod` oder `maxi`, und können nicht verändert werden.

Das Format der erzeugten Schnittanweisungen wird an folgendem Beispiel erläutert:

```
sheet 1
h 2989
 v 1992 ID: 421486 GID: 3871 not turned wagon: 0 fill: 15 wc 0
 v 2631
  h 1341 ID: 421854 GID: 3870 turned wagon: 0 fill: 15 wc 0
h 5163
 v 1875 ID: 1 GID: 1 turned wagon: 0 fill: 15
```

Der Beginn eines Rohlings wird mittels *sheet* und der Rohlingsnummer gekennzeichnet. Danach folgt die Schnittanweisung für den Rohling. Jeder Schnitt besteht aus der Orientierung (h für horizontal und v für vertikal) und der absoluten Schnittkoordinate. Es erfolgt auch eine Einrückung, die der Schnitttiefe entspricht. Der tiefste Schnitt eines Objekts enthält zusätzlich noch die ID des Objektes; die GID der logischen Gruppe, der das Objekt angehört; die Orientierung des Objekts (`turned` oder `not turned`); die Nummer des Transportwagens; den Füllstand des Transportwagens in mm sowie, falls ein Wagenwechsel erfolgt, den Hinweis `wc` mit der Nummer des zu wechselnden Wagens.

A.2 File Format

Die Probleminstanzen müssen in folgendem Format vorliegen:

- Zeilen, die mit # beginnen, sind Kommentare.
- Die erste Zeile enthält die allgemeinen Maße (in mm) der Instanz:
  `Rohlingshöhe Rohlingsbreite Rohlingsdicke max-Füllmenge min-Füllmenge`
- Danach kommen die einzelnen logischen Gruppen und Elemente: In der ersten Zeile steht die jeweilige `ID` der Gruppe, und in den nachfolgenden Zeilen werden die Spezifikationen der in dieser Gruppe vorhandenen Elemente folgendermaßen angegeben:
  `ID Breite Höhe Anzahl Drehbarkeit K-Faktor`
  Die Maße der Elemente und der K-Faktor werden in mm angegeben, für die Drehbarkeit ist `Y` (ja) oder `N` (nein) zu setzen. Der (optionale) K-Faktor wird zu Breite und Höhe addiert.

Ein Beispiel für das File Format:

```
# HEIGHT WIDTH DEPTH MIN_OCCUPANCY MAX_OCCUPANCY
6000 3000 10 200 250

# LOGICAL_GROUP
1

# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N) K
72638 500 1000 25 Y 2
2278 652 2500 45 N 1

2
2245 211 1072 49 N 0
7338 530 2030 9 N 2
```

Anhang B

Implementierung

Hier werden die wichtigsten Klassen des Projekts **cutter** mit ihren maßgeblichen Methoden vorgestellt. Dieses Projekt stellt die Umsetzung der in der Diplomarbeit beschriebenen Verfahren dar. Die dafür entwickelten Klassen lassen sich in die Bereiche Framework/Infrastruktur und Algorithmen einteilen.

Die Infrastruktur-Klassen sind für das Lesen der Probleminstanzen und das Schreiben der Ergebnisse sowie für die Bereitstellung der Probleminstanz für die Algorithmen-Klassen zuständig. Einen Überblick der wichtigsten Klassen von **cutter** bietet Abbildung B.1.

Die Grundlage der evolutionären Algorithmen ist die von Günther Raidl in der Abteilung Algorithmen und Datenstrukuren am Institut für Computergraphik der Technischen Universität Wien entwickelte Bibliothek für evolutionäre Algorithmen: EAlib [32]. Hier werden nur die im Rahmen des Projekts **cutter** entwickelten Klassen vorgestellt. Es handelt sich um Chromosomklassen, welche von der ealib-Klasse **chromosome** abgeleitet sind.

B.1 Überblick

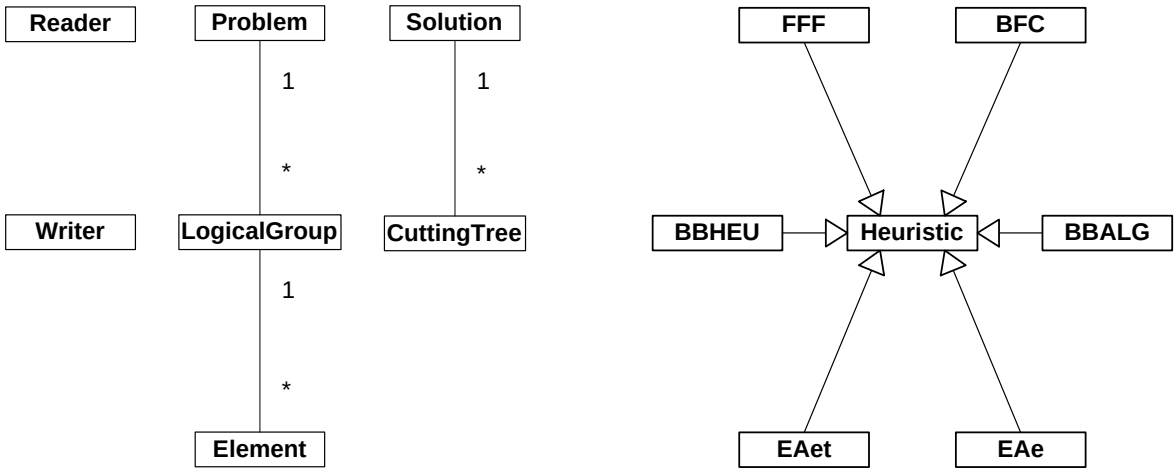


Abbildung B.1: Überblick über wichtigsten Klassen von **cutter**

Infrastruktur-Klassen

Algorithmen-Klassen

B.2 Definitionen

Um Einheitlichkeit zu gewährleisten, wurden folgende Definitionen und Konstanten in einer zentralen Header-Datei zusammengefasst:

```
typedef int                       Integer;
typedef char                      Boolean;
typedef const char *              ConstText;
typedef char *                    Text;
typedef char                      Character;
typedef std::string               String;
typedef double                    Float;

typedef class ProblemClass *      Problem;
typedef class LogicalGroupClass * LogicalGroup;
typedef class ElementClass *      Element;
typedef class SolutionClass *     Solution;
typedef class CuttingTreeClass *  CuttingTree;
typedef class HeuristicClass *    Heuristic;
typedef class ReaderClass *       Reader;
typedef class WriterClass *       Writer;

typedef std::vector<Element>      ElementVector;
typedef std::vector<LogicalGroup> LogicalGroupVector;
typedef std::vector<CuttingTree>  CuttingTreeVector;

typedef class FFFClass *          FFF;
typedef class BBFCClass *         BFC;

typedef class BBHEUClass *        BBHEU;
typedef class BBALGClass *        BBALG;

typedef class EAetlass *          EAet;
typedef class EAeClass *          EAe;
```

B.3 ReaderClass

Die Klasse **Reader** liest eine Probleminstanz (File Format siehe Abschnitt A.2) ein und kann daraus eine Instanz der Klasse **Problem** erzeugen.

Konstruktoren, Destruktoren und Methoden

- **ReaderClass** (Text filename)
- ~**ReaderClass** ()
- Problem **getProblem** ()

B.4 WriterClass

Die Klasse **Writer** erzeugt aus einer Lösung eine Datei, welche die Schnittanweisung enthält. Sie kann die Schnittanweisung auf Benutzerwunsch auch graphisch (in Form einer .fig Datei) erzeugen. Dabei werden die Elemente je nach Abstellplatz des Transportwagens, auf den sie abgelegt werden, verschieden eingefärbt. Die Elemente, nach denen ein Wagenwechsel erfolgt, sind dünkler als die übrigen Elemente.

Konstruktoren, Destruktoren und Methoden

- **WriterClass** (Problem, Solution)
- ~**WriterClass** ()
- void **createImages** (Text filename)
- void **createCuttingInstruction** (Text filename)

B.5 ProblemClass

In der Klasse **Problem** wird eine Problemdefinition gespeichert. Die Methoden dieser Klasse stellen die Probleminstanz zur Verfügung.

Alle Zähler der **LogicalGroup**s und der **Element**s sowie die Elementvektoren (**tmp_elements**) der **LogicalGroup**s können hier zurückgesetzt werden.

Datenelemente

- LogicalGroupVector **logical_groups**
- ElementVector **problem_elements**
- ElementVector **tmp_elements**

Konstruktoren, Destruktoren und Methoden

- **ProblemClass** (Integer sheetheight, Integer sheetwidth, Integer sheetdepth, Integer minoccupancy, Integer maxoccupancy, Text filename)
- **ProblemClass** (Problem problem)
- ∼**ProblemClass** ()
- Float **calcLowerBound** ()
- Integer **getSheetHeight** () const
- Integer **getSheetWidth** () const
- Integer **getSheetDepth** () const
- Integer **getMinOccupancy** () const
- Integer **getMaxOccupancy** () const
- void **beforeFirstLogicalGroup** ()
- Boolean **hasNextLogicalGroup** () const
- void **addLogicalGroup** (LogicalGroup)
- Element **getRandomElement** (Integer maxheight, Integer maxwidth)
- void **beforeFirstElement** ()
- Boolean **hasNextElement** () const
- void **addElement** (Element)
- void **sortElementsByHeight** ()
- void **sortElementsBySurface** ()
- void **resetElementsTmpRN** ()
- void **resetRemainingNumber** ()
- void **decreaseRemainingNumber** ()
- void **increaseRemainingNumber** ()
- Integer **getRemainingNumber** () const
- void **resetElementList** ()
- void **removeElement** (Order order)
- ElementVector::iterator **getElementVectorBegin** ()
- ElementVector::iterator **getElementVectorEnd** ()
- Integer **getElementVectorSize** ()
- Element **getElement** (Integer i)
- Integer **findElement** (Order o)
- LogicalGroup **getNextLogicalGroup** ()
- Element **getNextElement** ()
- Element **getFirstElement** ()
- Element **getLastElement** ()

B.6 LogicalGroupClass

Die Klasse **LogicalGroup** stellt eine logische Gruppe dar, welche Elemente enthält.

Ähnlich wie bei der Klasse **Element** gibt es auch hier zwei Zähler: **NumberOfElements** und **TmpNOE**. Dabei handelt es sich um die Anzahl der noch in der logischen Gruppe vorhandenen Elemente. Die Dekrementierung dieser Werte sollte nur von den Elmenten aus geschehen. Die Rücksetzfunktionen funktionieren so wie bei der Klasse **Element** und sollten nur von den das gesamte Problem erfassenden Rücksetzmethoden der Klasse **Problem**, verwendet werden.

Datenelemente

- ElementVector **logical_group_elements**
- ElementVector **tmp_elements**

Konstruktoren, Destruktoren und Methoden

- **LogicalGroupClass** (String id)
- **~LogicalGroupClass** ()
- Element **getRandomElement** (Integer maxheight, Integer maxwidth)
- void **beforeFirstElement** ()
- Boolean **hasNextElement** () const
- void **addElement** (Element)
- void **sortElements** ()
- String **getId** ()
- void **resetNumberOfElements** ()
- Integer **getRemainingNumberOfElements** () const
- void **decreaseElementNumber** ()
- void **resetTmpNOE** ()
- void **decreaseTmpNOE** ()
- Integer **getTmpNOE** ()
- void **resetElementVector** ()
- void **removeElement** (Order order)
- Element **getNextElement** ()

B.7 ElementClass

Die Klasse **Element** stellt ein Element dar und speichert seine Maße und Anzahl. Im Konstruktor werden alle Werte, wie Höhe, Breite, Anzahl usw. gesetzt. Elemente können auch

gedreht dargestellt werden. Dabei wird das Element verdoppelt und gedreht. Die beiden Elemente zeigen aufeinander, somit kann die Anzahl bei beiden vermindert (bzw. zurückgesetzt) werden.

Des weiteren besitzt jedes **Element** zwei Zähler: **RemainingNumber** und **TmpRN**. **RemainingNumber** kann vermindert und auf den Wert von **elementnumber** zurückgesetzt werden. **TmpRN** hingegen kann auf den Wert von **RemainingNumber** zurückgesetzt werden. Das Zurücksetzen der Zähler sollte nur mit den Rücksetzmethoden der Klasse **Problem** erfolgen, da so garantiert werden kann, dass alle **Elemente** zurückgesetzt werden. Die Zähler vermindern auch, wenn **RemainingNumber** bzw. **TmpRN** null werden, die korrespondierenden Zähler **NumberOfElements** und **TmpNE** der **LogicalGroup**, der sie angehören. Außerdem entfernt sich ein **Element** selbst aus den Elementvektoren (**tmp_elements**) von **Problem** und **LogicalGroup**, denen es angehört. Diese Vektoren können über eine Rücksetztmethode der Klasse **Problem** (siehe Abschnitt B.5 Methode: **resetElementVector**) wieder hergestellt werden.

Die Klasse beinhaltet auch zwei verschiedene Vergleichsoperatoren (Länge, Fläche), die dem **sort**-Algorithmus der **stl** übergeben werden können um Elementvektoren zu sortieren.

Konstruktoren, Destruktoren und Methoden

- **ElementClass** (String elementid, Integer elementheight, Integer elementwidth, Integer elementnumber, Boolean elementturning, LogicalGroup lg, Problem problem)
- **ElementClass** (Element element, Problem problem)
- ∼**ElementClass** ()
- String **getId** () const
- Integer **getElementNumber** () const
- Integer **getElementHeight** () const
- Integer **getElementWidth** () const
- Boolean **getElementTurning** () const
- LogicalGroup **getLogicalGroup** () const
- Problem **getProblem** () const
- Boolean **isTurned** () const
- Element **createElementDouble** ()
- Element **getElementDouble** ()
- Boolean **hasElementDouble** () const
- Boolean **isElementDouble** () const
- Integer **getRemainingNumber** () const
- void **decreaseRemainingNumber** ()
- void **resetRemainingNumber** ()
- Integer **getTmpRN** ()
- void **resetTmpRN** ()
- void **decreaseTmpRN** (Integer dec)
- Boolean **ElementSmallerHeight** (const Element, const Element)
- Boolean **ElementSmallerSurface** (const Element, const Element)

B.8 SolutionClass

Die Klasse **Solution** stellt eine Lösung des Problems dar, sie enthält einen oder mehrere gefüllte Rohlinge (**CuttingTree**).

Datenelemente

- CuttingTreeVector **solution_sheets**

Konstruktoren, Destruktoren und Methoden

- **SolutionClass** ()
- ~**SolutionClass** ()
- void **beforeFirstSheet** ()
- Boolean **hasNextSheet** () const
- void **addSheet** (CuttingTree)
- Integer **getNumberOfSheets** () const
- CuttingTree **getNextSheet** ()
- CuttingTree **getLastSheet** ()

B.9 CuttingTreeClass

Die Klasse **CuttingTree** wird zur Darstellung der Schnittmuster der Rohlinge benutzt. Die Klasse stellt auch Methoden zur Verfügung, welche den Verschnitt berechnen (**calculateWastePercentage** und **calculateLastWastePercentage**). Damit kann für den letzten Rohling der Verschnitt so berechnet werden, dass die Fläche unter dem letzten horizontalen Schnitt nicht berücksichtigt wird.

Datenelemente

- CuttingTreeVector **sub_trees**

Konstruktoren, Destruktoren und Methoden

- **CuttingTreeClass** ()
- **CuttingTreeClass** (CuttingTreeClass &ct)
- **CuttingTreeClass** (Problem problem)
- **CuttingTreeClass** (Element element, Boolean horizontalcut, Integer cutcoordinate, Integer wagon, Boolean wagonchange)
- ~**CuttingTreeClass** ()
- void **beforeFirstSubTree** ()

- Boolean **hasNextSubTree** () const
- void **addSubTree** (CuttingTree)
- void **delLastSubTree** ()
- Boolean **isHorizontalCut** () const
- void **setCutCoordinate** (Integer cutcoordinate)
- Integer **getCutCoordinate** () const
- void **setWagonChange** (Boolean wagonchange)
- Boolean **getWagonChange** () const
- Integer **getWagon** () const
- Float **getWastePercentage** () const
- void **calculateWastePercentage** ()
- void **calculateLastWastePercentage** ()
- CuttingTree **getNextSubTree** ()
- CuttingTree **getLastSubTree** ()
- Element **getElement** () const
- Problem **getProblem** () const

B.10 HeuristicClass

Die Klasse **Heuristic** ist eine abstrakte Klasse, die allen Heuristiken zugrunde liegt.

Basisklasse für **FFFClass**, **BFCClass**, **EAetClass**, **EAeClass**, **BBHEUClass** und **BBALGClass**.

Methoden

- virtual Solution **solveProblem** (Problem)=0

B.11 FFFClass

Die **FFF** Klasse ist eine Implementierung der in Abschnitt 3.2 beschriebenen FFF Heuristik für das 2BP-WR.

Die Rohlinge werden einer nach dem anderen befüllt, die Implementierung orientiert sich an der in Algorithmus 3.1 gegeben Beschreibung.

Abgeleitet von **HeuristicClass**.

B.12 BFCClass

Die **BFC** Klasse ist eine Implementierung der in Abschnitt 3.3 beschriebenen *Best Fit Cut* Heuristik.

Die Implementierung orientiert sich an Algorithmus 3.3. Die Rohlinge werden einer nach dem anderen generiert, wobei der jeweils aktuelle Abschnitt zufällig initialisiert und dann befüllt wird.

Abgeleitet von **HeuristicClass**.

B.13 BBHEUClass

Die Klasse **BBHEU** setzt die in Abschnitt 4.2 beschriebene BBHEU Heuristik um.

In einer ersten Phase werden mittels B&B einzelne Abschnitte erzeugt. Die Abschnitte werden in einer Liste gespeichert und Grenzabschnitte werden als solche markiert. Danach werden die Abschnitte innerhalb der Grenzabschnitte der Höhe nach sortiert. In der zweiten Phase werden die Abschnitte dann auf die Rohlinge platziert, wobei die Grenzen eingehalten werden.

Abgeleitet von **HeuristicClass**.

B.14 BBALGClass

Die Klasse **BBALG** ist eine Implementierung des in Abschnitt 4.3 beschriebenen Verfahrens. Es werden alle Rohlinge mittels B&B erzeugt. Während des B&B wird laufend der Status (Füllstand der Transportwagen sowie die logischen Gruppen) aktualisiert. Die einzelnen Subprobleme führen ihren Status mit, womit erreicht wird, dass die jeweiligen Füllstände nicht immer zurückgerechnet werden müssen, um die für das jeweilige Subproblem erlaubten Gruppen zu definieren.

Abgeleitet von **HeuristicClass**.

B.15 EAetClass

Die Klasse **EAet** implementiert das in Abschnitt 5.2.1 beschriebene Verfahren und verwendet dabei die **ealib** sowie das Chromosom **EAetChrom** (siehe Abschnitt B.18).

Abgeleitet von **HeuristicClass**.

B.16 EAeClass

Die Klasse **EAe** realisiert die in Abschnitt 5.2.3 und 5.2.4 beschriebenen Algorithmen EAe und EAeBB mit Hilfe der **ealib**. Je nachdem ob der Parameter `cut_useBB` gesetzt ist oder nicht, wird entweder **EAeChrom** oder **EAeBBChrom** (Abschnitte B.19 und B.20) verwendet.

Abgeleitet von **HeuristicClass**.

B.17 permChrom

Die Klasse **permChrom** ist eine auf der Klasse **Chromosome** der **ealib** basierende Klasse für Chromsome mit Permutationsdarstellung. Basisklasse für **EAetChrom** und **EAeChrom**.

Konstruktoren, Destruktoren und Methoden

- **permChrom** (const chromosome &c)
- **permChrom** (int l)
- virtual ∼**permChrom** ()
- virtual void **initialize** (int count)=0
- virtual void **copy** (const chromosome &orig)
- virtual bool **equals** (chromosome &orig)
- virtual double **dist** (chromosome &c)
- virtual void **write** (ostream &ostr, int detailed=0) const
- virtual void **mutate** (int count)
- virtual void **crossover** (const chromosome &parA, const chromosome &parB)
- double **hashvalue** ()

B.18 EAetChrom

Die Klasse **EAetChrom** ist die Chromosomklasse für den in Abschnitt 5.2.1 beschriebenen EAet Algorithmus. Als Mutation wird der Zweier-Austausch verwendet. Sie enthält zwei Crossovertypen: OX3 und PMX. Standardmäßig wird OX3 verwendet. Der Crossovertyp wird mit dem Parameter `cut_xtype` gesetzt.

Abgeleitet von **permChrom**.

Konstruktoren, Destruktoren und Methoden

- **EAetChrom** (const chromosome &c)
- ∼**EAetChrom** ()
- **EAetChrom** (int l)

- virtual chromosome * **createUninitialized** () const
- virtual void **copy** (const chromosome &orig)
- virtual void **initialize** (int count)
- virtual void **write** (ostream &ostr, int detailed=0) const
- virtual void **mutate** (int count)
- virtual void **crossover** (const chromosome &parA, const chromosome &parB)
- void **setProblem** (Problem prob)
- double **objective** ()
- Solution **getSolution** ()

B.19 EAeChrom

Die Klasse **EAeChrom** ist die Chromosomklasse für den in Abschnitt 5.2.3 beschriebenen EAe Algorithmus. Als Mutation kann der Zweier-Austausch, k-Block Austausch, Gruppierungsoperator oder eine Mischform der drei Mutationsoperatoren wie in Abschnitt 5.3 beschrieben verwendet werden. Der Mutationstyp wird mit dem Parameter `cut_mtype` gesetzt. Sie enthält drei Crossovertypen: OX3, PMX und GOX. Standardmäßig wird OX3 verwendet. Der Crossovertyp wird mit dem Parameter `cut_xtype` gesetzt.

Abgeleitet von **permChrom**.

Basisklasse für **EAebbChrom**.

Konstruktoren, Destruktoren und Methoden

- **EAeChrom** (const chromosome &c)
- **EAeChrom** (int l)
- ~**EAeChrom** ()
- virtual chromosome * **createUninitialized** () const
- virtual void **copy** (const chromosome &orig)
- virtual void **initialize** (int count)
- virtual void **write** (ostream &ostr, int detailed=0) const
- virtual void **crossover** (const chromosome &parA, const chromosome &parB)
- virtual void **mutate** (int count)
- void **setProblem** (Problem prob)
- virtual double **objective** ()
- virtual Solution **getSolution** ()

B.20 EAebbBBChrom

Die Klasse **EAebbChrom** ist die Chromosomklasse für den in Abschnitt 5.2.4 beschriebenen EAebb Algorithmus. Es handelt sich im Wesentlichen um eine Erweiterung der Klasse

EAeChrom, wobei die **objective** Funktion so abgeändert wurde, dass B&B mit einer gewissen Wahrscheinlichkeit (Parameter `cut_pBB`) zum Einsatz kommt und einen Abschnitt erzeugt. Die Lösung wird, falls B&B zum Einsatz kam, in das Chromosom zurückgeschrieben.

Abgeleitet von **EAeChrom**.

Konstruktoren, Destruktoren und Methoden

- virtual chromosome $*$ **createUninitialized** () const
- **EAebbChrom** (const chromosome &c)
- **EAebbChrom** (int l)
- double **objective** ()

Anhang C

Daten

Die Instanzen, welche für die Beispiele in der Diplomarbeit genutzt wurden, sind in diesem Appendix beschrieben.

real_21.cut

```
##############################
#                            #
# real 210802 083356         #
#                            #
##############################


# HEIGHT WIDTH DEPTH MIN_OCCUPANCY MAX_OCCUPANCY
5930 3140 8 200 250
# LOGICAL_GROUP
3820
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
421495 442 324 115 Y 2
# LOGICAL_GROUP
3841
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
421282 1003 1374 2 Y 2
# LOGICAL_GROUP
3482
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
421847 1622 657 4 Y 2
421850 748 1070 4 Y 2
421851 748 900 4 Y 2
421852 669 980 4 Y 2
421853 669 900 4 Y 2
# LOGICAL_GROUP
3483
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
421032 2500 1200 2 Y 2
421034 2500 1000 2 Y 2
421030 1920 1100 4 Y 2
```

```
421024 1000 1500 4 Y 2
# LOGICAL_GROUP
3844
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
423190 1414 2852 1 Y
# LOGICAL_GROUP
1
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
1 920 1355 1 Y
```

real_22.cut

```
##############################
#                            #
# real 210802 090417       #
#                            #
##############################


# HEIGHT WIDTH DEPTH MIN_OCCUPANCY MAX_OCCUPANCY
5880 3090 12 200 250
# LOGICAL_GROUP
3852
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
421855 1326 635 4 Y 2
421857 1330 635 5 Y 2
421856 1337 635 3 Y 2
421858 1334 635 5 Y 2
# LOGICAL_GROUP
3853
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
420008 2576 1320 1 Y 2
420012 1340 1320 1 Y 2
# LOGICAL_GROUP
3854
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
403232 1550 3135 1 Y 2
403233 1550 3135 1 Y 2
# LOGICAL_GROUP
2
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
1 1335 1508 5 Y 2
# LOGICAL_GROUP
3864
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
415731 1640 2865 8 Y 2
# LOGICAL_GROUP
3865
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
415735 1640 2865 8 Y 2
# LOGICAL_GROUP
3866
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
415739 1640 2865 8 Y 2
# LOGICAL_GROUP
```

```
3867
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
415743 1640 2865 9 Y 2
# LOGICAL_GROUP
3868
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
415747 1640 2865 9 Y 2
# LOGICAL_GROUP
3869
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
415751 1640 2865 9 Y 2
```

real_31.cut

```
##############################
#                            #
# real 210802 152048         #
#                            #
##############################


# HEIGHT WIDTH DEPTH MIN_OCCUPANCY MAX_OCCUPANCY
5882 3093 8 200 250
# LOGICAL_GROUP
3959
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
422501 1095 4279 5 Y 2
# LOGICAL_GROUP
3960
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
422551 1093 4279 9 Y 2
# LOGICAL_GROUP
3961
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
422640 1095 4193 3 Y 2
422650 1093 4193 5 Y 2
# LOGICAL_GROUP
3962
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
422589 1095 4159 3 Y 2
422660 1095 4153 3 Y 2
422600 1093 4159 5 Y 2
# LOGICAL_GROUP
3963
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
422670 1093 4153 4 Y 2
422620 1095 4099 3 Y 2
422630 1093 4099 4 Y 2
# LOGICAL_GROUP
3964
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
422681 1095 3648 5 Y 2
# LOGICAL_GROUP
3965
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
```

```
422695 1093 3648 9 Y 2
# LOGICAL_GROUP
3967
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
423049 720 4028 1 Y
423016 720 3674 1 Y
422942 720 3311 1 Y
423045 642 3669 1 Y
423046 642 3669 1 Y
422569 2982 717 9 Y 2
422616 2933 717 9 Y 2
422909 720 2949 1 Y
423012 642 3306 1 Y
# LOGICAL_GROUP
3968
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
423013 642 3306 1 Y
422938 642 2944 1 Y
422939 642 2944 1 Y
422803 720 2586 1 Y
422612 2725 639 9 Y 2
422613 2725 639 9 Y 2
422565 2660 639 9 Y 2
# LOGICAL_GROUP
3969
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
422566 2660 639 9 Y 2
422905 642 2582 1 Y
422906 642 2582 1 Y
422749 729 2224 1 Y
422799 642 2219 1 Y
422800 642 3319 1 Y
422762 720 1755 1 Y
422736 642 1857 1 Y
422737 642 1857 1 Y
422758 642 1388 1 Y
422759 642 1388 1 Y
# LOGICAL_GROUP
3970
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
423748 910 1510 4 Y 2
423744 835 1510 4 Y 2
423757 910 1310 2 Y 2
423752 835 1310 2 Y 2
423739 910 650 2 Y 2 2
423735 835 650 2 Y 2
```

real_36.cut

```
##############################
#                            #
# real 220802 082344         #
#                            #
##############################
```

```
# HEIGHT WIDTH DEPTH MIN_OCCUPANCY MAX_OCCUPANCY
5940 3150 10 200 250
# LOGICAL_GROUP
1
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
1 1538 1453 2 Y
# LOGICAL_GROUP
2
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
2 1538 1453 2 Y
# LOGICAL_GROUP
3
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
3 1539 1570 20 Y
# LOGICAL_GROUP
4
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
4 1539 1570 20 Y
# LOGICAL_GROUP
5
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
5 2540 1220 1 Y 2
# LOGICAL_GROUP
6
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
6 1244 3337 1 Y 2
# LOGICAL_GROUP
7
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
7 1124 1259 2 Y 2
# LOGICAL_GROUP
8
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
8 1032 2466 1 Y 2
# LOGICAL_GROUP
9
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
9 2152 3327 4 Y 2
```

real_37.cut

```
##############################
#                            #
# real 220802 085627         #
#                            #
##############################


# HEIGHT WIDTH DEPTH MIN_OCCUPANCY MAX_OCCUPANCY
5960 3170 4 200 250
# LOGICAL_GROUP
4144
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
1 932 468 1 Y
```

```
# LOGICAL_GROUP
1
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
2 1485 1201 3 Y
# LOGICAL_GROUP
2
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
3 715 472 5 Y 2
# LOGICAL_GROUP
3
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
4 933 23532 Y
# LOGICAL_GROUP
4
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
5 933 2530 6 Y
# LOGICAL_GROUP
5
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
6 707 517 1 Y
# LOGICAL_GROUP
6
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
7 1330 1372 2 Y
# LOGICAL_GROUP
7
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
8 1360 1372 8 Y
9 1330 1372 2 Y
# LOGICAL_GROUP
8
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
10 1393 1344 2 Y
# LOGICAL_GROUP
9
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
11 1364 1323 6 Y
# LOGICAL_GROUP
10
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
12 1331 1307 4 Y 2
13 1331 1307 1 Y 2
# LOGICAL_GROUP
11
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
14 1489 2489 1 Y
# LOGICAL_GROUP
12
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
15 1469 2547  2 Y
# LOGICAL_GROUP
13
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
16 1489 2498 1 Y
17 1479 960 1 Y
18 1518 960 1 Y
19 1533 2498 2 Y
```

```
# LOGICAL_GROUP
14
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
20 1503 2462 2 Y
21 1454 945 1 Y
22 1635 2462 4 Y
# LOGICAL_GROUP
15
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
23 1598 2433 2 Y
```

real_51.cut

```
##############################
#                            #
# real 230802 120352         #
#                            #
##############################


# HEIGHT WIDTH DEPTH MIN_OCCUPANCY MAX_OCCUPANCY
5882 3178 6 200 250
# LOGICAL_GROUP
4370
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
424563 1130 1081 11 Y
423740 960 651 11 Y
# LOGICAL_GROUP
1
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
1 537 862 27 Y
# LOGICAL_GROUP
2
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
2 658 145 2 Y 2
# LOGICAL_GROUP
3
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
3 1330 1822 2 Y
# LOGICAL_GROUP
4
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
4 1280 726 1 Y 2
# LOGICAL_GROUP
5
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
5 604 870 2 Y 2
# LOGICAL_GROUP
6
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
6 1359 913 1 Y
# LOGICAL_GROUP
7
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
7 1449 1093 1 Y 2
```

```
# LOGICAL_GROUP
8
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
8 665 3326 1 Y
# LOGICAL_GROUP
9
# ORDER WIDTH HEIGHT NUMBER TURNING(Y/N)
9 3178 5882 2 Y
```

Anhang D

Graphische Darstellung einiger Ergebnisse

In diesem Anhang sind die besten Ergebnisse für ausgewählte Instanzen abgebildet. Die Daten der dargestellten Instanzen sind in Appendix C aufgelistet.

real_22 Abbildung D.1 zeigt das von EAebb/GOX/0,01/-1 erzielte Ergebnis für die Instanz real_22. Das Verfahren erreicht einen Wert von $Z = 18,53$, das entspricht $98,53\%$ des von XOPTS erzielten Ergebnisses und $120,17\%$ der KUS.

real_31 Abbildung D.2 zeigt das von EAe/GOX/0,01/-1 erzielte Ergebnis für die Instanz real_31. Das Verfahren erreicht einen Wert von $Z = 28,62$, das entspricht 100% des von XOPTS erzielten Ergebnisses und $125,25\%$ der KUS.

real_36 Abbildung D.3 zeigt das von EAebb/GOX/0,01/-1 erzielte Ergebnis für die Instanz real_36. Das Verfahren erreicht einen Wert von $Z = 10,24$, das entspricht $96,97\%$ des von XOPTS erzielten Ergebnisses und $130,28\%$ der KUS.

real_51 Abbildung D.4 zeigt das von EAebb/GOX/0,01/-0,01 erzielte Ergebnis für die Instanz real_51. Das Verfahren erreicht einen Wert von $Z = 4,55$, das entspricht $99,34\%$ des von XOPTS erzielten Ergebnisses und $103,41\%$ der KUS.

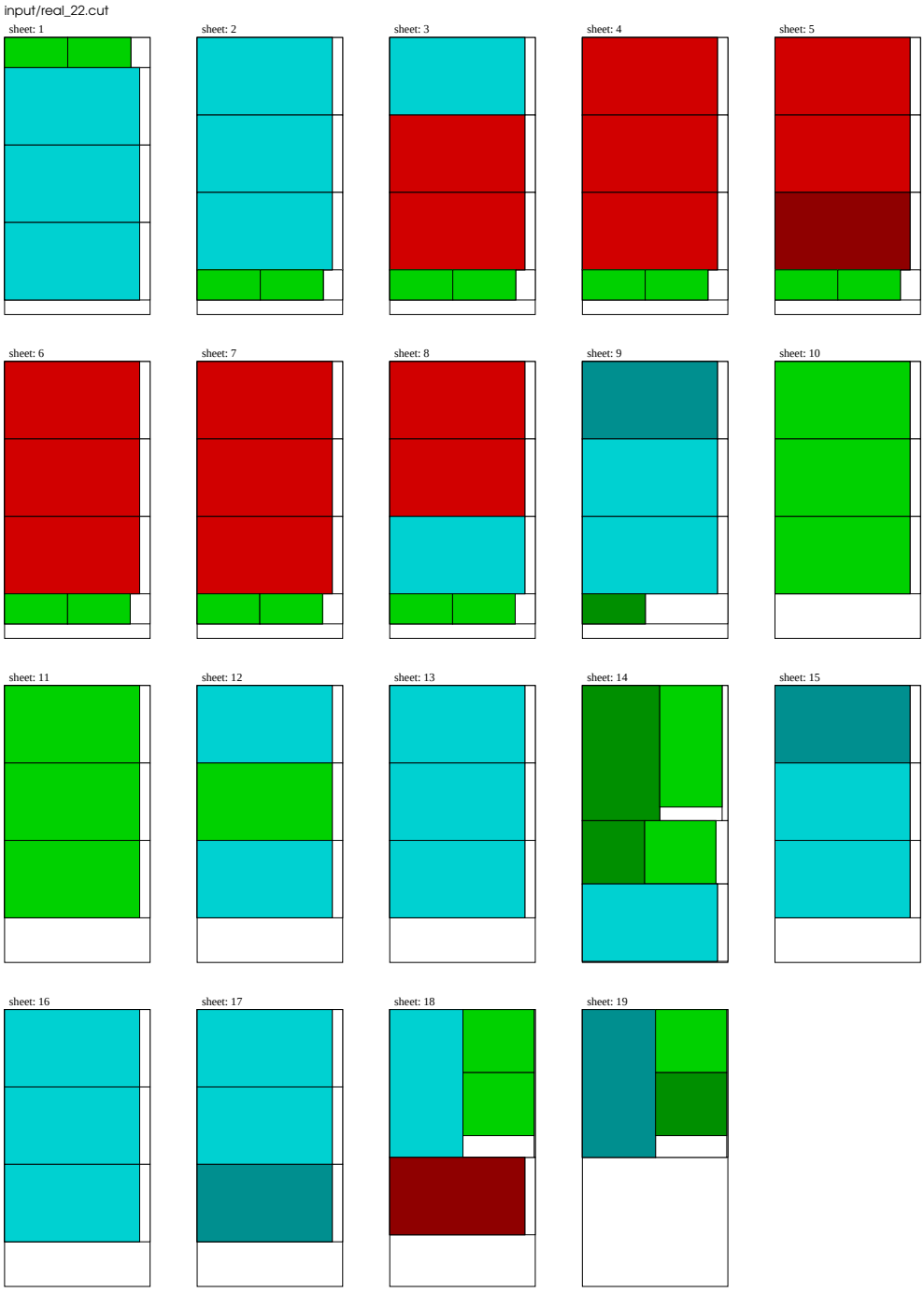


Abbildung D.1: EAebb/GOX/0,01/-1_real_22

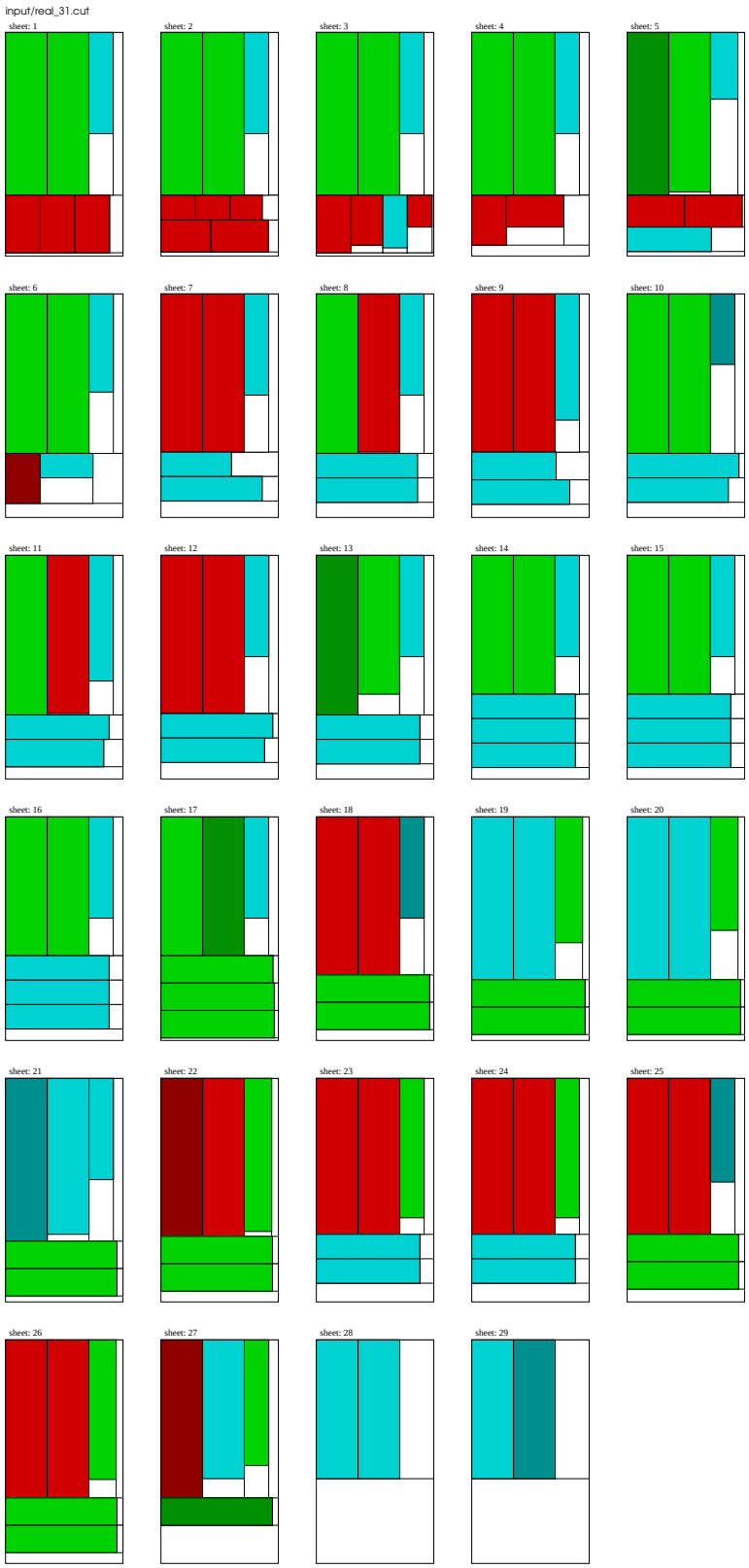


Abbildung D.2: EAe/GOX/0,01/-1_real_31

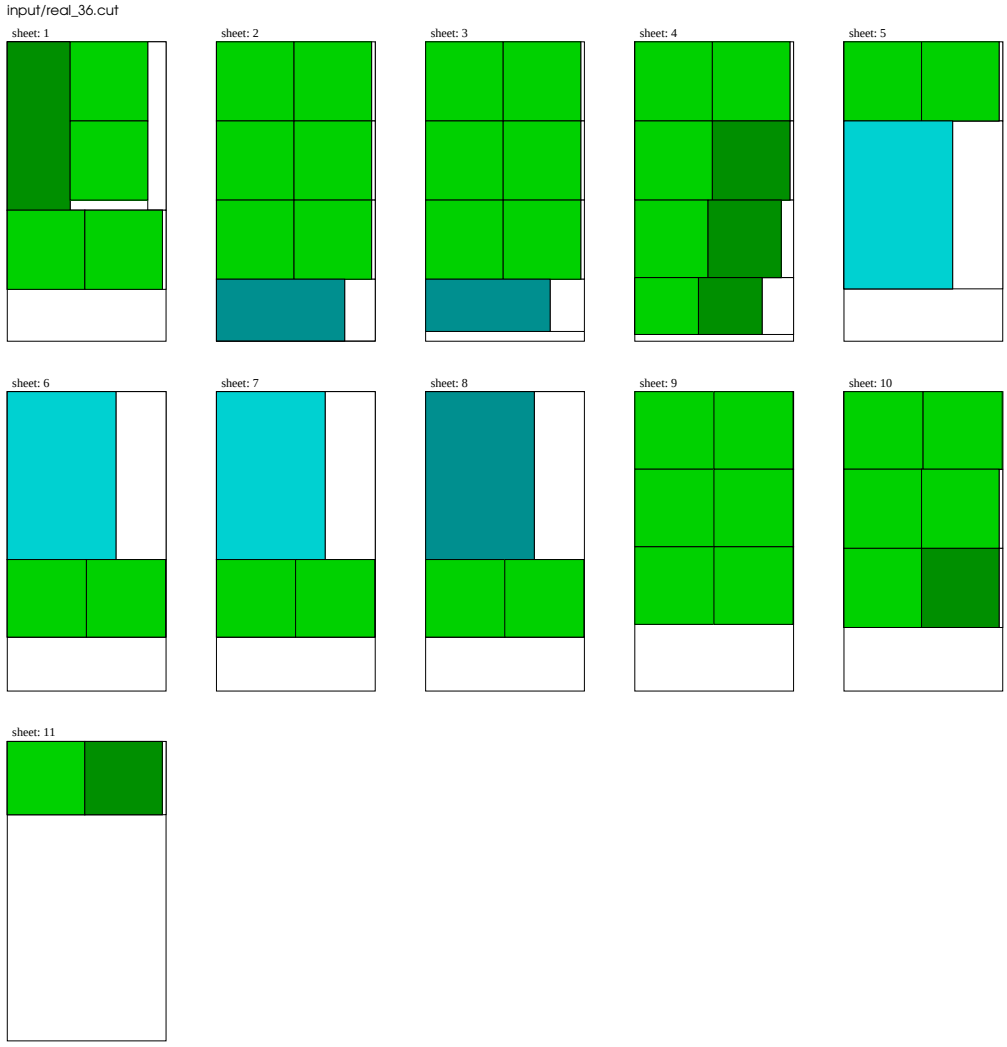


Abbildung D.3: EAebb/GOX/0,01/-1_real_36

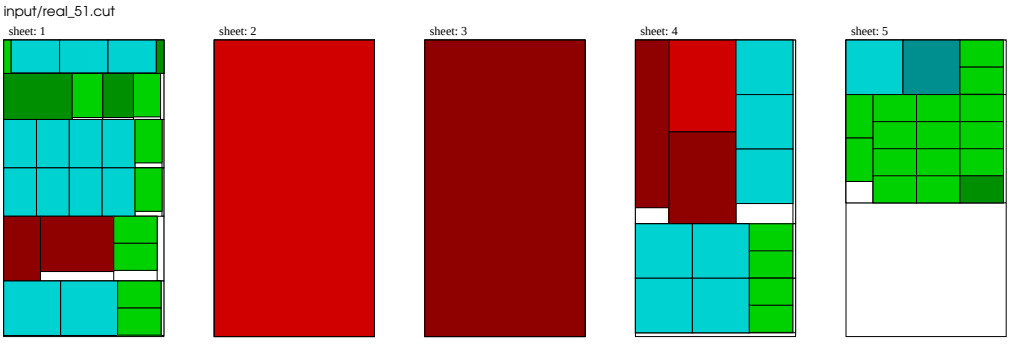


Abbildung D.4: EAebb/GOX/0,01/-0,01_real_51