Author postprint manuscript No.
(will be inserted by the editor)

Fast Algorithms for Fragmentable Items Bin Packing

Benjamin Byholm · Ivan Porres

the date of receipt and acceptance should be inserted later

Abstract Bin packing with fragmentable items is a variant of the classic bin packing problem where items may be cut into smaller fragments. The objective is to minimize the number of item fragments, or equivalently, to minimize the number of cuts, for a given number of bins. Models based on packing fragmentable items are useful for representing finite shared resources. In this article, we present improvements to approximation and metaheuristic algorithms to obtain an optimality-preserving optimization algorithm with polynomial complexity, worst-case performance guarantees and parametrizable running time. We also present a new family of fast lower bounds and prove their worst-case performance ratios. We evaluate the performance and quality of the algorithm and the best lower bound through a series of computational experiments on representative problem instances. For the studied problem sets, one consisting of 180 problems with up to 20 items and another consisting of 450 problems with up to 1024 items, the lower bound performs no worse than 5/6. For the first problem set, the algorithm found an optimal solution in 92 % of all 1800 runs. For the second problem set, the algorithm found an optimal solution in 99 % of all 4500 runs. No run lasted longer than 220 milliseconds.

Keywords Packing · Combinatorial optimization · Genetic algorithms

1 Introduction

Bin packing with fragmentable items is a variant of the classic bin packing problem where items may be cut into smaller fragments. The objective is to minimize the number of item fragments, or equivalently, the number of cuts, for a given number of bins. Models based on fragmentable items are useful for

Benjamin Byholm E-mail: bbyholm@abo.fi · Ivan Porres E-mail: iporres@abo.fi
Åbo Akademi University, Informationsteknologi, Domkyrkotorget 3, 20500 Turku, Finland
Turku Centre for Computer Science, Vesilinnantie 3, 20500 Turku, Finland

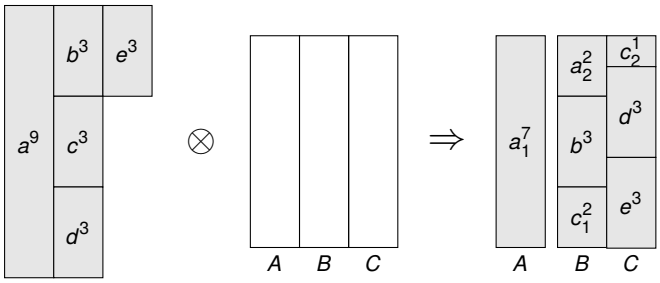

Fig. 1 Packing the items into three bins of size seven allowing fragmentation

representing finite shared resources. Fig. 1 illustrates how a group of items are packed into three bins of size seven.

An example of such a problem arises in the operation of file systems. A file system manages the physical storage of files with various sizes on a storage medium. Over time, some files may be removed, leaving gaps of unallocated space on the storage medium. As the storage medium fills up, there comes a time when the data in a file do not fit into a contiguous region and need to be divided into smaller fragments. As file system fragmentation increases, performance decreases, since sequential access becomes more unlikely. Therefore, it is desirable to minimize fragmentation.

Modern file systems attempt to avoid fragmentation by leaving unallocated space around new files, so that they have space to grow, and by reorganizing old files during normal operation. However, the former strategy introduces overhead in the utilization of available storage space and becomes less effective when the amount of contiguous free space decreases. The second option introduces overhead in throughput, since the storage device is preoccupied with reordering data. How can we minimize these overheads?

In this scenario, we can model files as items and contiguous regions of the storage medium as bins. Each bin has a possibly distinct capacity, corresponding to the size of the region, while each file has a possibly distinct size, quantified at the granularity of the system block size. By finding a good allocation of items to bins, which minimizes fragmentation, we may reduce the overhead in all of these aspects.

The classic bin packing problem is a related, well-known combinatorial optimization problem. Garey and Johnson (1990) noted that its decision form: "Can these items be packed into m bins?", is strongly $\mathcal{NP}$-complete. LeCun et al (2015) observed that this problem is a special case of bin packing with fragmentable items, i.e. "Can these items be packed into m bins with $k = 0$ cuts?", which makes the latter at least as hard as the former. Hence, any deterministic, exact algorithm requires time superpolynomial in the size of input, unless $\mathcal{P} = \mathcal{NP}$. Problems of the optimization form are at least as hard.

In this article, we present an optimality-preserving approximation algorithm (it can always reach an optimal solution) with polynomial complexity, worst-case performance guarantees and parametrizable running time. Its design combines

carefully chosen approximation algorithms with a suitable metaheuristic in the form of a grouping genetic algorithm. We also evaluate its performance through a series of computational experiments on representative problem instances using an actual implementation written in C++. The source code of the implementation Byholm (2017a), the problem sets and their optimal values Byholm (2017b) as well as the obtained results Byholm (2017c) are publicly available and can be used to reproduce the results presented in this article as well as to evaluate new algorithms.

2 Related Work

As previously mentioned, bin packing is a classic combinatorial optimization problem which has been studied for decades. Some forms of it have even dealt with fragmentable items, but it was not until recently that LeCun et al (2015) formalized the problem and presented a family of approximation algorithms explicitly trying to minimize the number of fragments for a given set of bins and items. Our work builds on these results and reduces the computational complexity of the approximation algorithms.

Bin packing BP closely resembles Min-FIBP (Minimum Fragmentable Items Bin Packing) and Min-FIBP-EQ (Minimum Fragmentable Items Bin Packing with Equal Capacities). However, LeCun et al (2015) note that solutions to the corresponding optimization problems are very different and that approximation algorithms which work well for BP, like FFD (First Fit Decreasing), yield unsatisfactory solutions for Min-FIBP-EQ. This is because filling a bin nearly completely is a good result for BP, since minimizing wasted space results in fewer required bins, but it is generally a bad strategy for Min-FIBP-EQ, since it results in plenty of fragmentation. LeCun et al (2015) further note that known variants of BP which allow cutting items into smaller fragments at a cost may produce bad solutions for Min-FIBP-EQ, such as a problem for circuit design by Mandal et al (1998) where each cut introduces an overhead and the objective is to minimize the number of bins required.

In a recent paper, Casazza and Ceselli (2016) present more results on bin packing with fragmentable items under the name BPPIF (Bin Packing Problem with Item Fragmentation), expanding on Casazza and Ceselli (2014). Here, the problem that LeCun et al (2015) called Min-FIBP is known as fm-BPPSPF (Fragmentation-Minimization Bin Packing Problem with Size-Preserving Fragmentation). Casazza and Ceselli (2016) also present exact solution algorithms for many problem variants. Unfortunately, due to the hardness of the problem, exact algorithms do not scale well. The presented computational experiments involve problem instances with few items. Our focus in this article is on approximations and heuristic algorithms that produce high quality for thousands of items in fractions of a second on a standard computer.

Our work uses a metaheuristic known as a grouping genetic algorithm to approximately solve large problem instances.The grouping genetic algorithms are a family of genetic algorithms different from the traditional, Holland-style

genetic algorithms, more applicable to grouping problems. They were first proposed by Falkenauer (1992) 25 years ago. Falkenauer (1996) found that a certain form of grouping genetic algorithms produce results far better than those obtained from their underlying heuristics in isolation. Based on this, Quiroz Castellanos et al (2015) recently presented a new grouping genetic algorithm for the classic bin packing problem BP. However, to the best of our knowledge, no genetic algorithms or other metaheuristic approaches for MIN-FIBP or MIN-FIBP-EQ exist in the literature. Our work builds on the grouping genetic algorithm presented by Quiroz Castellanos et al (2015), developing a more efficient genetic representation for all variants of bin packing in general, while adapting the genetic operators for MIN-FIBP-EQ in particular.

3 Problem Description

Bin packing with fragmentable items is a variant of the classic bin packing problem BP, where items $i \in \mathcal{I}$ having integer size s_i may be cut into smaller, integer-sized fragments. Each cut introduces an additional fragment and each additional fragment requires a cut. We wish to minimize the number of fragments, or equivalently, the number of cuts for a given number of bins $j \in \mathcal{B}$ having capacity c_j. LeCun et al (2015) concluded that the decision problem FIBP (Fragmentable Items Bin Packing) is a generalization of the classic bin packing problem and therefore strongly $\mathcal{NP}$-complete. An exact algorithm for solving the optimization problem MIN-FIBP directly solves the decision problem FIBP. Hence, MIN-FIBP is at least as hard as FIBP.

3.1 Slack

The approximation algorithms presented by LeCun et al (2015) only work for the special case of MIN-FIBP where the items completely fill the available capacity. To make the approximation algorithms work with general cases of MIN-FIBP, LeCun et al (2015, Property 1) proposed transforming general problem instances into the special case by adding dummy items of size 1 to fill out the unused space. This requires adding $s = \sum_{j \in \mathcal{B}} c_j - \sum_{i \in \mathcal{I}} s_i$ dummy items of size 1 to the set of items $\mathcal{I}$ to produce a new set of items $\mathcal{I}'$. However, the described transformation actually has complexity $\mathcal{O}(n + s)$, which is not polynomial in the size of the problem, but pseudo-polynomial, since s is not bounded by n. For instance, if the total capacity of the bins $\sum_{j \in \mathcal{B}} c_j = \mathcal{O}(2^n)$, the described transformation also has complexity $\mathcal{O}(2^n)$, which is exponential.

We circumvent this problem by adding $|\mathcal{B}| - 1$ dummy items of size 0 and using the derived property slack s, as defined above. When $s = 0$, we have the special case of MIN-FIBP where the sum of item sizes is equal to the aggregated bin capacity. A general instance of MIN-FIBP is then defined by the set $\mathcal{I}$ of items and the set $\mathcal{B}$ of bins. For convenience, we also include the derived property slack s in our representation. We shall denote this as $(\mathcal{I}, \mathcal{B}, s)$.

3.2 Blocks

LeCun et al (2015) developed the notion of blocks, which can be summarized as follows: A block is a subset of items (and slack units) together with a subset of bins which are completely filled by said items (and slack units). An instance of Min-FIBP or Min-FIBP-EQ can itself be regarded as a block. The size of a block is the number of items and slack units it contains. A minimal block is a block which cannot be partitioned into smaller blocks. A key observation is that we can describe a solution by its minimal blocks, but a solution composed of minimal blocks is not necessarily optimal. In Corollary 2, LeCun et al (2015) proved that minimizing the number of fragments is equivalent to maximizing the number of (minimal) blocks. An optimal solution therefore has a maximum number of (minimal) blocks. The score of a block is the number of fragments and slack units it contains. A low score is better than a high score and a block with no items has an infinitely high score.

3.3 Non-Uniform Bins

Min-FIBP-EQ is a special case of Min-FIBP, where all the bins have equal capacity c. In Property 5, LeCun et al (2015) proved that any instance of Min-FIBP can be polynomially reduced into an instance of Min-FIBP-EQ. Rewriting this property to use our revised representation of Min-FIBP, we have that any instance $(\mathcal{I}, \mathcal{B}, s)$ of Min-FIBP can be polynomially reduced into an instance $(\mathcal{I}', \mathcal{B}', s)$ of Min-FIBP-EQ, with $|\mathcal{B}'| = |\mathcal{B}|$ (same number of bins) and $|\mathcal{I}'| = |\mathcal{I}| + |\mathcal{B}|$ (one additional item per bin). We realize this transformation by choosing a value $Z > \sum_{j \in \mathcal{B}} c_j$, and considering this value Z as the unique capacity of the $|\mathcal{B}|$ bins of $\mathcal{B}'$. We create the set $\mathcal{I}'$ by adding an item of size $Z - c_j$ for each bin $j \in \mathcal{B}$ to the original set of items $\mathcal{I}$. Since we only add $|\mathcal{B}|$ items, this transformation is strongly polynomial.

3.4 Large Items

When all the bin capacities are equal, $\forall j \in \mathcal{B}\colon c_j = c$, we have an instance of Min-FIBP-EQ. In Property 6, LeCun et al (2015) proved that there always exists an optimal solution where items larger than c are cut into some number of fragments of size c and a possible remaining fragment of size strictly less than c. So, without loss of generality, we will only consider instances where all items are smaller than c.

4 Approximation Algorithms

In this section we present approximation algorithms for Min-FIBP-EQ and discuss their worst-case computational complexities and performance. As mentioned in Section 3.2, a solution to a Min-FIBP-EQ-instance can be regarded

as a collection of (minimal) blocks of various sizes and we wish to maximize the number of (minimal) blocks which encode the solution. To this end we can start with algorithm $\mathcal{E}_1$, which exactly finds all blocks of size 1, followed by algorithm $\mathcal{E}_2$, which exactly finds all blocks of size 2. This can be regarded as the composition $\mathcal{E}_1\mathcal{E}_2$. When composed with an algorithm for approximately finding all blocks of size 3 or greater, this yields an improved approximation algorithm for MIN-FIBP-EQ.

4.1 $\mathcal{E}_1$ - Perfectly Fitting Items

In Property 2, LeCun et al (2015) proved that there is at least one optimal solution to any instance of MIN-FIBP, wherein any item i with size s_i is placed into bin j if the item perfectly fills the bin, $s_i = c_j$. Such items and bins can be removed from further consideration. They correspond to blocks of size 1.

Since any instance of MIN-FIBP can be reduced into an instance of MIN-FIBP-EQ, we need only consider instances of the latter. Under these conditions, all perfectly fitting items have size $s_i = c$, where c is the capacity of each bin, and LeCun et al (2015) presented the exact algorithm $\mathcal{E}_1$, which finds all blocks of size 1 by iterating over the set of items and packing each item of size $s_i = c$ into its own bin. The algorithm is exact, since it finds every block of size 1. It has complexity $\mathcal{O}(|\mathcal{I}|) = \mathcal{O}(n)$.

We may also observe that without perfectly fitting items, the number of bins in a block of size k is at most $k - 1$.

Lemma 1 *For any instance of* MIN-FIBP-EQ*, if there are no perfectly fitting items, i.e. of size c, the number of bins in a block of size k is at most $k - 1$.*

Proof Since there are no items of size c, the size of an item is at most $c - 1$. A slack unit has size $1 < c$. By definition, the bins in a block are fully used by items and slack. Since k items and slack cannot fill k bins, $k(c - 1) < kc$, at most $k - 1$ bins are used. □

4.2 $\mathcal{E}_2$ - Perfectly Fitting Pairs of Items

In Property 3, LeCun et al (2015) proved that for any instance of MIN-FIBP-EQ, if the sizes of two items k and l sum to c, there is at least one optimal solution wherein both items are placed in the same bin. Thus, such items and bins can be removed from further consideration. LeCun et al (2015) presented the exact algorithm $\mathcal{E}_2$, which achieves this by matching. We have adapted it to work with general instances of MIN-FIBP-EQ by accounting for slack. Algorithm $\mathcal{E}_2$ can be implemented as follows:

1. Sort the items in non-decreasing order of size
2. Create two indices l and r corresponding to the beginning and end of the array
3. While $l < r$

(a) If the sizes of the items at l and r sum to c, pack them in a bin, increment l and decrement r
(b) If the item at r has size $c-1$ and there is slack, pack the item in a bin, decrement s and r
(c) If the sum is smaller than c, increment l
(d) If the sum is greater than c, decrement r

It has complexity $\mathcal{O}(|\mathcal{I}|\log|\mathcal{I}|) = \mathcal{O}(n\log n)$, since the sorting step dominates.

To illustrate the workings of the algorithm, consider a problem instance consisting of items with sizes $[2,3,4,4,5,5,7]$, capacity $c=8$ and slack $s=2$. The first case does not apply, since $2+7 \neq 8$, so we proceed to the next case, which is the key modification which accounts for possible slack in the problem instance. Had we performed the transformation described in Section 3.1, we would instead have started with the array $[1,1,2,3,4,4,5,5,7]$ and no slack, then the first case would have applied, as we would have started with $1+7=8$. So, since an item of size $c-1=7$ would have formed a pair with a dummy item, we now need to account for this case. Eliminating the pair leaves us with a search space of $[2,3,4,4,5,5]$ and 1 unit of slack left. Since $2+5<8$, the sum needs to increase, which requires incrementing l. Now the search space is $[3,4,4,5,5]$ and we process the pair $3+5=8$, leaving $[4,4,5]$. Because $4+5>8$, we need the sum to shrink. We decrement r, leaving $[4,4]$, which sum to 8.

4.3 A Linear Approximation

In this section we present a strongly polynomial 2-approximation algorithm for the general case of Min-FIBP-EQ. LeCun et al (2015) already presented a greedy 2-approximation algorithm with complexity $\mathcal{O}(|\mathcal{I}|) = \mathcal{O}(n)$ called $\mathcal{G}$ for the special case of Min-FIBP-EQ where the sum of item sizes is equal to the aggregated bin capacity. Algorithm $\mathcal{G}$ is a simple variation of next fit: Fill the bins $\mathcal{B}$ with items chosen in an arbitrary order from $\mathcal{I}$. If an item does not fit into the remaining space of a bin, cut it into two fragments. Complete the current bin with the first fragment and place the other fragment in a new bin.

As mentioned in Section 3.1, LeCun et al (2015) proposed handling the general case of Min-FIBP-EQ by adding s dummy items of size 1. The key idea behind our algorithm is that instead of adding s dummy items, we add $|\mathcal{B}|-1$ dummy items of size 0 acting as border items which are subsequently ignored. Blocks with dummy items can use slack.

We have developed a greedy 2-approximation algorithm with complexity $\mathcal{O}(|\mathcal{I}|+|\mathcal{B}|-1) = \mathcal{O}(n)$ called $\mathcal{G}^+$ for the general case of Min-FIBP-EQ. The pseudocode is given in Fig. 2. The algorithm takes as input a set $\mathcal{I}$ of items (and dummy items), an integer s indicating how many units of slack are available, and an integer c for the bin capacity. It outputs a set of blocks.

We start by constructing a new current block b, and setting the slack indicator q to 0. The algorithm then iterates over a random permutation of the set of items and dummy items $\mathcal{I}$. If the current item i is a dummy item, we set the slack indicator q to 1. Else, if the current item i does not fit in the current

```
1: function G+(I, s, c)
2:     b ← Block(c)
3:     q ← 0
4:     for all i ∈ I do
5:         if i.size = 0 then
6:             q ← 1
7:         else
8:             if i.size > b.slack ∧ q × s ≥ b.slack then
9:                 q ← 0
10:                s ← s − b.slack
11:                yield b
12:                b ← Block(c)
13:            end if
14:            b ← b ∪ {i}
15:        end if
16:    end for
17:    r ← (s − b.slack)/c
18:    yield b
19:    for j ← 0, r − 1 do
20:        yield Block(c)
21:    end for
22: end function
```

Fig. 2 Algorithm $\mathcal{G}^+$ makes blocks from a random permutation of a set of items (and dummy items) $\mathcal{I}$ together with s slack units. Block(c) constructs a block with capacity c. Encountered dummy items allow blocks to use slack, but they are not included in the actual blocks produced by the algorithm

block b, but there is enough slack available, we reset the slack indicator q to 0, decrease the amount of available slack s, yield the current block b and create a new block. Then we add the current item i to the current block b. After all items have been packed, we yield the last proper block b and a number of empty blocks according to leftover slack s.

The proposed algorithm $\mathcal{G}^+$ is at least as good as the original algorithm $\mathcal{G}$, and is therefore a 2-approximation algorithm with complexity $\mathcal{O}(n)$ for the general case of Min-FIBP-EQ. The original algorithm $\mathcal{G}$ also had the additional property of optimality preservation, meaning that there always exists a permutation of the set of input items reaching the optimal packing. This is highly desirable when using a heuristic as part of a genetic algorithm, since we desire the chance to eventually reach an optimal solution. We can see that $\mathcal{G}^+$ also possesses this property, since it is functionally equivalent to $\mathcal{G}$.

As mentioned in Section 4.1, the quality of the packing can be improved by first packing all perfectly fitting items of size c through the exact algorithm $\mathcal{E}_1$. The composition $\mathcal{E}_1\mathcal{G}^+$ results in a 3/2-approximation, since LeCun et al

(2015) proved that $\mathcal{E}_1\mathcal{G}$ yields a 3/2-approximation. A better approximation is obtained by also accounting for perfectly fitting pairs of items through the exact algorithm $\mathcal{E}_2$, as mentioned in Section 4.2. The composition $\mathcal{E}_1\mathcal{E}_2\mathcal{G}^+$ yields a 4/3-approximation, since LeCun et al (2015) proved that $\mathcal{E}_1\mathcal{E}_2\mathcal{G}$ yields a 4/3-approximation.

4.4 A Quadratic Approximation

We can obtain a better approximation by finding blocks of size 3. LeCun et al (2015) presented a 1/3-approximation algorithm $\mathcal{A}_3$, with complexity $\mathcal{O}(n^4)$ for this problem. This algorithm is for the special case of MIN-FIBP-EQ where the sum of item sizes is equal to the aggregated bin capacity. It is based on k-set packing, attempting to construct a maximal number of disjoint subsets of items with cardinality k by constructing subsets and successively attempting to improve the packing. Based on algorithm $\mathcal{A}_3$, we have constructed algorithm $\mathcal{B}_3$ for the general case of MIN-FIBP-EQ with the same approximation ratio but complexity $\mathcal{O}(n + |\mathcal{W}|^2) = \mathcal{O}(n^2)$, where $\mathcal{W}$ is the set of item sizes.

Algorithm $\mathcal{B}_3$, presented in Fig. 3, looks for an optimal set of blocks with size 3. It takes as input a set $\mathcal{I}$ of items, an integer s for the available amount of slack and an integer c representing bin capacity. It returns a set $\mathcal{A}$ of blocks of size 3. According to Lemma 1, a block of size 3 consists of 1 or 2 bins. Therefore, such a block is equivalent to an integer 3-partition of c or $2c$. So, by computing the 3-partitions of c and $2c$, we can generate all possible blocks of size 3. The 3SUM problem, presented by Gajentaan and Overmars (2012), asks if a collection of non-negative integers contain 3 elements that sum to some value v, or equivalently, if the items contain a 3-partition of v. By solving the 3SUM-problem, as shown by Gajentaan and Overmars (2012), we can compute the 3-partitions in $\mathcal{O}(|\mathcal{W}|^2)$ for sorted inputs, where $\mathcal{W}$ is the set of sizes among items in $\mathcal{I}$. The algorithm works by testing all pairs of items in an order which avoids the need for binary search.

We begin by initializing the return value $\mathcal{A}$ to the empty set. We then call the function 3SUM to obtain an array $\mathcal{P}$ of 3-partitions of c and $2c$, corresponding to blocks of size 3. We then initialize an index r to the size of the obtained partition array. While r is greater than 0 and there are unpacked items, we pick a random partition from $\mathcal{P}$ using the discrete uniform distribution $\mathcal{U}\{0, r-1\}$, check that it is feasible with regard to the available items and slack, and add it to the return value while reducing available items and slack. If the chosen partition is not feasible, we swap it with the partition at $\mathcal{P}[r-1]$ and decrement r. When no possible partitions remain, we return the chosen set $\mathcal{A}$. To guarantee a 1/3-approximation, $\mathcal{B}_3$ always produces a maximal set of 3-partitions and since $\mathcal{B}_3$ picks partitions randomly, it is optimality preserving when it comes to finding a maximum set of 3-partitions. However, there are instances of MIN-FIBP-EQ where no optimal solution has a maximal set of 3-partitions. Hence, $\mathcal{B}_3$ is not optimality preserving for MIN-FIBP-EQ. However, when $\mathcal{B}_3$ is used in an iterative algorithm, such as a grouping genetic algorithm,

```
1: function B_3(I, s, c)
2:     A ← ∅
3:     P ← 3SUM(I, s, c) ∪ 3SUM(I, s, 2 × c)
4:     r ← |P|
5:     while r > 0 do
6:         i ← U{0, r − 1}
7:         if P[i] ⊆ I ∧ P[i].slack ≤ s then
8:             A ← A ∪ {P[i]}
9:             I ← I \ P[i]
10:            s ← s − P[i].slack
11:        else
12:            SWAP(P[i], P[r − 1])
13:            r ← r − 1
14:        end if
15:    end while
16:    return A
17: end function
```

Fig. 3 Algorithm $\mathcal{B}_3$ searches for an optimal set of blocks with size 3. Its input is a set of items $\mathcal{I}$, the available amount of slack s and the bin capacity c. It returns a set $\mathcal{A}$ of blocks of size 3. The function $3\textsc{sum}(\mathcal{I}, s, v)$ returns the possible 3-partitions of v given $\mathcal{I}$ and s, $\mathcal{U}\{a, b\}$ gives a discrete uniform variate on $[a, b]$ and $\textsc{swap}(x, y)$ swaps the elements at x and y

we can remedy this problem by destroying some of the blocks produced by $\mathcal{B}_3$ on subsequent iterations, after producing an initial population, and packing their items with an optimality-preserving algorithm, such as $\mathcal{G}^+$. However, to ensure the approximation guarantee, $\mathcal{B}_3$ must run normally at least once.

We cannot create more than $\mathcal{O}(n)$ blocks and there are only $\mathcal{O}(|\mathcal{W}|^2)$ potential types of blocks, so the algorithm runs in $\mathcal{O}(n + |\mathcal{W}|^2) = \mathcal{O}(n^2)$. Since LeCun et al (2015) proved that the composition $\mathcal{E}_1\mathcal{E}_2\mathcal{A}_3\mathcal{G}$ gives a 5/4-approximation with complexity $\mathcal{O}(n^4)$, the composition $\mathcal{E}_1\mathcal{E}_2\mathcal{B}_3\mathcal{G}^+$ gives a 5/4-approximation with complexity $\mathcal{O}(n^2)$.

5 Lower Bounds

We present lower bounds for Min-FIBP-EQ and analyze their worst case computational complexities and performance, using the standard definition of performance guarantees, here given by Fekete and Schepers (2001):

Definition 1 (Worst-Case Performance) Let L be a lower bound for a problem P. Let $\mathrm{OPT}(I)$ denote the optimal value of P for an instance I. The worst-case performance of L is given by

$$r(L) \triangleq \sup \left\{ \frac{L(I)}{\mathrm{OPT}(I)} \;\middle|\; I \text{ is an instance of } P \right\}$$

and the asymptotic worst-case performance of L is given by

$$r_\infty(L) \triangleq \limsup_{s\to\infty} \left\{ \frac{L(I)}{\mathrm{OPT}(I)} \;\middle|\; I \text{ is an instance of } P \text{ with } \mathrm{OPT}(I) \geq s \right\}$$

Since the classic bin packing problem BP is a special case of MIN-FIBP-EQ where no cuts are allowed, any lower bound for the former can be converted into a lower bound for the latter. This is done by taking the difference between the lower bound on bins required for BP and the number of bins $|\mathcal{B}|$ available in MIN-FIBP-EQ.

Fekete and Schepers (2001) derived a class of bounds $L_*^{(p)}$ for BP with worst case performance ratio 2/3, having complexity $\mathcal{O}(pn)$ for sorted inputs, where n is the number of items and p is an arbitrary parameter signifying the number of iterations of the main loop of the algorithm. We can roughly outline how the lower bound works: It divides the set of items into subsets based on their sizes and then computes a lower bound on the number of bins required based on this information. For example, no item $i \in \mathcal{I}$ with size $s_i > c/2$ can be combined with another such item without an ensuing cut. Thus, we know that at least this many bins are necessary to have no cuts. It repeats this process in a specific manner to tighten the ensuing bound by neglecting all items smaller than a successively increasing size threshold and processing the item sizes with a step function at quantization levels between 2 and p. The *asymptotic* performance ratio of this bound is 3/4, but Fekete and Schepers (2001) noted that it has been proven via a reduction from PARTITION, that no lower bound for BP can achieve a worst case performance ratio better than 2/3 unless $\mathcal{P} = \mathcal{NP}$. We derive a lower bound for the MIN-FIBP-EQ problem:

$$L_0^{(p)} = |\mathcal{I}| + L_*^{(p)} - |\mathcal{B}|.$$

Lemma 2 *$L_0^{(p)}$ is a lower bound for* MIN-FIBP-EQ*.*

Proof Any bin used in $L_*^{(p)}$ contains at least one item which does not fit in any other bin. Thus, there will be at least one cut item for each available bin less than this. □

In practice, we exclusively use $L_0^{(20)}$. According to Fekete and Schepers (2001), it offers a good trade-off between quality and speed.

Unfortunately, the worst case performance ratio of $L_0^{(p)}$ is arbitrarily bad, since we may have more bins available than the lower bound for BP. However, it takes negligible time to compute and performs rather well in practice. Moreover, by recognizing that we cannot have fewer than 0 cuts, which is equivalent to $|\mathcal{I}|$ fragments, we trivially arrive at a better worst-case bound:

$$L_1 = |\mathcal{I}|.$$

This bound has a worst-case performance of 1/2, as we show in Lemma 3.

Lemma 3 $r(L_1) = 1/2$.

Proof An upper bound for any instance I of MIN-FIBP-EQ is $\mathrm{OPT}(I) \leq |\mathcal{I}| + |\mathcal{B}| - 1$, as shown by LeCun et al (2015). Since no items are larger than the bin capacity, $|\mathcal{B}| \leq |\mathcal{I}|$. Thus, $\mathrm{OPT}(I) \leq 2|\mathcal{I}| - 1$, so

$$\frac{L_1(I)}{\mathrm{OPT}(I)} \geq \frac{L_1(I)}{|\mathcal{I}| + |\mathcal{B}| - 1} \geq \frac{L_1(I)}{2|\mathcal{I}| - 1} \geq \frac{|\mathcal{I}|}{2|\mathcal{I}| - 1} \geq \frac{1}{2}.$$

To prove that this performance ratio is tight, consider a problem instance with capacity $c = k + 1$ and $|\mathcal{I}| = k + 1$ items of size k, requiring $|\mathcal{B}| = k$ bins with no slack left over. As k increases, we approach $1/2$. □

We can compute tighter bounds by reducing the problem through the exact algorithms $\mathcal{E}_1$ and $\mathcal{E}_2$. After removing all blocks of size 1 through algorithm $\mathcal{E}_1$, we can compute a better bound as

$$L_2 = \max\left(L_1, |\mathcal{I}| + |\mathcal{B}| - \left\lfloor \frac{|\mathcal{I}| + \hat{x}}{2} \right\rfloor\right),$$

where $\hat{x}$ is the maximum number of possible blocks consisting of a single item and a nonzero amount of slack. We obtain $\hat{x}$ by solving a special-case knapsack problem where all items have the same value, which amounts to picking blocks in non-increasing order of slack use. This has complexity $\mathcal{O}(n)$ for sorted items.

Lemma 4 *If there are no blocks of size* 1*, the number of blocks* b^* *in an optimal solution with* x *blocks consisting of one item is* $x \leq \hat{x} \leq b^* \leq \lfloor(|\mathcal{I}| + x)/2\rfloor \leq \lfloor(|\mathcal{I}| + \hat{x})/2\rfloor$.

Proof Let $b^* = x + q$ be the number of blocks in an optimal solution, where x is the number of blocks consisting of one item and a nonzero amount of slack, while q is the number of blocks consisting of more than one item. Every block requires at least one bin, so $x + q \leq |\mathcal{B}| \leq |\mathcal{I}|$. Every block of type x requires one item, while every block of type q requires at least two items, i.e. $x + 2q \leq |\mathcal{I}|$. So, $x + 2q \leq |\mathcal{I}| \Rightarrow 2x + 2q \leq |\mathcal{I}| + x \Rightarrow 2b^* \leq |\mathcal{I}| + x \Rightarrow b^* \leq \lfloor(|\mathcal{I}| + x)/2\rfloor$. □

Lemma 5 L_2 *is a lower bound for* MIN-FIBP-EQ.

Proof We need to show that $L_2(I) \leq \mathrm{OPT}(I)$. To do this, we consider two cases. In the first case, $L_2 = L_1$, so L_2 is a lower bound, since L_1 is a lower bound. In the other case, $L_2 = |\mathcal{I}| + |\mathcal{B}| - \lfloor(|\mathcal{I}| + \hat{x})/2\rfloor$. Given Corollary 1 of LeCun et al (2015), $\mathrm{OPT}(I) = |\mathcal{I}| + |\mathcal{B}| - b^*$, where b^* is the optimal number of blocks. Since $b^* \leq \lfloor(|\mathcal{I}| + \hat{x})/2\rfloor$, $L_2 \leq OPT$. □

It is clear that $L_2 \geq L_1$. This bound has a worst-case performance of $2/3$.

Lemma 6 $r(L_2) = 2/3$.

Proof We consider two cases: if $L_2 = L_1 = |\mathcal{I}|$, it follows that $|\mathcal{B}| \leq (|\mathcal{I}| + \hat{x})/2$. Given Corollary 1 of LeCun et al (2015), $\mathrm{OPT}(I) = |\mathcal{I}| + |\mathcal{B}| - b^*$. Since $b^* \geq \hat{x}$, $\mathrm{OPT}(I) \leq |\mathcal{I}| + |\mathcal{I}|/2$, and

$$\frac{L_2(I)}{\mathrm{OPT}(I)} \geq \frac{|\mathcal{I}|}{\frac{3|\mathcal{I}|}{2}} = \frac{2}{3}.$$

In the other case, $L_2 \geq |\mathcal{I}| + |\mathcal{B}| - (|\mathcal{I}| + \hat{x})/2$ and $|\mathcal{B}| \geq (|\mathcal{I}| + \hat{x})/2$. Given Corollary 1 of LeCun et al (2015), $\mathrm{OPT}(I) = |\mathcal{I}| + |\mathcal{B}| - b^*$. Since $b^* \geq \hat{x}$,

$$\frac{L_2(I)}{\mathrm{OPT}(I)} \geq \frac{|\mathcal{I}| + 2|\mathcal{B}| - \hat{x}}{2(|\mathcal{I}| + |\mathcal{B}| - b^*)} \geq \frac{1}{2}\left(2 - \frac{|\mathcal{I}|}{|\mathcal{I}| + \frac{|\mathcal{I}|}{2}}\right) = \frac{2}{3}.$$

To prove that this performance ratio is tight, consider a problem instance with capacity $c = 2k + 1$ and $|\mathcal{I}| = 2k + 1$ items of size k, completely filling $|\mathcal{B}| = k$ bins with no slack. As k increases, we approach 2/3. □

After also removing all blocks of size 2 through algorithm $\mathcal{E}_2$, we obtain an even better bound as

$$L_3 = \max\left(L_2, |\mathcal{I}| + |\mathcal{B}| - \left\lfloor \frac{|\mathcal{I}| + 2\hat{x} + \hat{y} - v}{3} \right\rfloor\right),$$

where $\hat{x}$ is the maximum number of possible blocks consisting of a single item and a nonzero amount of slack, $\hat{y}$ is the maximum number of possible blocks consisting of two items and a nonzero amount of slack, while $v = 0$ if the blocks of type $\hat{x}$ combined with the blocks of type $\hat{y}$ are feasible, otherwise $v = 1$. We obtain $\hat{x}$ as in bound L_2, while we obtain $\hat{y}$ in $\mathcal{O}(n \log n)$ by solving a special-case bin packing problem where no bin may contain more than two items, using a variation of FFD, after which we pick blocks in non-increasing order of slack use.

Lemma 7 *With no blocks of size* 1 *or* 2*, the number of blocks* b^* *in an optimal solution with* x *blocks consisting of one item and* y *blocks consisting of two items is* $\max(\hat{x}, \hat{y}) \leq x + y \leq b^* \leq \lfloor(|\mathcal{I}| + 2x + y)/3\rfloor \leq \lfloor(|\mathcal{I}| + 2\hat{x} + \hat{y} - v)/3\rfloor$.

Proof Let $b^* = x + y + q$ be the number of blocks in an optimal solution, where x is the number of blocks consisting of one item and a nonzero amount of slack, y is the number of blocks consisting of two items and a nonzero amount of slack, while q is the number of blocks consisting of more than two items. Every block requires at least one bin, so $x + y + q \leq |\mathcal{B}| \leq |\mathcal{I}|$. Every block of type x requires one item, every block of type y requires two items, while every block of type q requires at least three items, so $x + 2y + 3q \leq |\mathcal{I}|$. Hence, $x + 2y + 3q \leq |\mathcal{I}| \Rightarrow 3b^* \leq |\mathcal{I}| + 2x + y \Rightarrow b^* \leq \lfloor(|\mathcal{I}| + 2x + y)/3\rfloor$. □

Lemma 8 L_3 *is a lower bound for* Min-FIBP-EQ.

Proof We need to show that $L_3(I) \leq \mathrm{OPT}(I)$. We consider two cases: In the first case, $L_3 = L_2$, so L_3 is a lower bound, since L_2 is a lower bound. In the other case, $L_3 = |\mathcal{I}| + |\mathcal{B}| - \lfloor(|\mathcal{I}| + 2\hat{x} + \hat{y} - v)/3\rfloor$. Given Corollary 1 of LeCun et al (2015), $\mathrm{OPT}(I) = |\mathcal{I}| + |\mathcal{B}| - b^*$. Since $b^* \leq \lfloor(|\mathcal{I}| + 2\hat{x} + \hat{y} - v)/3\rfloor$, $L_3 \leq OPT$. □

It is clear that $L_3 \geq L_2 \geq L_1$. This bound has worst-case performance 3/4.

Lemma 9 $r(L_3) = 3/4$.

Proof We consider two cases: if $L_3 = L_2 = L_1 = |\mathcal{I}|$, it follows that $|\mathcal{B}| \leq (|\mathcal{I}| + 2\hat{x} + \hat{y} - v)/3$. Then, since $b^* \geq \max(\hat{x}, \hat{y})$, an upper bound for any instance I of MIN-FIBP-EQ is $\mathrm{OPT}(I) \leq |\mathcal{I}| + |\mathcal{I}|/3$, and

$$\frac{L_3(I)}{\mathrm{OPT}(I)} \geq \frac{|\mathcal{I}|}{\frac{4|\mathcal{I}|}{3}} = \frac{3}{4}.$$

Otherwise, $L_3 \geq |\mathcal{I}| + |\mathcal{B}| - (|\mathcal{I}| + 2\hat{x} + \hat{y} - v)/3$ and $|\mathcal{B}| \geq (|\mathcal{I}| + 2\hat{x} + \hat{y} - v)/3$. Given Corollary 1 of LeCun et al (2015), we define $\mathrm{OPT}(I) = |\mathcal{I}| + |\mathcal{B}| - b^*$. Since $b^* \geq \max(\hat{x}, \hat{y})$,

$$\frac{L_3(I)}{\mathrm{OPT}(I)} \geq \frac{2|\mathcal{I}| + 3|\mathcal{B}| + v - 2\hat{x} - \hat{y}}{3(|\mathcal{I}| + |\mathcal{B}| - b^*)} \geq \frac{1}{3}\left(3 - \frac{|\mathcal{I}|}{|\mathcal{I}| + \frac{|\mathcal{I}|}{3}}\right) = \frac{3}{4}.$$

To prove that this performance ratio is tight, consider a problem instance with capacity $c = 3k + 1$ and $|\mathcal{I}| = 3k + 1$ items of size k, completely filling $|\mathcal{B}| = k$ bins with no slack. As k increases, we approach 3/4. □

We improve the expected performance of bounds L_1, L_2 and L_3 by combining them with $L_0^{(p)}$:

$$L_1^* = \max(L_0^{(p)}, L_1)$$
$$L_2^* = \max(L_0^{(p)}, L_2)$$
$$L_3^* = \max(L_0^{(p)}, L_3).$$

Since bound L_3^* dominates the other bounds we will now use only this bound.

6 Grouping Genetic Algorithm

In this section we present the grouping genetic algorithm for solving the MIN-FIBP-EQ-problem. Its objective is to minimize the number of fragments, which is realized by maximizing the number of (minimal) blocks which encode the solution. The algorithm is adapted from Quiroz Castellanos et al (2015) to better suit our problem. Since individuals may be mutated in an optimality preserving fashion, the grouping genetic algorithm is optimality preserving. Solutions are evaluated according to the number of blocks they contain. More blocks means less fragments, which is better. It works as follows:

1. Start by removing all single items and pairs of items that completely fill a bin since these are optimal blocks. This is done through $\mathcal{E}_1\mathcal{E}_2$, which runs in $\mathcal{O}(n \log n)$.
2. If no items remain, the solution is optimal, else proceed by generating an initial population $\mathcal{P}$ by applying $\mathcal{G}^+$ or $\mathcal{B}_3\mathcal{G}^+$ to $|\mathcal{P}|$ random permutations of the items, resulting in no worse than a 4/3-approximation or a 5/4-approximation, depending on the use of $\mathcal{B}_3$.
3. Repeat while all of the following are true:

   (a) The best individual in the population has more fragments than the lower bound L_3^*
   (b) The number of generations run is less than the generation limit max_gen
   (c) The number of consecutive generations without improvement is less than the limit DL
4. Select n_c individuals
5. Do gene level crossover with the chosen heuristic on the selected individuals
6. Apply controlled replacement to introduce progeny
7. Select n_m individuals to mutate and clone elites with controlled selection
8. Do adaptive mutation with the chosen heuristic on the best n_m individuals
9. Apply controlled replacement to introduce clones
10. Update the current best solution

6.1 Selection

The selection operator is important in genetic algorithms. We use the controlled selection scheme of Quiroz Castellanos et al (2015) for crossover and mutation.

For crossover, n_c parents are selected to generate n_c children. Parents are selected through two sets, $\mathcal{G}$ and $\mathcal{R}$ for good individuals and random individuals. This enables elitism to increase the likelihood of good individuals reproducing, while avoiding premature convergence. The individuals in $\mathcal{G}$ and $\mathcal{R}$ are pairwise combined through crossover to produce the children. The elite set $\mathcal{Q}$ contains the $|\mathcal{Q}|$ best individuals of $\mathcal{P}$. The set $\mathcal{G}$ contains $n_c/2$ individuals selected at random with uniform distribution from the best n_c individuals of $\mathcal{P}$, while set $\mathcal{R}$ contains $n_c/2$ individuals chosen at random from $\mathcal{P} \setminus \mathcal{Q}$ with uniform distribution. Since there may be overlap between sets $\mathcal{G}$ and $\mathcal{R}$, we guard against selecting the same individual in the same position, to prevent self-crossover: We iterate over the element pairs, if two elements are equal, we swap one element with its right neighbor and advance two steps. If the end elements are equal, we swap one of them with its preceding element.

For mutation, we select the $n_m > |\mathcal{Q}|$ best individuals of $\mathcal{P}$ to be mutated. To preserve the best solutions, individuals of the elite set $\mathcal{Q}$ whose age is less than the predefined life span of elite solutions are cloned to form set $\mathcal{C}$. The cloned solutions may later be reinserted into the population.

6.2 Replacement

The replacement strategy is an important aspect of the algorithm. We use the controlled replacement strategy of Quiroz Castellanos et al (2015) for reinsertion after crossover and mutation.

Having selected the sets $\mathcal{G}$ and $\mathcal{R}$ of good and random solutions for crossover, the good solutions are used as the first parents and the random solutions as the second parents. The first $n_c/2$ children replace the individuals of the set of random parents $\mathcal{R} \subset \mathcal{P}$. The remaining $n_c/2$ children are introduced

to $\mathcal{P} \setminus (\mathcal{Q} \cup \mathcal{R})$ in two stages: firstly, if there are individuals with duplicate fitness, replace them with new offspring. Secondly, any remaining offspring are inserted by replacing the worst individuals. Note that two individuals having the same fitness does not imply that they have duplicate chromosomes. According to Quiroz Castellanos et al (2015), the rationale behind replacing individuals with duplicate fitness is that it acts as a proxy for replacing individuals with equivalent genomes, without incurring the associated expense of actually comparing genomes. This strategy maintains population diversity while preserving known good solutions without becoming stagnant, balancing exploration and exploitation.

When it comes to mutation, some of the best individuals may have been cloned during the selection process, forming the previously mentioned set of clones $\mathcal{C}$. Clones in $\mathcal{C}$ are reintroduced to the population in the same fashion as the previously mentioned offspring: if there are individuals with duplicate fitness, replace them with cloned solutions; any remaining clones are used to replace the worst individuals.

6.3 Grouping Crossover

We make use of an improved grouping crossover operator which we have derived from Quiroz Castellanos et al (2015) with modifications to better suit our problem. The crossover operator explores the search space while preserving the best blocks of the chosen individuals. Our genome representation is more efficient than that of Quiroz Castellanos et al (2015), since we use a cutting-stock type representation with no overhead. This reduces the search space by reducing symmetry, since the placement of specific items does not matter; the patterns of item sizes are what matters.

Given two parent solutions p_1 and p_2, we generate two children by combining p_1 with p_2 and p_2 with p_1. The crossover operator considers the blocks of the solutions in non-decreasing order of score. The blocks of each parent are compared pairwise, which can easily be done in parallel. The lower scored block in each pair is inherited first, followed by the other block. If both blocks have the same score, the block from the first parent is inherited first.

For example, if the scores of the blocks comprising the two parents are $p_1 = (A = 3, B = 4, C = 5)$ and $p_2 = (D = 1, E = 5, F = 5)$, we form the pairs (A, D), (B, E) and (C, F). We then sort each pair in non-decreasing order on score, giving (D, A), (B, E) and (C, F), which are joined to form the offspring $f_1 = (D, A, B, E, C, F)$ containing all blocks from its parents. We then repeat, forming the other child $f_2 = (D, A, B, E, F, C)$.

If the parents contain an unequal number of blocks, we perform a right justification, meaning that we first inherit the lowest scored blocks of the longer parent, until the number of remaining blocks is equal to that in the other parent, before proceeding as above. This is because we want as many groups as possible, contrary to the algorithm of Quiroz Castellanos et al (2015), which tried to obtain as few groups as possible.

Blocks requiring more items than available of some size are not inherited, instead their items join the set of free items. After combination, new blocks are formed from a random permutation of the set of free items through the chosen packing heuristic, $\mathcal{G}^+$ or $\mathcal{B}_3\mathcal{G}^+$, and added to the child.

6.4 Mutation

The mutation operator serves to explore the search space by random search, avoiding stagnation in local minimia. Quiroz Castellanos et al (2015) note that modifying a fixed proportion of chromosomes is undesirable when the number of chromosomes is variable. For example, given mutation proportion $\varepsilon = 0.1$, only 2 chromosomes would be mutated in a solution of size $b = 20$ blocks. This may be too few chromosomes to have an impact. However, in a solution of size $b = 100$ blocks, 10 chromosomes would be mutated, which may be too aggressive. We therefore desire an adaptive mutation operator which accounts for the size of the solution. Quiroz Castellanos et al (2015) also note that individuals which have been cloned may be mutated more aggressively, since their genetic material will be preserved if no better solutions are found through mutation. With this in mind, we designed an adaptive mutation operator for our problem, where individuals of size b blocks with $u < b$ empty blocks are mutated by eliminating $n_b = \max(\lceil b\varepsilon \rceil, u + v)$ blocks, where $v = 1$ if the worst non-empty block consists of a single bin, else $v = 0$. This ensures that mutation actually has a chance of doing something, since mutating a single non-empty block consisting of a single bin cannot lead to an improvement. Note that there naturally cannot be any empty blocks if a problem instance has as few available bins as possible.

$$\begin{aligned} \varepsilon &= \text{Kumaraswamy}(9p_\varepsilon, 9(1-p_\varepsilon)) \\ p_\varepsilon &= \left(\frac{b'-b}{2b'}\right)^{\frac{1}{k}} \\ b' &= |\mathcal{I}| + |\mathcal{B}| - L_3^* \end{aligned}$$

Here, the elimination proportion ε is drawn from a Kumaraswamy distribution on the domain $(0, 1)$, parametrized with the variable p_ε. Jones (2009) studied the Kumaraswamy distribution. The upper bound on blocks b' is the dual of the lower bound on fragments. As the number of blocks b in a solution grows, p_ε decreases, which makes small values of ε more likely. The parameter k is a predetermined constant signifying the desired rate of change. We use two different values for this parameter, depending on whether the current individual has been cloned or not. Larger values of k increase the value of p_ε, which in turn increases likelihood of ε being large. The resulting distribution of ε is depicted in Fig. 4.

Having decided how many blocks to eliminate, we now need to determine which ones. We always eliminate the worst scored block and all empty blocks, since they have a high potential for improvement. To keep the mutation

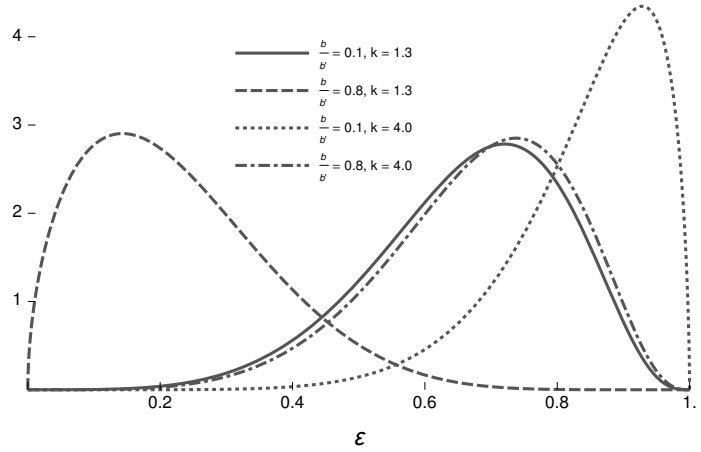

Fig. 4 Distribution of the elimination proportion ε for relative solution sizes b/b' and rates of change k

operator optimality preserving, we pick additional blocks to remove according to a uniformly random distribution.

The items of the eliminated blocks are added to the set of free items. Thereafter, new blocks are formed from a random permutation of the set of free items through the chosen packing heuristic, $\mathcal{G}^+$ or $\mathcal{B}_3\mathcal{G}^+$, and added back to the individual. Since any block is eligible for elimination and algorithm $\mathcal{G}^+$ is optimality preserving, the mutation operator is optimality preserving if $\mathcal{G}^+$ is used alone. However, $\mathcal{B}_3\mathcal{G}^+$ is not optimality preserving. We remedy this by eliminating a random number of blocks produced by $\mathcal{B}_3$ before continuing with $\mathcal{G}^+$. This is optimality preserving, since $\mathcal{G}^+$ is not bound by $\mathcal{B}_3$. We have chosen a binomial distribution $B(x, 1/8)$, where x is the number of blocks previously produced by $\mathcal{B}_3$ to draw the number of blocks to eliminate.

7 Experimental Evaluation

We have implemented a MIN-FIBP-EQ solver in Byholm (2017a) using the grouping genetic algorithm with the $\mathcal{B}_3\mathcal{G}^+$ approximation algorithm as its core heuristic. We chose the C++ programming language for the implementation, since it offers high performance and ease of use. The goal of the evaluation is to experimentally show:

1. The feasibility of the implementation to tackle large problem instances.
2. The actual performance of L_3^* in a large and varied problem set.
3. The actual performance of the genetic algorithm in a large problem set.

We used a regular desktop computer with an Intel® Core™ i7-4770 CPU running at 3.40 GHz and 16 GiB of RAM to run the solver. The processor has 4 cores but our implementation uses only one thread.

7.1 The Problem Sets

While it would be possible to reuse a set for classic bin packing, it is not guaranteed that a MIN-FIBP-EQ instance derived from a BP-instance which is considered hard would also be hard, and vice versa. Additionally, we would not have knowledge of the optimal solution to such a MIN-FIBP-EQ instance, unless the number of available bins was equal to the optimal solution of the BP instance. Therefore, we use two problem sets specifically designed for MIN-FIBP-EQ: One is the set $\mathcal{P}_1$ presented by Casazza and Ceselli (2014) and the other is a new problem set $\mathcal{P}_2$ designed by us. We ran each instance 10 times.

The problem set $\mathcal{P}_1$ of Casazza and Ceselli (2014) considers instances of MIN-FIBP-EQ with bin capacity $c = 1000$ having 10, 15 and 20 items, with three types of size distribution: large items with sizes between 500 and 900, small items with sizes between 100 and 500 and free items with sizes between100 and 900. There are two further classes, complete instances having 0 slack units and incomplete instances having 100 slack units. Each problem class has 10 instances, for a total of 180 instances. The instances are named like bc_is/bpp_20_0, which is the first instance with 20 items, small sizes and no slack, while bi_il/bpp_10_9 is the tenth instance with 10 items, large sizes and 100 slack.

The problem set was designed to exercise exact algorithms. As such, it is not an interesting problem set for our approach, since it has very few items. We also consider it highly artificial and find problems of the cutting-stock type more interesting for MIN-FIBP-EQ. We should also note that the result set referenced by Casazza and Ceselli (2014) reports four incorrect optima for their problem set, as described in Table 2.

Thus, to properly evaluate the performance and execution times of the solver, we constructed another set $\mathcal{P}_2$ with 450 problem instances according to the parameters in Table 1, published as Byholm (2017b). The problem instances have 256, 512 and 1024 items with bin capacities of 8, 16 and 32. We chose to divide the item size distribution into all combinations of thirds because of how it affects algorithm $\mathcal{B}_3$. For example, there are no blocks of size 3 if all items have size greater than $2c/3$. There are 10 instances for each parameter set. All problem instances used the fewest bins possible, which makes for harder problem instances. We refer to problem classes from set $\mathcal{P}_2$ as a tuple. For example, $(32, 1, 21, 256)$ is the problem class with bin capacity 32, item sizes between 1 and 21 having 256 items.

We also implemented an exact solver to obtain the optima of the problem instances in the second set. It works by enumerating all blocks of size $k = 3, \ldots, K$ possible with the current set of item sizes and then attempting to pick as many blocks as possible while using all items. The solver uses Mathematica, requiring many days of computing time to process our problem set.

We designed a problem set $\mathcal{P}_3$ to evaluate the execution time of the implementation when dealing with a large number of unique item sizes. This set comprises six problem instances involving n items whose sizes are uniformly distributed over the entire capacity range from 1 to c for each $n = c = 1000, 2000, 4000, 8000, 16000, 32000$.

Table 1 Parameters for problem set $\mathcal{P}_2$. Capacity is the bin capacity. Low and High form the range of item sizes, while Items indicates the number of items

Capacity = 8			Capacity = 16			Capacity = 32		
Low	High	Items	Low	High	Items	Low	High	Items
1	2	256	1	5	256	1	10	256
1	2	512	1	5	512	1	10	512
1	2	1024	1	5	1024	1	10	1024
1	5	256	1	10	256	1	21	256
1	5	512	1	10	512	1	21	512
1	5	1024	1	10	1024	1	21	1024
1	8	256	1	16	256	1	32	256
1	8	512	1	16	512	1	32	512
1	8	1024	1	16	1024	1	32	1024
3	8	256	6	16	256	11	32	256
3	8	512	6	16	512	11	32	512
3	8	1024	6	16	1024	11	32	1024
6	8	256	11	16	256	22	32	256
6	8	512	11	16	512	22	32	512
6	8	1024	11	16	1024	22	32	1024

Table 2 Reported and actual optima in problem set $\mathcal{P}_1$ of Casazza and Ceselli (2014)

Problem	Reported	Actual
bc_il/bpp_15_0	24	23
bc_il/bpp_15_8	24	23
bc_il/bpp_20_8	31	30
bi_if/bpp_20_2	20	21

7.2 Parameters for the Genetic Algorithm

We use the parameters of Quiroz Castellanos et al (2015), which were obtained through extensive computational experiments for classic BP. Since our genetic algorithm is derived therefrom, we assume that they translate reasonably well. We added one additional parameter: the maximum number of generations without improvement DL, which is useful for quickly obtaining good solutions.

Population size	$\lvert\mathcal{P}\rvert = 100$
Maximal number of generations	$max\_gen = 500$
Maximum number of generations without improvement	$DL = 100$
Number of individuals to be recombined	$n_c = 20$
Number of individuals to mutate	$n_m = 83$
Number of individuals in the elite set	$\lvert\mathcal{Q}\rvert = 10$
Maximal individual age to be cloned	$life\_span = 10$
Rate of change in mutation for non-cloned individuals	$k = 1.3$
Rate of change in mutation for cloned individuals	$k = 4$

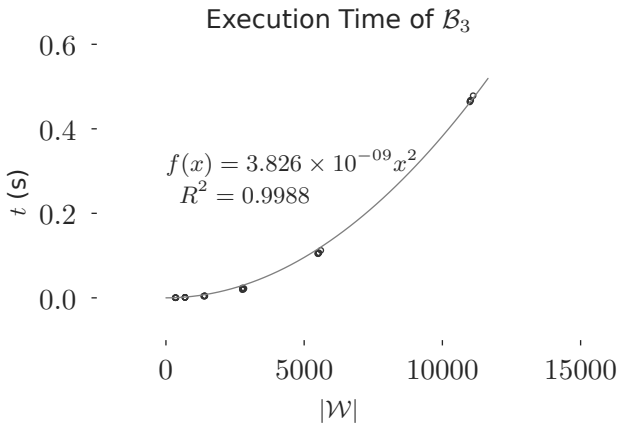

Fig. 5 The execution time of $\mathcal{B}_3$ for $\mathcal{P}_3$ seems to grow quadratically with a coefficient of determination $R^2 = 0.9988$. $\mathcal{B}_3$ processes more than 11000 unique item sizes in less than 0.5 s

7.3 Execution Time

In Section 4.4, we show that algorithm $\mathcal{B}_3$ has a time complexity $\mathcal{O}(n + |\mathcal{W}|^2)$. The complexity of this algorithm dominates the grouping genetic algorithm. We can now corroborate experimentally that the C++ implementation also exhibits quadratic growth in execution time. For this task, we measured the execution time of $\mathcal{B}_3$ for five randomly generated problem instances in each of the six problem classes of $\mathcal{P}_3$ described in Section 7.1.

Fig. 5 shows the execution time of $\mathcal{B}_3$ as the set of item sizes $\mathcal{W}$ grows. The full result set is available in Byholm (2017c). It is worth mentioning that the implementation of $\mathcal{B}_3$ can process a rather large problem instance with more than 11000 unique item sizes in less than 0.5 s. The trend appears quadratic with a coefficient of determination $R^2 = 0.9988$. This is further supported by the slope $k = 2.1528$ of the linear regression line on the logarithms of the variables, confirming that the execution time grows quadratically.

The execution time for the grouping genetic algorithm depends on the number of generations. As shown in Fig. 6, it never runs for more than 17 ms for any instance of $\mathcal{P}_1$. Most of the 1800 runs completed in less than 6 ms. As shown in Fig. 7, the algorithm never runs for more than 220 ms for any instance of $\mathcal{P}_2$. Most of the 4500 runs completed in less than 13 ms. The full result set is available in Byholm (2017c).

7.4 Lower Bound Performance

We define the lower bound performance for a problem instance as the ratio between the lower bound and the optimum of the corresponding reduced problem instance. This can be related to Definition 1 (Worst-Case Performance).

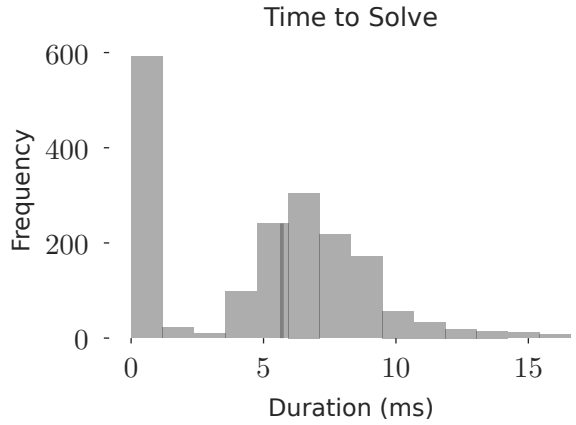

Fig. 6 Duration until termination for problem set $\mathcal{P}_1$

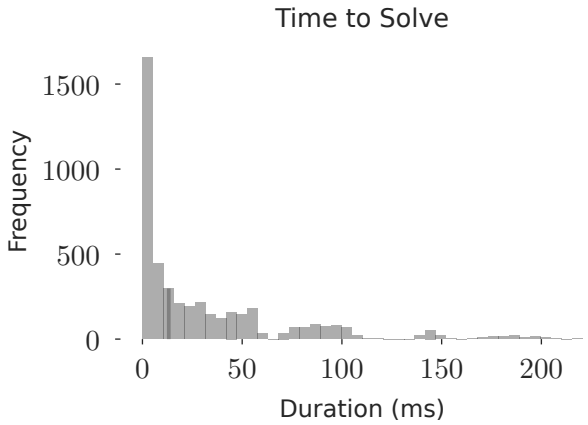

Fig. 7 Duration until termination for problem set $\mathcal{P}_2$

The results for L_3^* applied to problem set $\mathcal{P}_1$ show that the observed lower bound performance is no worse than 5/6 in any of the 180 problem instances, as depicted in Fig. 8. We would like to note that the lower bound coincided with the optimum in 65 instances (36 %). Lower bound L_0 weakly dominated L_3 in 79 instances (44 %). The actual performance is better than the worst case. The median performance ratio is 37/40.

The observed performance of L_3^* applied to problem set $\mathcal{P}_2$ is no worse than 9/10 in any of the 450 problem instances, as depicted in Fig. 9. We would like to note that the lower bound coincided with the optimum in 211 instances (47 %). Lower bound $L_0^{(20)}$ weakly dominated L_3 in 213 instances (47 %). The actual performance is again better than the worst case. The median performance ratio is 39/40.

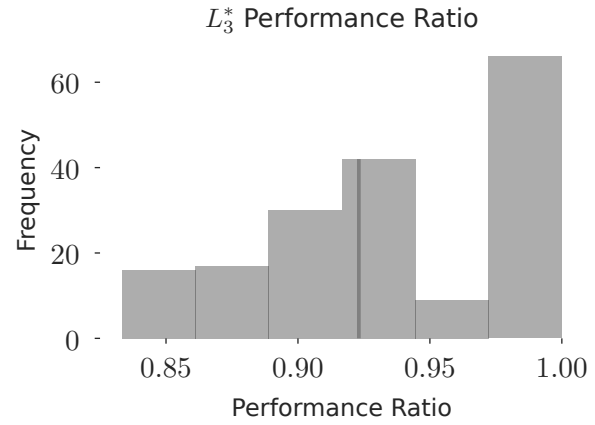

Fig. 8 Lower bound performance ratio for problem set $\mathcal{P}_1$

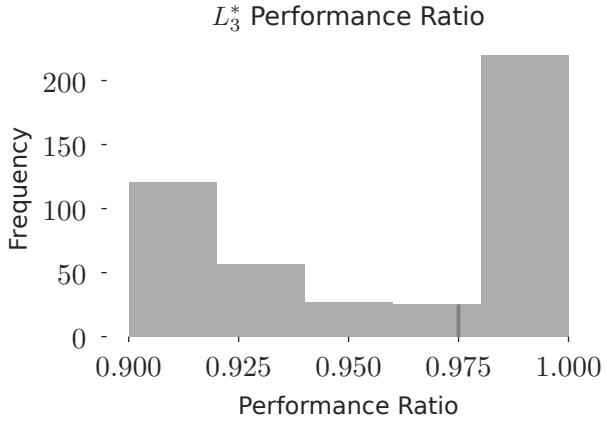

Fig. 9 Lower bound performance ratio for problem set $\mathcal{P}_2$

This indicates that the lower bound L_3^* works well in practice and that lower bound $L_0^{(p)}$ can be highly useful in applications where it is desirable to use as few bins as possible. The full result set is available in Byholm (2017c).

7.5 Solution Performance

We define the solution performance for a reduced problem instance as the ratio between the value of a solution computed by a solver for that instance and the optimal solution. As shown in Section 6, there were three termination conditions for the genetic algorithm: 1. Reaching the lower bound. 2. The maximum number of generations were run. 3. Too many generations passed without improvement.

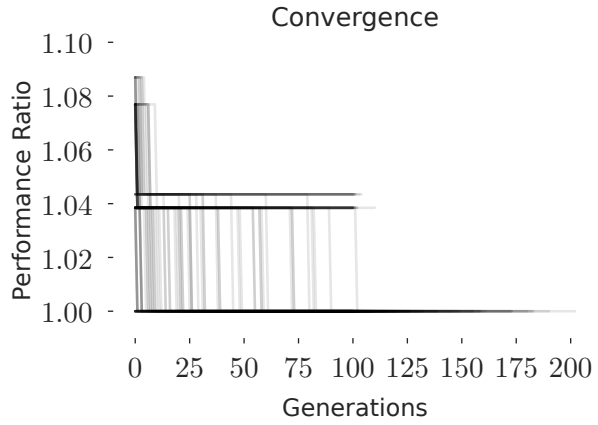

Fig. 10 Convergence rate of problem class bc_if_20 from problem set $\mathcal{P}_1$

The maximum number of generations was never reached in any run of the studied problem instances. No run lasted for more than 202 generations in problem set$\mathcal{P}_1$, while no run lasted for more than 219 generations in $\mathcal{P}_2$. Out of 1800 runs for $\mathcal{P}_1$, only 138 (8 %) failed to reach the optimum before terminating, while only 27 out of 4500 runs (1 %) failed to reach the optimum before terminating in $\mathcal{P}_2$. The full result set is available in Byholm (2017c). For both problem sets, every problem instance was solved to optimality in some run, but this cannot be guaranteed in general.

We observe that the grouping genetic algorithm outperforms the worst-case performance guarantee 5/4 of the basic approximation algorithm $\mathcal{E}_1\mathcal{E}_2\mathcal{B}_3\mathcal{G}^+$ for every run of every instance of either problem set. We can also see that the grouping genetic solver converges quickly, as shown in Fig. 10, 11, 12 and 13 for a subset of our tests with different configuration parameters.

8 Conclusions

We presented a fast grouping genetic algorithm for the bin packing problem with fragmentable items. It delivers a worst-case guarantee of a 5/4-approximation with complexity $\mathcal{O}(n \log n + |\mathcal{W}|^2) = \mathcal{O}(n^2)$. The metaheuristic works well in practice. With problem set $\mathcal{P}_1$, it reached the optimum in 92 % of runs (1662 out of 1800 runs). With problem set $\mathcal{P}_2$, it reached the optimum in 99 % of runs (4473 out of 4500 runs). To this end, we made the following contributions: 1. We developed the greedy algorithm $\mathcal{G}^+$ for constructing a feasible solution. It has worst-case complexity $\mathcal{O}(n)$ and works for general instances of Min-FIBP. Its predecessor $\mathcal{G}$ has complexity $\mathcal{O}(n)$ only for a special case of Min-FIBP. 2. We made a minor modification to the exact algorithm $\mathcal{E}_2$, adapting it to the general case of Min-FIBP-EQ. This avoids the need for a pseudo-polynomial transformation. 3. Algorithm $\mathcal{B}_3$ is a 1/3-approximation

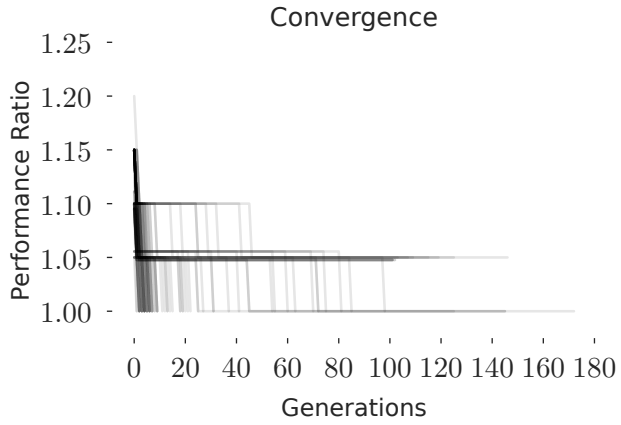

Fig. 11 Convergence rate of problem class bi_if_20 from problem set $\mathcal{P}_1$

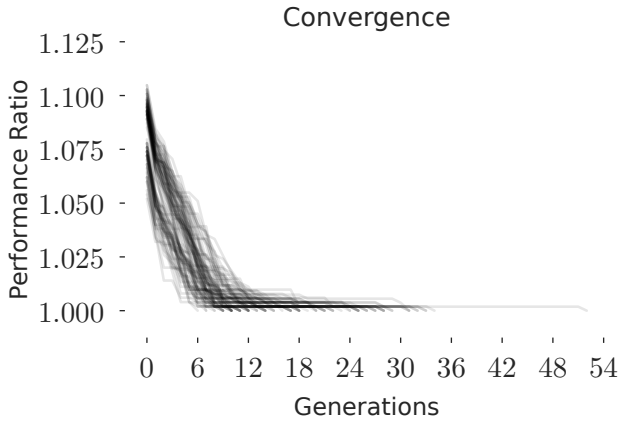

Fig. 12 Convergence rate of problem class $(32, 1, 21, 1024)$ from problem set $\mathcal{P}_2$

with complexity $\mathcal{O}(n + |\mathcal{W}|^2) = \mathcal{O}(n^2)$ for finding blocks of size 3 in the general case of MIN-FIBP-EQ. Its predecessor $\mathcal{A}_3$ is a 1/3-approximation with complexity $\mathcal{O}(n^4)$ for the special case of MIN-FIBP-EQ where the sum of item sizes is equal to the aggregated bin capacity. 4. We designed a new family of fast lower bounds for MIN-FIBP-EQ and proved their worst-case performance ratios. The best lower bound offers a guaranteed worst-case performance of 3/4. With the first problem set, it performed no worse than 5/6. With the second problem set, it performed no worse than 9/10. Its worst-case complexity is $\mathcal{O}(n \log n)$. 5. We made significant changes to a grouping genetic algorithm for classic bin packing, as well as its constituent operators, and adapted it to the new problem MIN-FIBP-EQ. We also proposed a more efficient genome representation for all grouping genetic algorithms designed

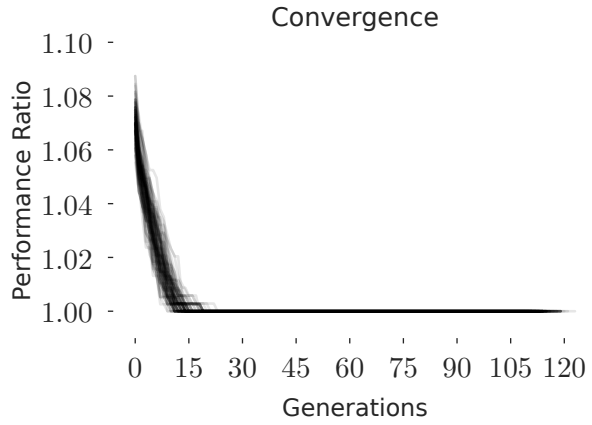

Fig. 13 Convergence rate of problem class $(32, 22, 32, 256)$ from problem set $\mathcal{P}_2$

for bin packing. 6. We created a reference implementation of a state-of-the-art solver incorporating the proposed algorithms and made it available to the public in Byholm (2017a). 7. We designed a comprehensive problem instance set $\mathcal{P}_2$ to evaluate the solver and made it publicly available in Byholm (2017b). We published the full result set in Byholm (2017c) to allow for future comparisons.

8.1 Future work

In this article, we decided to focus on algorithms with no worse than quadratic time complexity. LeCun et al (2015) also presented a 2/5-approximation algorithm $\mathcal{A}_4$, with complexity $\mathcal{O}(n^{13})$ for finding blocks of size 3 and 4. While its computational complexity can be significantly reduced using similar methods as in $\mathcal{B}_3$, the resulting algorithm would still have too high complexity for use as a frequently evaluated heuristic.

The presented approximation algorithms are also useful for other metaheuristics than genetic algorithms. We chose a grouping genetic algorithm since the approach of Quiroz Castellanos et al (2015) worked well for the related problem of classic BP. Combining the approximation algorithms with, e.g. a local neighborhood search strategy might be another option. Other strategies for quickly solving this kind of packing problem may also be worth exploring.

Future work also involves designing an efficient rearrangement heuristic for bin packing with fragmentable items, which should further increase the convergence rate of the grouping genetic algorithm. It would also be interesting to look for better lower bounds than those presented. A parallelized implementation might provide further performance gains. To find better approximation algorithms with the same complexity would probably require a new approach, which is not based on set packing. Multi-dimensional variants of bin packing

with fragmentable items is another possible extension. We will continue our work by investigating new applications and algorithms.

Acknowledgements Benjamin Byholm received scholarships from the Nokia Foundation and the Finnish Foundation for Technology Promotion. This was part of the N4S project.

References

Byholm B (2017a) fragbinpacking-optimizer v1.3. DOI 10.5281/zenodo.1068975
Byholm B (2017b) fragbinpacking-problems v1.1. DOI 10.5281/zenodo.253942
Byholm B (2017c) fragbinpacking-results v1.3. DOI 10.5281/zenodo.1068972
Casazza M, Ceselli A (2014) Mathematical programming algorithms for bin packing problems with item fragmentation. Comput Oper Res 46(C):1–11, DOI 10.1016/j.cor.2013.12.008
Casazza M, Ceselli A (2016) Exactly solving packing problems with fragmentation. Comput Oper Res 75(C):202–213, DOI 10.1016/j.cor.2016.06.007
Falkenauer E (1992) The grouping genetic algorithms – widening the scope of the GAs. Belg J Oper Res Stat Comput Sci 33(1):2
Falkenauer E (1996) A hybrid grouping genetic algorithm for bin packing. J Heuristics 2(1):5–30, DOI 10.1007/BF00226291
Fekete SP, Schepers J (2001) New classes of fast lower bounds for bin packing problems. Math Program 91(1):11–31, DOI 10.1007/s101070100243
Gajentaan A, Overmars MH (2012) On a class of $O(n^2)$ problems in computational geometry. Comput Geom 45(4):140–152, DOI 10.1016/j.comgeo.2011.11.006
Garey MR, Johnson DS (1990) Computers and Intractability. W. H. Freeman & Co., New York, NY, USA
Jones MC (2009) Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. Stat Methodol 6(1):70–81, DOI 10.1016/j.stamet.2008.04.001
LeCun B, Mautor T, Quessette F, Weisser MA (2015) Bin packing with fragmentable items: Presentation and approximations. Theor Comput Sci 602:50–59, DOI 10.1016/j.tcs.2015.08.005
Mandal CA, Chakrabarti PP, Ghose S (1998) Complexity of fragmentable object bin packing and an application. Comput Math Appl 35(11):91–97, DOI 10.1016/S0898-1221(98)00087-X
Quiroz Castellanos M, Cruz Reyes L, Torres Jiménez J, Gómez Santillán C, Fraire Huacuja HJ, Alvim ACdF (2015) A grouping genetic algorithm with controlled gene transmission for the bin packing problem. Comput Oper Res 55:52–64, DOI 10.1016/j.cor.2014.10.010