A Study of Bi-space Search for Solving the One-Dimensional Bin Packing Problem

Derrick Beckedahl(✉) and Nelishia Pillay

University of Pretoria, Pretoria, South Africa
d.beckedahl@gmail.com, npillay@cs.up.ac.za

Abstract. Traditionally search techniques explore a single space to solve the problem at hand. This paper investigates performing search across more than one space which we refer to as bi-space search. As a proof of concept we illustrate this using the solution and heuristic spaces. In previous work two approaches for combining search across the heuristic and solution spaces have been studied. The first approach, the *sequential* approach, firstly explores the heuristic space to obtain complete solutions and then applies local search to explore the solution space created by these solutions. The second approach, the *interleaving* approach, alternates search in the heuristic and solution space on partial solutions until complete solutions are produced. This paper provides an alternative to these two approaches, namely, the *concurrent* approach, which searches the heuristic and solution spaces simultaneously. This is achieved by implementing a genetic algorithm selection hyper-heuristic that evolves a combination of low-level construction heuristics and local search move operators that explore the space of solutions (both partial and complete). The performance of the three approaches are compared, to one another as well as with a standard selection construction hyper-heuristic, using the one dimensional bin packing problem. The study revealed that the concurrent approach is more effective than the other two approaches, with the interleaving approach outperforming the sequential approach. All 3 approaches outperformed the standard hyper-heuristic. Given the potential of searching more than one space and the effectiveness of the concurrent approach, future work will examine additional spaces such as the design space and the program space, as well as extending the bi-space search to a multi-space search.

Keywords: Bi-space search · Heuristic space · Solution space · Genetic algorithms

1 Introduction

The vast majority of search algorithms employed when solving combinatorial optimization problems (COPs) restrict their search to only a single search space. For example a genetic algorithm generally searches within the solution space, while genetic programming searches within the program space. The potential benefit of searching across two spaces can be seen in [11] where the performance of the graph-based hyper-heuristic (GHH) improves when introducing a solution space search. In this work we present an alternative approach to those presented in [11], namely a simultaneous approach, in which the search across the heuristic and solution spaces is optimised using a hybridised selection hyper-heuristic. A comprehensive coverage of the field of hyper-heuristics can be found in [10].

Hence, we investigate three different methods of combining search in the heuristic and solution spaces, two of which are taken from [11]. The first method performs a greedy local search on complete solutions created by a sequence of construction heuristics [11]. That is to say that a series of construction heuristics is consecutively applied until a complete solution is obtained, after which the greedy local search is applied. This method will be referred to as the sequential search approach (SSA) in this paper.

The second method is to interleave the local search with the application of the construction heuristics [11]. In other words, after the application of a single construction heuristic a partial solution is obtained, on which local search is performed until there is no improvement. The next construction heuristic in the sequence is applied to the resulting partial solution, with the process being repeated until a complete solution is obtained. For the remainder of this paper, we will refer to this method as the interleaving search approach (ISA).

In [11], it was found that the ISA method performed better than the SSA method. We hypothesize that this is due to the ISA method working on partial solutions as opposed to on complete solutions. This study uses a hybridisation of selection constructive and selection perturbative hyper-heuristics to optimise the search between the heuristic and solution spaces. We hypothesize that this approach will be an improvement on the ISA approach because, rather than forcing search in the solution space at periodic intervals, the proposed method will optimise when each space is searched. This is achieved by employing a selection hyper-heuristic which explores the space of heuristic combinations comprising low-level construction heuristics as well as local search operators. When a local search operator is encountered, the search switches to the solution space by applying said operator within the solution space. We will refer to this method as the concurrent search approach (CSA).

The work presented here differs from previous work in that a multi-point search technique has been used when searching the two spaces (as opposed to the single-point search techniques employed in [11]), as well as using a different problem domain, namely the one-dimensional bin packing problem (1BPP) from the cutting and packing class of problems. This domain was chosen as it is a well-researched problem domain, with numerous variations, that are directly applicable to industry [8,13]. The results show that the CSA method performs the best across the benchmark problem instances, with the second best performing being the ISA method.

The main contribution of the research presented in this paper is an alternative approach for search across the heuristic and solution spaces, which performs better than previous approaches applied for this purpose. This study also further emphasizes the potential of bi-space search as opposed to restricting the search to a single search space. The remainder of this paper is organized as follows. Section 2 details the different approaches, followed by the experimental setup in Sect. 3. Experimental results and analyses are provided in Sect. 4. Finally conclusions and future work are given in Sect. 5.

2 Bi-space Search Algorithms

The hyper-heuristic was implemented using a Genetic Algorithm (GA) for the high level heuristic, with the GA employing the generational control model and tournament selection. Algorithm 1 details the pseudocode for the GA, which was implemented using the *EvoHyp* [9] toolkit.

Algorithm 1. Genetic Algorithm Pseudocode

```
1: Create initial population
2: for $i \leftarrow 1, N$ do                    ▷ $N$ = number of generations
3:     Evaluate the population
4:     Select parents of the next generation
5:     Apply the genetic operators (mutation and crossover)
6: end for
```

The following subsections describe each of the three approaches that were implemented, namely the SSA (Sect. 2.1), ISA (Sect. 2.2) and CSA (Sect. 2.3) approaches.

2.1 Sequential Search Approach (SSA)

The SSA approach is implemented by using a selection hyper-heuristic to search through combinations of low-level construction heuristics. The construction heuristics are used to build a complete solution, which is used as a starting point for a search in the solution space (local search). Each combination of low-level heuristics is evaluated by firstly applying it to create a solution. Local search is then applied to that solution, and the objective value of the resulting solution is used as the fitness of the combination. This fitness value is used to guide the hyper-heuristic search toward finding an optimal solution to the problem at hand. Algorithm 2 provides pseudocode for evaluating an individual in the SSA approach.

2.2 Interleaving Search Approach (ISA)

The ISA approach is implemented by using a selection hyper-heuristic to search through a space consisting of sequences of low-level construction heuristics (the

Algorithm 2. SSA Evaluate

```
1: for $i \leftarrow 1, n$ do                    ▷ $n$ = length of heuristic sequence
2:     Apply $i^{\text{th}}$ construction heuristic in the sequence
3: end for
4: repeat
5:     Apply perturbative heuristic
6: until no improvement in solution quality
```

heuristic space). The fitness of a particular heuristic sequence is determined as follows. After the application of a single heuristic within the sequence, the resulting partial solution is used as a starting point for local search (search in the solution space). As previously mentioned, an exhaustive local search, i.e. until there is no improvement, is performed on this partial solution [11]. After the local search has been conducted on the partial solution, the next construction heuristic in the sequence is applied. This procedure is repeated until a complete solution is obtained, with a final exhaustive local search being performed on the complete solution. The fitness of the solution resulting from this procedure is used as the fitness value for the heuristic sequence being evaluated. The pseudocode for this evaluation process is detailed in Algorithm 3.

Algorithm 3. ISA Evaluate

```
1: for $i \leftarrow 1, n$ do                    ▷ $n$ = length of heuristic sequence
2:     Apply $i^{\text{th}}$ construction heuristic in the sequence
3:     repeat
4:         Apply perturbative heuristic
5:     until no improvement in (partial) solution quality
6: end for
```

2.3 Concurrent Search Approach (CSA)

The CSA approach is implemented as a hybrid selection hyper-heuristic which searches through both constructive and perturbative low-level heuristics, i.e. a hybridisation of a selection constructive and a selection perturbative hyper-heuristic. The fitness of a heuristic sequence is determined by consecutively applying each heuristic within the sequence. When the perturbative heuristic is encountered within the sequence, then a single step of the local search procedure is performed, on the solution obtained from the application of all previous heuristics in the current sequence. This can be either a partial or complete solution. The fitness of the resulting solution is used as the fitness for the heuristic sequence. The reason for only performing a single step of the local search procedure is that the higher level search is meant to determine when each space is searched, as well as for how long the space is searched. Hence if multiple steps of the local search procedure are required, then the perturbative heuristic will appear multiple times within the sequence. Algorithm 4 details how an individual is evaluated in the CSA approach, where the heuristic in Line 2 can be either constructive or perturbative.

Algorithm 4. CSA Evaluate

```
1: for $i \leftarrow 1, n$ do                    ▷ $n$ = length of heuristic sequence
2:     Apply $i^{\text{th}}$ heuristic in the sequence
3: end for
```

3 Experimental Setup

As previously stated, each of the bi-space search approaches were implemented for the one-dimensional bin packing problem (1BPP) domain, using the Scholl [13] benchmark data sets. These data sets are grouped into three broad categories, namely easy, medium and hard[1]. The complete benchmark consists of a total of 1210 problem instances, separated into groups of 720, 480 and 10 for easy, medium and hard respectively.

The heuristic space consists of sequences of construction heuristics for the 1BPP. In our implementation, each sequence of heuristics is represented by a string of characters, where each character corresponds to a low-level heuristic (or a local search operator in the case of the CSA approach). Each sequence is used to construct a solution in the solution space, the fitness of which is used as a measure of the quality of the heuristic sequence. The fitness of the solutions was calculated using the function proposed by Falkenauer [4], which is to minimize:

$$f_{BPP} = 1 - \frac{\sum_{i=1}^{N}(F_i/C)^2}{N} \tag{1}$$

where N is the number of bins, F_i the sum of the item sizes in the i^{th} bin and C the capacity of each bin. The following low-level construction heuristics were used [10]:

- **First-Fit Decreasing**: The items to be packed are sorted in descending order, the first item in the list is placed in the first bin in which it fits. If the item does not fit into an existing bin, a new bin is created and the item placed inside.
- **Best-Fit Decreasing**: The items to be packed are sorted in descending order, the first item in the list is placed in the bin with the least residual capacity after the item has been packed. If the item does not fit into an existing bin, a new bin is created and the item placed inside.

[1] A full explanation of the datasets and their respective classes can be found at https://www2.wiwi.uni-jena.de/Entscheidung/binpp/index.htm.

- **Next-Fit Decreasing**: The items to be packed are sorted in descending order, the first item in the list is placed in the current bin if possible, else the item is placed in a new bin.
- **Worst-Fit Decreasing**: The items to be packed are sorted in descending order, the first item in the list is placed in the bin with the most residual capacity after the item has been packed. If the item does not fit into an existing bin, a new bin is created and the item placed inside.

The parameter values were tuned such that the performance of the GA would be as general as possible, in order to ensure that no approach had an advantage over any other. It was found that there were no changes when using larger values for the population size. A lower number of generations was found not to converge, whilst convergence occurred before reaching the generation limit when using higher values. Similar observations were made for changes in the application rates. It was found that convergence took longer to occur when a limit was imposed for the maximum depth of the offspring. Likewise for higher limits on the mutation depth. All approaches used the parameter values reported in Table 1.

Table 1. Table showing the parameter values used for the GA for all three approaches.

Parameter	Value
Population size	500
Tournament size	5
No. of generations	75
Mutation rate	0.15
Crossover rate	0.85
Max. depth of init. pop	10
Max. depth of offspring	No limit
Max. mutation depth	5

Search within the solution space was conducted through the use of a local search operator, which was adapted from [7]. This operator was chosen due to its popularity within the literature [1,7]. Algorithm 5 details the implementation of this operator. In cases where there are multiple swaps which would lead to a reduced residual capacity (free space) for a bin, the swap which leads to the greatest reduction is made. Similarly, for bins where no swaps are possible there is no action taken. When applying the local search operator on partial solutions, only the items which had been packed, i.e. only those items in the bins within the partial solution, were considered when attempting to make improvements. In addition to this, the operator was only applied in cases where two or more bins were present (no action was taken otherwise). The reasoning behind this is that in cases of only a single bin, the effect of Algorithm 5 would be the equivalent of unpacking the items in the bin and repacking them using the First Fit Descending construction heuristic. We felt that in such cases this would have negatively impacted on the performance of the hyper-heuristic search, particularly in the case of the ISA approach.

Algorithm 5. Local Search Operator

```
1: $free \leftarrow$ items from least filled bin

2: for $i \leftarrow 1, n$ do                    ▷ $n$ = number of remaining bins
3:     Swap two items packed in the $i^{\text{th}}$ bin with two items in $free$ if the residual
       capacity of the bin is reduced after the swap
4: end for

5: for $i \leftarrow 1, n$ do                    ▷ $n$ = number of remaining bins
6:     Swap two items packed in the $i^{\text{th}}$ bin with one item in $free$ if the residual
       capacity of the bin is reduced after the swap
7: end for

8: for $i \leftarrow 1, n$ do                    ▷ $n$ = number of remaining bins
9:     Swap one item packed in the $i^{\text{th}}$ bin with one item in $free$ if the residual capacity
       of the bin is reduced after the swap
10: end for

11: while $free$ contains items do
12:     remove the first item from $free$ and pack it using the First Fit Descending
        construction heuristic
13: end while
```

The performance of each of the three approaches was compared to one another, as well as with a standard GA searching only the heuristic space (GAHH), using the non-parametric Friedman test with the Nemenyi post-hoc test, as proposed by [2].

3.1 Technical Specifications

The simulations were performed using the Centre for High Performance Computing's (CHPC) Lengau cluster[2]. A total of 30 runs were performed for each problem instance, per approach, using an average of 12 compute threads/ cores (at a clock speed of 2.6 GHz) per run. The simulations were set up such that there were no restrictions on the memory.

4 Results and Analysis

In this section, the results obtained from each of the three approaches outlined previously are compared with each other as well as with those from a standard

[2] https://www.chpc.ac.za/index.php/resources/lengau-cluster.

GAHH (searching only the heuristic space). For each of the four approaches, the number of problem instances which were solved to optimality, the number of problem instances which were solved to near-optimality (one bin more than the optimum), and the number of problem instances that were more than one bin from the optimum are all reported in Table 2. The results are grouped according to the data set categories (easy, medium and hard), with the best performing approach in bold.

Table 2. Table showing the number of problem instances, for each of the data set categories, which were solved to optimality or near-optimality (one bin from the optimum), for each of the methods tested. The results in bold are for the best performing algorithm.

Prob. Set	Algorithm	No. opt	% at Opt	No. (Opt-1)	Sum	No. rem
Easy (720)	GAHH	590	81.9 %	82	672	48
	SSA	626	86.9%	63	689	31
	ISA	632	87.8%	79	711	9
	CSA	**673**	**93.5%**	**44**	**717**	**3**
Medium (480)	GAHH	242	50.4 %	121	363	117
	SSA	281	58.5%	94	375	105
	ISA	371	77.3%	67	438	42
	CSA	**428**	**89.2%**	**40**	**468**	**12**
Hard (10)	GAHH	0	00.0 %	0	0	10
	SSA	0	00.0%	0	0	10
	ISA	2	20.0%	6	8	2
	CSA	**6**	**60.0%**	**4**	**10**	**0**
Total (1210)	GAHH	832	68.8%	203	1035	175
	SSA	907	75.0%	157	1064	146
	ISA	1005	83.1%	152	1157	53
	CSA	**1107**	**91.5%**	**88**	**1195**	**15**

The performance of the approaches was ranked for each problem instance, where the performance was assessed based on the number of bins in the found solution relative to the known optimum. Solutions that were closest to the known optimum were assigned the highest rank (a value of 1, indicating the best performing approach), with solutions furthest from the optimum receiving the lowest rank (a value of 4). In the cases of ties, the average rank was assigned. Table 3 shows an example of the rank assignments for some of the problem instances.

As proposed by [2], the non-parametric Friedman test was used to determine if there is a statistically significant difference in the average ranks of each algorithm. Using the values provided in Table 4, the F-statistic evaluates to $F_F = 61.10934$. Using 4 algorithms across 1210 problem instances,

Table 3. Table showing example rank assignments for each algorithms' performance.

Prob. instance	Algorithm	Bins to Opt	Rank
N4C3W4_R	GAHH	5	4
	SSA	2	3
	ISA	1	2
	CSA	0	1
N3C3W4_Q	GAHH	2	3.5
	SSA	2	3.5
	ISA	1	2
	CSA	0	1

the value of F_F is distributed according to the F-distribution with $4 - 1 = 3$ and $(4 - 1) \times (1210 - 1) = 3627$ degrees of freedom. For an α-level of 5%, this leads to a critical value of $F_{0.05}(3, 3627) = 2.60736$, and we can therefore reject the null hypothesis which states that the algorithms are equivalent (since $F_F > F_{0.05}(3, 3627)$).

Table 4. Table showing the average rank and its square for each method implemented. Averages were calculated across all problem instances.

Algorithm	Avg. rank	(Avg. rank)^2
GAHH	2.82397	7.97479
SSA	2.63719	6.95477
ISA	2.34174	5.48373
CSA	2.19711	4.82728

Proceeding with the Nemenyi post-hoc test, for an α-level of 5% the critical difference evaluates to $CD = 0.13484$. Therefore any two algorithms are significantly different if their corresponding average ranks differ by at least 0.13484. The differences in the average ranks between each of the methods are presented in Table 5. From Table 5 one can see that the performance of each of the approaches are significantly different from one another, as all the values are greater than the critical difference.

The average runtimes per problem instance for each of the approaches implemented are reported in Table 6, from which one can see that there is a considerable increase in the runtimes when incorporating local search into the algorithm. From Table 6 one can see that the SSA has, on average, more than double the runtime of the GAHH, with the ISA and CSA taking considerably longer than the SSA.

However, as previously mentioned, the CSA approach significantly outperforms the three other approaches and, as can be seen from Table 6, has over half

Table 5. Table showing the differences in average ranks between each of the methods implemented.

	GAHH	SSA	ISA	CSA
GAHH	–	0.18678	0.48223	0.62686
SSA	0.18678	–	0.29545	0.44008
ISA	0.48223	0.29545	–	0.14463
CSA	0.62686	0.44008	0.14463	–

Table 6. Table showing the average runtime per problem instance, for each of the four approaches implemented. The averages across each of the problem difficulty categories are reported, as well as the average across all problem instances.

Problem category	Average runtime per problem instance			
	GAHH	SSA	ISA	CSA
Easy	13.5 s	21.6 s	53 min 05.5 s	13 min 37.5 s
Medium	06.0 s	30.6 s	41 min 47.3 s	26 min 29.8 s
Hard	12.7 s	34.4 s	16 min 37.4 s	29 min 21.1 s
Total	10.5 s	25.3 s	48 min 18.4 s	18 min 51.6 s

the runtime of the next best performing approach. Although the CSA produces the best quality solutions, it cannot be ignored that the runtime is considerably longer than either the SSA or GAHH approaches (over 40 and 100 times longer respectively). Therefore, it is necessary to consider the required quality of solutions, as well as any time constraints that may be present for obtaining said solution (i.e.: obtaining "good enough" solutions relatively quickly versus long runtimes to obtain optimal solutions).

4.1 Comparison with the State of the Art

As the main aim of this research is not to compete with state of the art approaches at this stage, but rather to provide an alternative approach for bi-space search, a very simple local search operator has been used to explore the solution space. Hence, we do not expect the approaches presented in the paper to outperform state of the art approaches. However, for the sake of completeness we present a comparison of the performance of our approach against those taken from the literature. These include:

– HI-BP [1]: uses complex heuristics that both construct and improve solutions
– PERT-SAWMBS [5]: uses complex heuristics that both construct and improve solutions
– EXON-MBS-BFD [3]: uses a grouping genetic algorithm together with complex construction heuristics

- GGA-CGT [12]: uses a grouping genetic algorithm to explore the solution space
- IPGGA [6]: uses a grouping genetic algorithm to explore the solution space

The number of problem instances that were solved to optimality are presented in Table 7 for each of these methods.

Table 7. Table showing the number of instances solved to optimality for the CSA approach compared with those taken from the literature.

Method	Easy (/720)	Medium (/480)	Hard (/10)
HI-BP	**720**	**480**	**10**
PERT-SAWMBS	**720**	**480**	**10**
EXON-MBS-BFD	667	412	8
GGA-CGT	**720**	**480**	**10**
IPGGA	**720**	**480**	**10**
CSA	673	428	6

From Table 7 one can see that the proposed CSA approach does not compare well with the state of the art. Only the EXON-MBS-BFD is outperformed by the CSA, and only for the easier categories. This is to be expected because the GGA-CGT, EXON-MBS-BFD and IPGGA all implement a grouping genetic algorithm [3,6,12], which is a modification of the canonical GA for the express purpose of solving grouping problems [6], such as the 1BPP. In addition to this, the majority of the optimisation for the CSA occurs in the heuristic space. The CSA only implements very simple heuristics and move operator as opposed to the more complex heuristics used by HI-BP and PERT-SAWMBS. Hence it is understandable that the CSA performs poorly when compared with the state of the art. Future work will look at incorporating more robust search techniques, such as those used in the state of the art approaches, for exploring the search space, in the bi-space search.

Furthermore, from the *no free lunch* theorem it is known that the performance of methods differs according to the problem domain and search space. With this in mind, future work will investigate the effectiveness of selecting a search technique according to the search space and problem domain at hand, as well as including additional spaces such as the design space (ie: a multi-space search).

5 Conclusions and Future Work

A new alternative approach to bi-space search was proposed, namely the concurrent search approach (CSA). This approach was investigated using the heuristic and solution spaces, and its performance was compared with two other approaches taken from the literature (termed the sequential search approach (SSA) and interleaving search approach (ISA)), as well as with a purely heuristic space search (implemented using a GA). The Scholl benchmark datasets for the one dimensional bin packing problem were used.

Experimental results showed each of the approaches' performance were significantly (within the 95% confidence interval) different from one another, with the newly proposed approach being the best performing. All the bi-space search approaches outperformed the single-space search, thus highlighting the potential benefits of bi-space search.

Although there is a significant improvement in the quality of the solutions obtained, there is also a considerable increase in the runtimes, particularly when performing local search during the solution construction process (the ISA and CSA approaches). It is to be noted that although these approaches have longer runtimes, the proposed CSA runs over 2.5 times faster than the ISA taken from the literature. Hence, careful consideration needs to be taken with respect to both the desired quality of solutions ("good enough" versus optimal solutions) and the time available in which to find said solutions (upper bounds on the runtime).

Results from the CSA approach were also compared with state of the art approaches for the one-dimensional bin packing problem (1BPP). The intent when implementing the CSA approach was not to solve the 1BPP, thus the CSA did not perform as well as the state of the art, as was expected. Future work will investigate adapting/selecting the search technique according to the current search space at hand, as well as the use of techniques such as neighbourhood landscape analysis to better decide when to search a given space. Search across more than two spaces (ie: a multi-space search) will also be investigated.

Acknowledgements. This work was funded as part of the Multichoice Research Chair in Machine Learning at the University of Pretoria, South Africa. The authors acknowledge the Centre for High Performance Computing (CHPC), South Africa, for providing computational resources toward this research.

References

1. Alvim, A.C., Ribeiro, C.C., Glover, F., Aloise, D.J.: A hybrid improvement heuristic for the one-dimensional bin packing problem. J. Heuristics **10**(2), 205–229 (2004). https://doi.org/10.1023/b:heur.0000026267.44673.ed
2. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
3. Dokeroglu, T., Cosar, A.: Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms. Comput. Ind. Eng. **75**, 176–186 (2014). https://doi.org/10.1016/j.cie.2014.06.002
4. Falkenauer, E.: A hybrid grouping genetic algorithm for bin packing. J. Heuristics **2**(1), 5–30 (1996). https://doi.org/10.1007/bf00226291
5. Fleszar, K., Charalambous, C.: Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. Eur. J. Oper. Res. **210**(2), 176–184 (2011). https://doi.org/10.1016/j.ejor.2010.11.004

6. Kucukyilmaz, T., Kiziloz, H.E.: Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. Comput. Ind. Eng. **125**, 157–170 (2018). https://doi.org/10.1016/j.cie.2018.08.021
7. Levine, J., Ducatelle, F.: Ant colony optimization and local search for bin packing and cutting stock problems. J. Oper. Res. Soc. **55**(7), 705–716 (2004). https://doi.org/10.1057/palgrave.jors.2601771
8. López-Camacho, E., Terashima-Marin, H., Ross, P., Ochoa, G.: A unified hyper-heuristic framework for solving bin packing problems. Expert Syst. Appl. **41**(15), 6876–6889 (2014). https://doi.org/10.1016/j.eswa.2014.04.043
9. Pillay, N., Beckedahl, D.: EvoHyp - a Java toolkit for evolutionary algorithm hyper-heuristics. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 2706–2713, June 2017. https://doi.org/10.1109/CEC.2017.7969636
10. Pillay, N., Qu, R.: Hyper-Heuristics: Theory and Applications. Natural Computing Series. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96514-7
11. Qu, R., Burke, E.K.: Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. J. Oper. Res. Soc. **60**(9), 1273–1285 (2009). https://doi.org/10.1057/jors.2008.102
12. Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J.S., Gomez, C.G., Huacuja, H.J.F., Alvim, A.C.: A grouping genetic algorithm with controlled gene transmission for the bin packing problem. Comput. Oper. Res. **55**, 52–64 (2015). https://doi.org/10.1016/j.cor.2014.10.010
13. Scholl, A., Klein, R., Jürgens, C.: Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Comput. Oper. Res. **24**(7), 627–645 (1997). https://doi.org/10.1016/s0305-0548(96)00082-2