A Study of Evolutionary Algorithm Selection Hyper-Heuristics for the One-Dimensional Bin-Packing Problem

Nelishia Pillay[1]

School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, South Africa.

ABSTRACT

Hyper-heuristics are aimed at providing a generalized solution to optimization problems rather than producing the best result for one or more problem instances. This paper examines the use of evolutionary algorithm (EA) selection hyper-heuristics to solve the offline one-dimensional bin-packing problem. Two EA hyper-heuristics are evaluated. The first (EA-HH1) searches a heuristic space of combinations of low-level construction heuristics for bin selection. The second (EA-HH2) explores a space of combinations of both item selection and bin selection heuristic combinations. These EA hyper-heuristics use tournament selection to choose parents, and mutation and crossover with hill-climbing to create the offspring of each generation. The performance of the hyper-heuristics is compared to that of each of the low-level heuristics applied independently to solve this problem. Furthermore, the performance of both hyper-heuristics is also compared. The comparisons revealed that hyper-heuristics in general perform better than any single low-level construction heuristic in solving the problem. In addition to this it was found that the hyper-heuristic exploring a space of both item selection and bin selection heuristic combinations is more effective than the hyper-heuristic searching a space of just bin selection heuristic combinations. The performance of this hyper-heuristic was found to be comparable to other methods applied to the same benchmark sets of problems.

CATEGORIES AND SUBJECT DESCRIPTORS

I.2. [**Computing Methodologies**]: Artificial Intelligence.

GENERAL TERMS

Algorithms, Theory

KEYWORDS

Hyper-heuristics, one-dimensional bin-packing, evolutionary algorithms

1. INTRODUCTION

The one-dimensional bin-packing problem requires a set of items to be stored in bins, each having the same capacity, with a minimum number of bins being used. This problem has numerous real-world applications including machine scheduling, LSVI chip layout, one-dimensional stock cutting, cable-length optimization, and processor task allocation. Hence, this problem has been fairly well-studied and various methods such as genetic algorithms, tabu search, and simulated annealing have been applied to this domain. These methods have generally focused on producing the best result for one or more problems in a benchmark problem set. Hyper-heuristics on the other hand aim at providing generalized solutions to problems instead of producing best results [18]. The advantage of this is that solutions are found for a set of problems instead of the best solution for one or two problems in the set. This paper examines the use of constructive selection hyper-heuristics to solve the offline one-dimensional bin-packing problem. The hyper-heuristics presented in this paper employ an evolutionary algorithm to explore a space of low-level construction heuristic combinations. Two hyper-heuristics, one that searches a heuristic space of bin selection heuristic combinations (EA-HH1), and a second that explores a heuristic space of both item and bin selection heuristic combinations (EA-HH2), are examined. To the best of the author's knowledge, item selection heuristics have not previously been defined for this domain. The performance of both hyper-heuristics is compared in solving problems from the Faulkenauer and Scholl benchmark sets. The solutions produced by both hyper-heuristics are also compared to those obtained by using each low-level heuristic independently in solving these problems.

The following section presents the one-dimensional bin-packing problem. Hyper-heuristics is introduced in section 3 and an overview of previous work applying hyper-heuristics to solve the one-dimensional bin-packing problem is provided. Section 4 describes the evolutionary algorithm hyper-heuristics implemented to solve this problem and section 5 specifies the experimental setup used to evaluate the hyper-heuristics. The performance of the hyper-heuristics in solving the one-dimensional bin-packing problem is discussed in section 6. The

[1] pillayn32@ukzn.ac.za

findings of this study are summarized in section 7 together with future extensions of the research presented.

2. THE ONE-DIMENSIONAL BIN-PACKING PROBLEM

The one-dimensional bin-packing problem (BPP) involves placing a set of items of different sizes into one or more bins. All the bins have the same capacity and the main aim of the problem is to minimize the number of bins used to store all the items. There are two versions of the bin-packing problem, namely, the offline BPP and the online BPP [21]. In the offline version the size of all the items is known before the placement process begins. In the online version the size of each item is only known when it is being placed. Initially, sequential construction methods employing low-level construction heuristics were used to find solutions to this problem. This led to the derivation of a number of low-level construction heuristics for this domain including first-fit [9], best-fit [3], next-fit [13], worst-fit [6], better-fit [3] and Djang and Finch[2] [19] heuristics. These heuristics are bin selection heuristics and are used to determine which bin to place the next item in. Variations of these heuristics have also been implemented in which the items to be stored are sorted in decreasing order according to size and allocated accordingly, e.g. first-fit decreasing and best-fit decreasing [21].

Various methods have been applied to solving the one-dimensional bin-packing problem. One of the earliest contributions to this field is the grouping genetic algorithm developed by Faulkenauer [10] to solve the one-dimensional bin-packing problem. The GA employs the steady-state control model and uses the tournament selection method with a tournament size of two to select parents. Instead of evolving a population over a set number of generations, the steady-state control model replaces poorly performing individuals with the newly created offspring. The fitness of each individual is a function of the fullness of each bin, the maximum capacity of the bin and the number of bins used to store the items. The mutation, crossover and inversion operators are used to create offspring. Jing et al. [12] apply a genetic algorithm in combination with the next-fit heuristic to solve a variation of the one-dimensional bin-packing problem in which each bin has a different capacity. The fitness function is the same as that used in the Faulkenauer study [10]. Crossover and mutation are used to create offspring. Ulker et al. [22] also use a grouping genetic algorithm, employing linear linkage encoding for representation purposes, to obtain solutions to the one-dimensional BPP. An initial population of potential solutions is created using the first-fit heuristic. Tournament selection is used to choose parents. Smart mutation and crossover are applied to the chosen parents to create the offspring of each generation. The fitness of each individual is calculated using the evaluation function proposed by Faulkenauer [10]. The method was successfully applied to two of the Faulkenauer data sets.

Kasap et al. [14] use a neural network together with the first fit decreasing heuristic to solve one-dimensional bin-packing problems from the Scholl benchmark set. The bin-packing problem is modeled as a neural network. The neural network has a dummy and output node layer and between both these layers is an item layer and bin layer. This approach was found to find the minimum number of bins for a majority of the problems.

[2] This refers to a heuristic algorithm developed by Djang and Finch [19] which first allocates items, with the largest items receiving priority, until three-quarters of the bin is full. At this stage different combinations of items are examined to fill the bin.

Flezar et al. [11] firstly apply a minimum bin slack heuristic to create an initial solution to the problem. This solution is then optimized using variable neighbourhood search.

Lewis [15] presents the bin-packing problem as a minimum grouping problem and proposes a hill-climbing grouping algorithm to solve this problem. The algorithm firstly creates an initial solution using the first-fit descending heuristic. The hill-climbing algorithm improves this initial solution by performing swaps between bins. The algorithm was successfully applied to the set of Faulkenauer benchmark problems.

Loh et al. [16] uses a weight annealing approach, similar to simulated annealing, to solve the one-dimensional bin-packing problem. As with methods previously described an initial solution is created using the first-fit decreasing heuristic and then improved using weighted annealing. The method outperformed existing approaches applied to this problem.

Valerio de Carvalho [23] represents the one-dimensional bin-packing problem as a flow formulation model which a branch-and-bound algorithm is applied to. A linear relaxation of the model is also examined. This method produced results comparable to other methods applied to the Faulkenauer problem set.

Hybrid approaches have also been used to solve the one-dimensional bin-packing problem. Alvim et al. [1] solve the one-dimensional BPP using a hybrid method incorporating the use of tabu search, lower bounding strategies and load distribution based on dominance. Kao et al. [13] take a hybrid approach, combining simulated annealing and genetic algorithms, to solve the one-dimensional BPP. The fitness of each potential solution was the inverse of the number of bins used and the known minimum for the problem instance. The approach was applied to ten problems with a maximum bin capacity of 1000 and item weights in the range 1 to 999. This hybrid method produced competitive results when compared to the performance of the first-fit heuristic. Scoll et al. [21] present a hybrid approach, namely BISON, to solve the one-dimensional bin-packing problem which combines a variation of the branch and bound algorithm MTRP with new bounds and dominance rules, tabu search and a depth-first search branch and bound method. BISON was applied to three data sets and was found to outperform the standard MTRP in solving these problems.

3. HYPER-HEURISTICS

Most of the research aimed at solving combinatorial optimization problems has focused on developing methods that produce the best solutions for one or more problem instances of benchmark sets rather than providing a generalized solution to the problem. Hyper-heuristics have been developed with this in mind and focus on producing general solutions to problems instead of optimal solutions for a few problem instances [5]. This is achieved by exploring a heuristic space rather than a solution space. The heuristic space is comprised of low-level heuristics or combinations of heuristics. These heuristics can be constructive, i.e. they are used in the construction of a solution to the problem, or perturbative, i.e. they represent moves that improve an initially created potential solution to the problem [5]. Construction heuristics are domain dependent, e.g. the first-fit, best-fit, next-fit and worst-fit heuristics for the one-dimensional bin-packing problem and the largest degree, largest weighted degree, largest enrolment and saturation degree for university course and examination timetabling problems [9, 18]. Hyper-heuristics employ methodologies such as variable neighbourhood search, tabu search and genetic programming to either select a low-level heuristic or combination of heuristics or generate a new low-level heuristic for a problem domain.

Thus, in addition to being either constructive or perturbative, hyper-heuristics are also classified as selection or generation hyper-heuristics [8]. The study presented in this paper investigates the use of construction, selection hyper-heuristics for solving the one-dimensional bin-packing problem.

There has not been much research into the use of construction, selection hyper-heuristics for the one-dimensional bin-packing problem. There are two studies that examine construction, selection hyper-heuristics for this domain. The first study uses a learner classifier system to generate condition-action rules for heuristic selection [20]. The rules use four low-level heuristics, namely, first-fit decreasing, next-fit decreasing, the Djang and Finch heuristic with 3 items and the Djang and Finch heuristic with 5 items. The only study implementing a genetic algorithm to explore a heuristic space is that conducted by Ross et al. [19]. Each chromosome is a variable length string comprised of numbers representing the size of items still to be placed and the heuristic to choose a bin i.e. first-fit decreasing, next-fit decreasing or the Djang and Finch heuristic. The first five numbers represent the proportion of huge, large, medium and small items still to be packed and the number of remaining items to be packed. The sixth number represents the heuristic. The GA implements the steady-state control model. Tournament selection with a tournament size of 2 is used to choose parents which two crossover and three mutation operators are applied to, to create the next generation. The fitness of each chromosome is a measure of the performance of the chromosome in solving a set of five problems compared to using the individual heuristics to solve the problems.

There has been research conducted into using hyper-heuristics, other than construction, selection hyper-heuristics, to solve the one-dimensional bin-packing problem. For completeness these are included. Burke et al. [6,7] have implemented a generative hyper-heuristic to induce low-level construction heuristics for this problem. The hyper-heuristic uses genetic programming to evolve construction heuristics. Each element of the population is a parse-tree representing the heuristic and is comprised of elements from the function set and terminal set. The function set includes addition, multiplication, protected division and the absolute value operators. The terminal set consists of constants representing the characteristics of the problem, namely, the fullness of the bin, i.e. the sum of the size of the items in the bin, the size of the next item to place and bin capacity. The fitness of each parse tree was calculated to be the difference of the number of bins used and the ratio of the sum of the size of each piece in a bin divided by the total bin capacity. The fittest heuristic evolved by the GP system was found to perform the same function as the first-fit heuristic.

Some research has also been conducted into the use of selection perturbative hyper-heuristics to solve the one-dimensional bin-packing problem. Bai et al. [2] have used a simulated annealing selection hyper-heuristic to explore the heuristic space comprised of perturbative low-level bin-packing heuristics, e.g. shift, split, exchange. The hyper-heuristic uses stochastic selection strategies in combination with short-term memory. Burke et al. [4] have proposed a perturbative hyper-heuristic framework, namely, Hyperflex. Initial solutions to the bin-packing problem are created using the first-fit heuristic and are improved using the perturbative hyper-heuristic. The moves or perturbative heuristics used by the hyper-heuristic include two mutational heuristics, two create and ruin heuristics, a repack with best-fit heuristic and three local search heuristics.

It is evident from the survey presented in this section that there has not been much research conducted into the use of construction, selection hyper-heuristics to solve this problem. The only study using evolutionary algorithms to explore a heuristic space is that conducted by Ross et al. [19] in which a genetic algorithm is used to search a space of chromosomes where each chromosome is a string of numbers representing the number of items still to be placed, the sizes of these items and the bin selection heuristic. A different approach, which is similar to the construction hyper-heuristics implemented by Els et al. [8] and Pillay [18] for the university course and examination timetabling problems respectively, is taken in this study. The following section describes these construction, selection hyper-heuristics implemented.

4. THE EA HYPER-HEURISTICS

This section describes the hyper-heuristics used in this study to solve the one-dimensional bin-packing problem. In the first EA hyper-heuristic (EA-HH1), each chromosome is a string consisting of letters representing the low-level construction heuristics that will be used to decide which bin to place the next item in when creating a solution. This study differs from that conducted by Ross et al. [19, 20] in that item selection heuristics are introduced and the study investigates the use of a hyper-heuristic optimizing a space of combinations of both item and bin selection heuristics (EA-HH2) in solving the one-dimensional BPP.

The evolutionary algorithm implemented by both EA hyper-heuristics employs the generational control model and is depicted in Figure 1. Note that the evolutionary process is terminated when a solution with a number of bins equal to the minimum is found. A minimum is specified for each problem in both the Faulkenauer and Scholl benchmark sets. For example, for the Faulkenauer problems this is a theoretical minimum calculated to be the total capacity required divided by the capacity of a single bin. The minimum specified for each problem in the benchmark set is used as a termination criterion. The algorithm is given a preset number of generations within which to achieve this. Sections 4.1 and 4.2 describe processes of initial population generation, evaluation, selection, and the genetic operators for EA-HH1 and EA-HH2 respectively.

```
Create an initial population
Repeat
  Evaluate the population.
  Select parents for the next generation.
  Apply mutation and crossover to the chosen
  parents to create offspring.

Until a solution with the minimum number of bins is
found OR the generation limit is reached.
```

Figure 1. Evolutionary algorithm implemented by EA-HH1 and EA-HH2

4.1 EA-HH1

This section presents the first hyper-heuristic implemented which searches a space of combinations of low-level bin selection heuristics such as first-fit and best-fit. Two versions of this hyper-heuristic, namely, EA-HH11 and EA-HH12, have been implemented. EA-HH11 uses the availability index to determine which item to store next and EA-HH12 uses the saturation degree to select the next item. The availability and saturation degree heuristics are defined in section 4.2.1. These heuristics are used instead of storing items in decreasing order of size.

4.1.1 Initial population generation

Each element of the population is a variable length string consisting of characters representing the low-level bin selection heuristics. The following bin selection heuristics were used as these were the most commonly cited in the literature:

- First-fit (*f*) –The item to be placed is allocated to the first bin that has sufficient space for it to fit into.
- Best-fit (*b*) – The item is placed into the bin that will have the smallest residual space once the item is placed into it.
- Next-fit (*n*) – If the item does not fit into the current bin, i.e. the last bin to be created, it is placed in a new bin.
- Worst-fit (*w*) – The item to be placed is allocated to the bin with the largest residual space once the item is placed into it.

Trial runs indicated that the "decreasing" [21] variation of these heuristics performed better than the standard heuristics and thus this variation was implemented. The decreasing version of these heuristics requires that items be sorted in decreasing order of size and be allocated accordingly.

Duplicate chromosomes are not permitted in the initial population and thus the initial population has a variation of 100%. An example of an element of the population is *bbfnw*. Using this chromosome to create a solution to the problem, the first two items will be allocated to a bin chosen using the best-first heuristic, the third item using the first-fit heuristic, the fourth item the next-fit heuristic and the worst-fit heuristic will be used to choose a bin for the last item. The length of the chromosomes is variable and chosen to be between the range of one and the maximum number of items to be scheduled. If the length of the chromosome is smaller than the number of items to be stored, the heuristics are reapplied from the beginning of the string. An explanation of how the fitness of each chromosome is calculated is explained in the following section.

4.1.2 Fitness evaluation and selection

The fitness of each chromosome is determined by using it to solve the one-dimensional bin-packing problem for the particular problem instance. As described in the previous section each chromosome is used to decide which heuristic to use when selecting a bin to place an item into. The fitness of a chromosome is a function of the number of bins used to store the items in the solution. Previous studies [10, 22] have found that using the number of bins as a fitness measure is not effective as the resulting fitness landscape contains numerous local optima that use one more than the optimal number of bins. Faulkenauer et al. [10] proposed the following evaluation function which is generally used to assess solutions for the one-dimensional BPP:

$$\frac{\sum_{i=1}^{N}(\frac{F_i}{C})^2}{N}$$

where: N - is the number of bins
C – maximum capacity of a bin
F_i – is the fullness of the bin

The fullness of a bin is the sum of the sizes of the items in the bin. This evaluation function is used to calculate the fitness of the chromosomes. A fitter individual has a higher fitness value, i.e. the fitness is maximized.

The tournament selection method is used to select parents to apply genetic operators to. Individuals are randomly chosen from the population to form a tournament. The most appropriate size of the tournament is problem dependant. An individual may be chosen more than once to be an element of the same tournament. The fittest element of the tournament is deemed the winner and becomes a parent of the next generation. Selection is with replacement so an individual can be chosen as a parent more than once.

4.1.3 Genetic operators

The mutation and crossover operators are used to create the offspring of each generation.

The mutation operator randomly replaces a low-level heuristic in a copy of the parent with a newly created heuristic substring. The length of the substring is randomly chosen to be between 1 and the length of the parent and is created using the same method used to create each element of the initial population. An example is illustrated in Figure 2.

Parent: *wnf**n**b*
New substring: *bbw*
Offspring: *wnf**bbw**b*

Figure 2. An example of mutation

The mutation point is randomly chosen to be 4. The substring rooted at this point, namely, *n* is replaced with the newly created substring. Note that the newly created substring could have been an *n* which reduces the mutation operator to reproduction. A version of this operator incorporating hill-climbing was also tested. The hill-climbing mutation only accepts mutations of the parent that are fitter than the parent. If the first mutation operation does not result in a fitter offspring, the offspring is discarded and the process is repeated until a fitter offspring is produced. A limit is set on the number of attempts at producing a fitter offspring in order to prevent convergence to a local optimum and high runtimes. A value of 30 was found to be sufficient for the study. If a fitter offspring is not found within the 30 attempts, the offspring created on the thirtieth attempt is returned as the result of the mutation. In trial runs conducted the hill-climbing mutation operator performed better than the standard mutation operator and thus it was decided to use the mutation operator incorporating hill-climbing.

The crossover operator is applied to two parents chosen using tournament selection. Crossover points are randomly selected in copies of both the parents and the fragments divided by the crossover points are swapped. An example is illustrated in Figure 3.

Parent 1: *bb**n**ff* **Parent 2:** *w**b**nfw*
Offspring 1: *bbbnfw* **Offspring 2:** *wnff*

Figure 3. An example of crossover

The crossover point is randomly chosen to be 3 in the first parent and 2 in the second parent. Copies of the parents are "crossed over" at these points producing two offspring. Previous work with EA hyper-heuristics of this type has revealed that returning the fitter of the two offspring is more effective than returning both offspring [17]. The same approach is adopted in this study and the crossover operator returns one offspring. As with the mutation operator, the use of hill-climbing in the crossover operator was found to produce better results. In this case if the offspring is not fitter than both parents than the process is repeated. Again, a limit was set on the number of attempts at producing a fitter offspring and a value of 30 was also sufficient for crossover.

4.2 EA-HH2

This section describes the second hyper-heuristic which searches a space of combinations of both bin selection and item selection heuristics.

4.2.1 Initial population generation

In this case each chromosome is comprised of two strings. The first string, the bin selection heuristic combination, is essentially the chromosome described in section 4.1.1, i.e. a string combining bin selection heuristics. Note that the standard bin selection heuristics are used for EA-HH2, and not the decreasing variation of these heuristics. The reason for this is that a heuristic is used to choose which item to store next. The second string, the item selection heuristic combination, in the chromosome is comprised of the following item selection heuristics for this purpose:

- Largest item (*l*) – The item with the largest size is allocated next.
- Availability degree (*a*) – The first item that can fit into an existing bin is allocated next. Here the aim is to fill existing bins first, before opening a new bin.
- Saturation degree (*s*) – The item that has the fewest number of bins available that it will fit in, is given priority. Here an analogy is taken from the examination timetabling domain where the saturation degree of an examination defines the number of feasible periods in the timetable the examination can be scheduled in [17].

Both strings are of variable length and the length of each string is randomly chosen, independently of each other, to be in the range of 1 and the number of items to be stored. If the length of either string is less than the number of items to be allocated, the string is wrapped around, beginning at the first character in the string again, i.e. the heuristics are applied in order again beginning with the first heuristic in the string. As with EA-HH1 duplicates[3] are not permitted in the initial population. An example of an element of the population is *wnlbbf;aasll*. A solution to the problem constructed using this chromosome will result in the first item being chosen using the availability degree heuristic and the bin that it will be stored in will be selected using the worst-fit heuristic. Similarly, the sixth item to be stored will be chosen using the availability degree heuristic (as the string will be wrapped over) and the bin will be chosen according to the first-fit heuristic. The fitness of each chromosome is determined by using it to solve the instance of the bin-packing problem. EA-HH2 uses the same fitness function and selection method as EA-HH1 (described in section 4.1.2). The following subsection describes the genetic operators used by EA-HH2.

4.2.2 Genetic operators

Two mutation and crossover operators were implemented for use with EA-HH2. The first set of mutation and crossover operators extend the hill-climbing mutation and crossover operators implemented by EA-HH1 to randomly select whether to mutate/recombine the bin selection heuristic combination/s or the item selection heuristic combination/s. The second set of operators are also a variation of the EA-HH1 hill-climbing and mutation and crossover operators in which both the bin heuristic selection combination/s and the item selection combination/s are mutated and recombined respectively. In order to ascertain which operators are more effective, EA-HH2 will be tested with both the first set, will be referred to as EA-HH21, and the second set, EA-HH22, of operators.

[3] A chromosome is a duplicate of a second chromosome if both the strings in the chromosome, i.e. the bin selection heuristic combination and the item selection heuristic combination, are the same.

5. METHODOLOGY

This section describes the experimental setup used to test the EA hyper-heuristics. The performance of the EA hyper-heuristics is compared in solving 50 one-dimensional bin-packing problems. The performance of these hyper-heuristics is also compared to that of each of the low-level heuristics applied independently to solve these problems. The first 20 problems are taken from the Faulkenauer benchmark set [10]. In all 20 of these problems 120 items, with the size of each item ranging from 20 to 100, must be stored in a minimum number of bins, each of which has a maximum capacity of 150. The remaining 30 problems are from the Scholl benchmark set [21]. This range of problems was chosen to test the hyper-heuristics on problems of varying complexity. The characteristics of these problems are listed in Table 1.

Table 1. Characteristics of problems from the Scholl benchmark set

Problem	No. of Items	Bin capacity	Range of size of items
1	50	100	1-100
2	50	100	20-100
3	50	100	30-100
4	50	120	1-100
5	50	120	20-100
6	50	120	30-100
7	50	150	1-100
8	50	150	20-100
9	50	150	30-100
10	100	100	1-100
11	100	100	20-100
12	100	100	30-100
13	100	120	1-100
14	100	120	20-100
15	100	120	30-100
16	100	150	1-100
17	100	150	20-100
18	100	150	30-100
19	200	100	1-100
20	200	100	20-100
21	200	100	30-100
22	200	120	1-100
23	200	120	20-100
24	200	120	30-100
25	200	150	1-100
26	200	150	20-100
27	200	150	30-100
28	500	100	1-100
29	500	100	20-100
30	500	100	30-100

Table 2. Parameter values used by EA-HH1 and EA-HH2

Parameter	Value
Population size	500
Number of generations	50
Tournament size	10
Crossover rate	0.5
Mutation rate	0.5

The values of the genetic parameters used by both EA hyper-heuristics are listed in Table 2. These values were determined empirically by performing trial runs. Population sizes of 100, 500, 1000 were tested. A population size of 100 did not explore enough of the heuristic space, while a population of 1000 did not improve the performance of the EA hyper-heuristics. The evolutionary algorithm was found to converge within 50 generations. Tournament sizes of 2, 4 and 10 were tested. A value of 10 produced the best results during trial runs.

Trial runs were conducted with mutation and crossover rates in the range of 0 to 1, with 0.1 intervals. Creating half of each generation using mutation and the other half using crossover proved to be the most effective combination.

The EA hyper-heuristics were written in Java using JDK 1.6.0 and all simulations were run on an Intel Core 2 Duo processor with Windows XP. Due to the stochastic nature of evolutionary algorithms, 30 runs, each using a different seed for the random number generator, were performed for each EA hyper-heuristic for each problem. The performance of the hyper-heuristics are discussed in the following section.

6. RESULTS AND DISCUSSION

This section reports on the performance of the hyper-heuristics described in the previous section in solving the one-dimensional bin-packing problem. Section 6.1 examines the performance of a hyper-heuristic approach compared to using low-level heuristics in solving this problem. A comparison of the two types of hyper-heuristics, namely, EA-HH1 and EA-HH2, is provided in section 6.2. Finally, section 6.3 compares the performance of the hyper-heuristics to methods producing the best results for the problems from both benchmark sets.

6.1 Hyper-heuristics vs. low-level heuristics

This section compares the performance of EA-HH1 with that of each of the heuristics applied independently to solve each of the 50 problems. In the case where each heuristic is applied individually, the heuristic in question is used to select a bin for each item. The items to be allocated are sorted in descending order according to size and are stored in this order. Figure 4 shows the number of problems for which EA-HH1 and each of the low-level construction heuristics have produced solutions using the known minimum number of bins. It is evident from this bar chart that the EA-HH1 has performed better than the low-level construction heuristics, producing solutions with a minimum number of bins for 43[4] of the 50 problems while the best performing heuristic, namely, the worst-fit heuristic, produced solutions with the minimum number of bins for only 32 of the problems. The best-fit, first-fit and next-fit heuristic produced solutions using the minimum number of bins for 30, 29 and 0 of the problems respectively. The best-fit, first-fit and worst-fit heuristics have produced similar results.

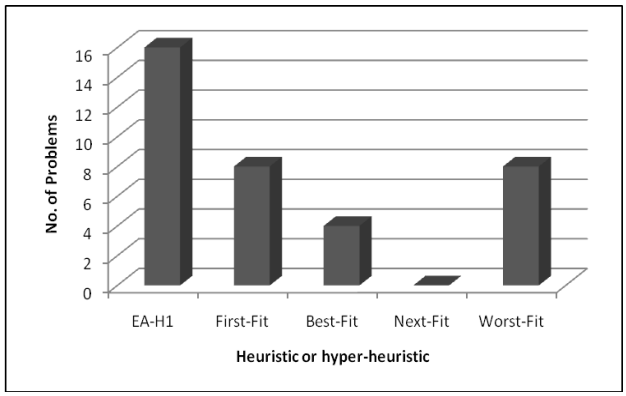

Figure 4. Performance comparison

Hypothesis tests were performed to ascertain the statistical significance of this result. The levels of significance, critical values, and decision rules for these tests are listed in Table 3 and the hypotheses and Z-values in Table 4.

Table 3. Levels of significance, critical values and decision rules

P	Critical Value	Decision Rule
0.01	2.33	Reject H_o if Z >2.33
0.05	1.64	Reject H_o if Z >1.64
0.10	1.28	Reject H_o if Z >1.28

Table 4. Hypotheses and Z values

Hypothesis	Z Values
H_o: $\mu_{EA\_HH1} = \mu_{BF}$, H_a: $\mu_{EA\_HH1} > \mu_{BF}$	3.03
H_o: $\mu_{EA\_HH1} = \mu_{FF}$, H_a: $\mu_{EA\_HH1} > \mu_{FF}$	3.25
H_o: $\mu_{EA\_HH1} = \mu_{NF}$, H_a: $\mu_{EA\_HH1} > \mu_{NF}$	17.35
H_o: $\mu_{EA\_HH1} = \mu_{WF}$, H_a: $\mu_{EA\_HH1} > \mu_{WF}$	2.60

The hypotheses that the EA-HH1 performs better than the low-level heuristics were found to be significant at the 1% level of significance for the best-first, first-fit, next-fit and worst-fit heuristics.

6.2 Hyper-heuristics performance comparison

Thirty runs were also performed for two variations of EA-HH1 for the 50 problems. EA-HH1 stores items in decreasing order according to size and the items are allocated accordingly. The first variation of EA-HH1, EA-HH11 uses the availability degree heuristic defined in section 4.2.1 to choose the next item to store instead of placing items according to size. Similarly, the second variation EA-HH12 allocates items using the saturation degree heuristic defined in section 4.2.1. Thirty runs were also performed for both the versions of EA-HH2, namely, EA-HH21 and EA-HH22 described in section 4.2, for each problem. Figure 5 illustrates the number of problems for which the hyper-heuristics were able to produce solutions that use the minimum number of bins and the success rate of the hyper-heuristics in solving each of the problems over the 30 runs is listed in Table 5. The best success rates are highlighted in bold. The hyper-heuristics that search a heuristic space of both bin selection and item selection heuristics, namely, EAHH21 and EAHH22 appear to have performed better than the other three hyper-heuristics and produced solutions with the minimum

[4] Thirty runs are performed for each problem. For 43 of the problems at least one run has produced a solution using a minimum number of bins.

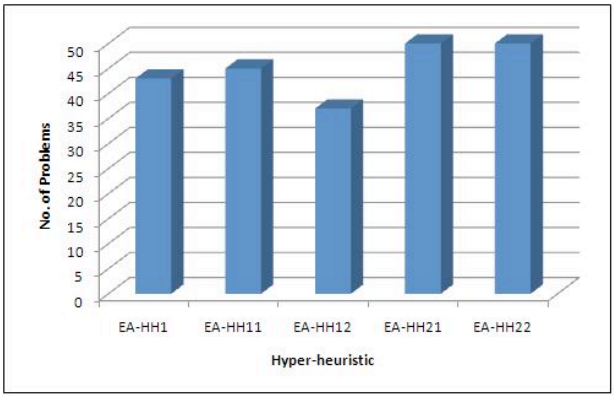

Figure 5. Hyper-heuristic performance

number of bins for all 50 problems. EA-HH1, EA-HH11, and EA-HH12 have produced solutions using the minimum number of bins for 43, 46 and 29 of the fifty problems respectively.

Table 5. Simulation results

Problem	EA-HH1	EA-HH11	EA-HH12	EA-HH21	EA-HH22
1	**100%**	**100%**	0%	**100%**	**100%**
2	**100%**	**100%**	**100%**	**100%**	**100%**
3	77%	**100%**	0%	**100%**	**100%**
4	27%	93%	0%	93%	**100%**
5	**100%**	**100%**	**100%**	**100%**	**100%**
6	**100%**	**100%**	0%	**100%**	**100%**
7	93%	**100%**	0%	**100%**	**100%**
8	0%	**73%**	0%	23%	67%
9	**100%**	**100%**	**100%**	**100%**	**100%**
10	0%	0%	0%	**27%**	13%
11	**100%**	**100%**	0%	**100%**	**100%**
12	**100%**	**100%**	0%	**100%**	**100%**
13	0%	**10%**	0%	3%	3%
14	**100%**	**100%**	**100%**	**100%**	**100%**
15	**100%**	**100%**	**100%**	**100%**	**100%**
16	**100%**	**100%**	0%	**100%**	**100%**
17	**100%**	**100%**	0%	**100%**	**100%**
18	**100%**	0%	0%	**100%**	**100%**
19	0%	**100%**	0%	**100%**	**100%**
20	**100%**	**100%**	**100%**	**100%**	**100%**
21	**100%**	**100%**	**100%**	**100%**	**100%**
22	**100%**	**100%**	**100%**	**100%**	**100%**
23	**100%**	**100%**	**100%**	**100%**	**100%**
24	**100%**	**100%**	**100%**	**100%**	**100%**
25	**100%**	**100%**	**100%**	**100%**	**100%**
26	**100%**	**100%**	**100%**	**100%**	**100%**
27	**100%**	**100%**	0%	**100%**	**100%**
28	**100%**	**100%**	**100%**	**100%**	**100%**
29	**100%**	**100%**	0%	**100%**	**100%**
30	**100%**	**100%**	**100%**	**100%**	**100%**
31	**100%**	**100%**	**100%**	**100%**	**100%**
32	**100%**	**100%**	**100%**	**100%**	**100%**
33	**100%**	**100%**	**100%**	**100%**	**100%**
34	**100%**	**100%**	**100%**	**100%**	**100%**
35	**100%**	**100%**	**100%**	**100%**	**100%**
36	**100%**	**100%**	**100%**	**100%**	**100%**
37	**100%**	**100%**	0%	**100%**	**100%**
38	**100%**	**100%**	0%	**100%**	**100%**
39	0%	0%	0%	**100%**	**100%**
40	**100%**	**100%**	**100%**	**100%**	**100%**
41	**100%**	**100%**	**100%**	**100%**	**100%**
42	**100%**	**100%**	**100%**	**100%**	**100%**
43	**100%**	**100%**	**100%**	**100%**	**100%**
44	**100%**	**100%**	**100%**	**100%**	**100%**
45	**100%**	**100%**	**100%**	**100%**	**100%**
46	**100%**	**100%**	0%	**100%**	**100%**
47	0%	0%	0%	83%	**100%**
48	0%	**100%**	**100%**	**100%**	**100%**
49	**100%**	**100%**	**100%**	**100%**	**100%**
50	**100%**	**100%**	**100%**	**100%**	**100%**

Hypothesis tests, namely, Z-tests, were conducted to test the significance of these results. The critical values and decision rules are listed in Table 3 and the hypotheses and Z-values are specified in Table 6. The performance of EA-HH21 and EA-HH22 are the same, producing solutions using the minimum number of bins for all the problems. Thus, the hypothesis tests compare the performance of EA-HH22 to EA-HH1, EA-HH11 and EA-HH12.

Table 6. Hypotheses and Z values

Hypothesis	Z Values
H_o: $\mu_{EA\_HH22} = \mu_{HH1}$, H_a: $\mu_{EA\_HH22} > \mu_{EA\_HH1}$	2.82
H_o: $\mu_{EA\_HH22} = \mu_{HH11}$, H_a: $\mu_{EA\_HH22} > \mu_{HH11}$	2.33
H_o: $\mu_{EA\_HH22} = \mu_{HH12}$, H_a: $\mu_{EA\_HH22} > \mu_{HH12}$	4.15

The hypothesis that EA-HH22 performs better than EA-HH1 and EA-HH11 was found to be significant at the 5% level of significance. The hypothesis that EA-H22 produces better results than EA-HH12 is significant at all levels of significance. The time taken by EA-HH1, EA-HH21 and EA-HH22 to evolve heuristic combinations that produce solutions using the minimum number of bins is listed in Table 7. The times are listed in milliseconds.

Table 7. Hyper-heuristic runtimes

Problem	EA-HH1	EA-HH21	EA-HH22
1	1000	2000	4000
2	16	31	31
3	76	1000	1000
4	48000	76000	135000
5	31	31	47
6	218	3000	2000
7	14	90000	119
8	-	118	117
9	16	16	16

10	-	380000	179
11	16	109	187
12	3000	2000	3000
13	-	5539000	235
14	16	16	16
15	16	31	78
16	7000	1000	1000
17	16	125	94
18	9000	41000	42000
19	-	187	141
20	16	344	16
21	16	16	15
22	16	15	15
23	16	16	16
24	16	16	16
25	16	16	16
26	16	16	16
27	32	15	64
28	16	16	16
29	391	1000	500
30	16	16	35
31	31	16	31
32	16	15	16
33	31	16	15
34	16	47	63
35	31	47	31
36	32	16	78
37	1000	31	344
38	39	23000	18000
39	-	8000	5000
40	31	250	3000
41	109	500	547
42	110	328	235
43	141	375	2000
44	93	1000	259
45	16	46	31
46	-	32000	77000
47	-	541000	9
48	3000	21000	297
49	578	20000	16000
50	313	31	42000

As can be anticipated in most cases EA-HH1 takes the least amount of time to execute, followed by EA-HH12 and EA-HH22. EA-HH1 only searches a space of bin selection combinations while EA-HH21 and EA-HH22 explore a space of both bin and item selection heuristics. Thus, it can be expected that EA-HH1 will have lower runtimes. Furthermore, EA-HH22 applies mutation and crossover to both the bin selection and item selection heuristic combinations in each chromosome, while EA-HH21 randomly chooses which combination to apply the genetic operator to, and thus EA-HH22 will have higher runtimes than EA-HH21. However, in some cases EA-HH1 has taken longer to evolve an optimal combination compared to EA-HH21 and EA-HH22, e.g. problem 6. Similarly, the runtime for EA-HH21 has been higher than EA-HH22, e.g. problem 10. This can possibly be attributed to a solution being found earlier during a run as a result of the heuristics contained in the evolved combinations or the use of both bin and item selection heuristics. Alternatively, this may occur as a result of the stochastic nature of evolutionary algorithms, resulting in the evolutionary algorithm following a different path for different random number generator seeds, thus causing a particular run to take longer in reaching an optimal area of the heuristic space.

For each problem, the optimal heuristic combinations evolved on different runs were found to be different. For example for problem 1 the heuristic combinations producing a solution using the minimum number of bins for one run were *fnbwwnbwwwnfwwwwffbwffwfwwwbwfffbfwnww* and s*assalsa ss* while for another run *bnfwnfwbnfwnbbnwwwwwfwffwfbbnw wbwfwfnnwf* and *lalaaalassalaaalalslsassa* were evolved. This can again be attributed to the stochastic nature of evolutionary algorithms. A different random number generator seed is used on each run resulting in the evolutionary algorithm following a different path and evolving different combinations of low-level heuristics. Furthermore, this also illustrates that there is more than one optimal heuristic combination for each problem.

6.3 Performance comparison with other methods

Section 2 provides an overview of the methodologies used to solve the one-dimensional bin-packing problem. This section compares the performance of both the hyper-heuristic approaches to those methods that have been able to generalize well and produce the best results for problems from both the Faulkenauer and the Scholl benchmark sets. The comparison is empirical as these methods vary and even the concept of a run differs from one method to the next. Thus, the comparison is defined in terms of the number of problems (out of the 50 problems) for which the method is able to produce an optimal solution.

The methods producing a solution using the minimum number of bins for all 50 problems include:

- The selection perturbative hyper-heuristic implemented by Bai et al. [2].
- The minimum slack heuristic and variable neigbourhood search used by Flezar et al. [15].
- The weighted annealing approach taken by Loh et al. [16].

Two types of hyper-heuristics, namely, EA-HH1, which searches a space of bin selection heuristic combinations and EA-HH2 which explores a space of bin selection heuristic combinations and a space of item selection heuristic combinations were implemented and evaluated in this study. Three versions of EA-HH1, each using a different heuristic to select an item for placement, were also tested. The version of EA-HH1 using the availability degree heuristic to select an item performed the best of the three versions finding solutions using the minimum number of bins for 46 of the 50 problems. EA-HH2 was able to produce solutions using the minimum number of bins for all 50 of the problems. Thus, a hyper-heuristic searching both the space of bin selection and item selection heuristic combinations appears to be more effective for this domain. Furthermore, this hyper-heuristic has produced results competitive to that of the best performing methods applied to these 50 problems.

7. CONCLUSION

The main aim of the study presented in this paper is to investigate the use of hyper-heuristics in solving the one-dimensional bin-packing problem. Two types of hyper-heuristics, a hyper-heuristic searching the space of bin selection heuristics, and a hyper-heuristic exploring the space of bin selection heuristic combinations and item selection heuristic combinations were implemented. The performance of the hyper-heuristics in solving the one-dimensional bin-packing problem was compared to using each low-level heuristic independently to solve the one-dimensional bin-packing problem. In addition to this the performance of the two types of hyper-heuristics was also compared. The individual heuristics and both types of hyper-heuristics were applied to solving fifty one-dimensional bin-packing problems. This study has revealed that hyper-heuristics produce much better results than each of the low-level heuristics in solving the one-dimensional bin-packing problem.

Furthermore, while previous work has only focused on using bin selection heuristics in solving this problem, this study introduces the use of item selection heuristics as well. It is evident from the study that the use of item selection heuristics are also needed to solve the one-dimensional bin-packing problem and the use of different item selection heuristics produce different results. The hyper-heuristic evolving both bin and item selection heuristic combinations was found to achieve a higher success rate than the hyper-heuristic searching the space of bin selection heuristic combinations only. This study has illustrated the effectiveness of hyper-heuristics and the need for item selection heuristics in solving the one-dimensional bin-packing problem. The study has also revealed that there is more than one optimal heuristic combination for each problem.

Future work will study the evolved heuristic combinations in more detail to identify any substring patterns and relationships between the problem characteristics and the evolved heuristic combinations.

8. ACKNOWLEDGEMENTS

The author would like to thank the reviewers for their helpful comments and suggestions.

REFERENCES

[1] Alvim, A.C.F., Ribeiro, C.C., Glover F. and Aloise, D. J. 2004. A Hybrid Improvement Heuristic for the One-Dimensional Bin Packing Problem, *Journal of Heuristics*, Vol. 10, No. 2, 205-229.

[2] Bai, R., Balzewicz, J., Burke, E.K., Kendall, G. and McCollum, B. 2007. *A Simulated Annealing Hyper-Heuristic Methodology for Flexible Decision Support*. Technical Report No. NOTTCS-TR-2007-8, School of Computing and Information Technology, University of Nottingham.

[3] Bhatia, A.K., Hazra, M., Basu, S. K. 2009. Better-Fit Heuristic for One-Dimensional Bin-Packing Problem. *In Proceedings of the IEEE International Advance Computing Conference (IACC 2009)*. IEEE Press, 193-196.

[4] Burke, E.K., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S. and Vazquez-Rodriques, J.A. 2009. HyFlex: A Flexible Framework for the Design and Analysis of Hyper-Heuristics. In *Blazewicz, J., Drozdowski, M., Kendall, G. and McCollum, B. (eds.): Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Application*s (Dublin, Ireland, August 2009). 790-797.

[5] Burke E. 2003. Hart E., Kendall G., Newall J., Ross P. and Schulenburg S., Hyper-Heuristics: An Emerging Direction in Modern Research. *In Handbook of Metaheuristics*, Chapter 16. Kluwer Academic Publishers, 457– 474.

[6] Burke, E.K., Hyde, M.R. and Kendall, K.2006. Evolving Bin Packing Heuristics with Genetic Programming. In Burke, E.K., Merelo-Guervos, J., Whitely, D., and Yao, X. (Eds.), *Lecture Notes in Computer Science*. 4193, *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)* (Reykjavik, Iceland). Springer, 860-869.

[7] Burke, E.K., Hyde, M.R., Kendall, G. and Woodward, J.R. 2007. Scalability of Evolved On Line Bin Packing Heuristics. In *Proceedings of the Congress of Evolutionary Computation (CEC 2007)* (Singapore). IEEE Press, 2530-2537.

[8] Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. and Woodard, J. 2010. A Classification of Hyper-Heuristic Approaches. In *Handbook of Metaheuristics, International Series in Operations Research and Management Science*, Volume 146, 449-468.

[9] Els R., Pillay N. 2010. An Evolutionary Algorithm Hyper-Heuristic for Producing Feasible Timetables for the Curriculum Based University Course Timetabling Problem. In *Proceedings of the World Congress on Nature and Biologically Inspired Computing (NaBIC 2010)* (Kitakyshu, Japan). IEEE Press, 467- 473.

[10] Falkenauer, E. 1996. A Hybrid Grouping Genetic Algorithm for Bin Packing. *Journal of Heuristics*, Vol. 2, No. 1, June 1996, 5-30.

[11] Fleszar, K. and Hindi, K.S. 2002. New Heuristics for One-Dimensional Bin-Packing. *Computers and Research*, Vol. 29, 821-839.

[12] Jing, X., Zhou, X. and Xu, Y. 2006. A Hybrid Genetic Algorithm for Bin Packing Problem Based on Item Sequencing. *Journal of Information and Computing Science*, Vol. 1, No. 1, 61-64.

[13] Kao, C. Y. and Lin, F.T. 1992. A Stochastic Approach for the One-Dimensional Bin-Packing Problems, In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (Chicago, USA, August 2006). IEEE Press, 1545-1551.

[14] Kasap, N. and Agarwal, A. 2004. Augmented Neural Networks Approach for the Bin-Packing Problem. In *Proceedings of the 4th International Symposium on Intelligent Manufacturing Systems* (September 6-8). 348-358.

[15] Lewis, R. 2009. A General-Purpose Hill-Climbing Method for Order Independent Minimum Grouping Problems: A Case Study in Graph Colouring and Bin Packing. *Computers and Operations Research*, Vol. 36, Issue 7, 2295-2310.

[16] Loh, K.H., Golden, B. and Wasil, E. 2006. Solving the One-Dimensional Bin Packing Problem with a Weight Annealing Heuristic. *Computers and Operations Research*, Vol. 35, 2283-2291.

[17] Pillay, N. 2011. Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem. *Journal of the Operational Research Society*. DOI:10.1057/jors.2011.12.

[18] Ross P. 2005. Hyper-heuristics. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Methodologies*, Burke E.K., Kendall G., Eds, Chapter 17, Kluwer, 529 -556.

[19] Ross, P., Marin-Blaquez, J.G., Schulenburg, S. and Hart, E. 2003. Learning a Procedure that can Solve Hard Bin-Packing Problems: A New GA-Based Approach to Hyper-Heuristics. *Lecture Notes in Computer Science*, Vol. 2724, 1295-1306.

[20] Ross, P., Schulenburg, S., Marin-Blaquez, J.G. and Hart, E. 2002. Hyper-Heuristics: Learning to Combine Simple Heuristics in Bin-Packing Problems. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '02)* (New York, July 2002). 942-948.

[21] Scholl, A, Klein, R and Jurgens, C. 1997. Bison: A Fast Hybrid Procedure for Exactly Solving the One-Dimensional Bin Packing Problem. *Computers and Operations Research*, Vol. 24, No. 7, 626-645.

[22] Ulker, O., Korkmaz, E.E. and Ozcan, E. 2008. A Grouping Genetic Algorithm Using Linear Linkage Encoding. In *Lecture Notes in Computer Science: Parallel Problem Solving from Nature – PPSN X*, Ruldolph et al., Eds, 5199, Springer-Verlag, Heidelberg, 1140-1149.

[23] Valerio de Carvalho, J. M. 1999. Exact Solution of Bin-Packing Problems Using Column Generation and Branch-and-Bound. *Annals of Operations Research*, Vol. 86, 629-659.