# Frequency Distribution based Hyper-Heuristic for the Bin-Packing Problem

He Jiang[1], Shuyan Zhang[2], Jifeng Xuan[3], and Youxi Wu[4]

[1] School of Software, Dalian University of Technology
jianghe@dlut.edu.cn
[2] School of Software Technology, Zhengzhou University
shyzhang@mail.dlut.edu.cn
[3] School of Mathematical Sciences, Dalian University of Technology
xuan@mail.dlut.edu.cn
[4] School of Computer Science and Software, Hebei University of Technology
wuc@scse.hebut.edu.cn *

**Abstract.** In the paper, we investigate the pair frequency of low-level heuristics for the bin packing problem and propose a Frequency Distribution based Hyper-Heuristic (FDHH). FDHH generates the heuristic sequences based on a pair of low-level heuristics rather than an individual low-level heuristic. An existing Simulated Annealing Hyper-Heuristic (SAHH) is employed to form the pair frequencies and is extended to guide the further selection of low-level heuristics. To represent the frequency distribution, a frequency matrix is built to collect the pair frequencies while a reverse-frequency matrix is generated to avoid getting trapped into the local optima. The experimental results on the bin-packing problems show that FDHH can obtain optimal solutions on more instances than the original hyper-heuristic.

**Key words:** hyper-heuristic, frequency distribution, bin-packing, pair frequency

## 1 Introduction

In recent years, hyper-heuristics were proposed to overcome the problem-specific drawbacks of existing heuristics. By definition, hyper-heuristics, termed 'heuristics to choose heuristics' [1], are heuristics utilizing a high-level heuristic to choose and assign a set of simple low-level heuristics(LLHs). The main difference between hyper-heuristics and other heuristics is that hyper-heuristics raise the level of generality [2]. Hyper-heuristics work on a LLH space rather than

* Our work is partially supported by the Natural Science Foundation of China under Grant No. 60805024, 60903049, 61033012, the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 20070141020, CAS Innovation Program under Grant No. ISCAS2009-DR01, and Natural Science Foundation of Dalian under Grant NO. 201000117

directly on the problem space. The goal of a hyper-heuristic is to generate a LLH sequence which can achieve a final solution to the problem at hand.

According to the search characteristics, hyper-heuristics can be classified into two categories, namely constructive hyper-heuristics and perturbative hyper-heuristics [3]. Constructive hyper-heuristics apply LLHs to gradually construct a complete solution from an empty initial solution while perturbative hyper-heuristics improve the solution quality from a complete solution. Both constructive and perturbative hyper-heuristics have been applied to some problems, such as bin-packing [4], [5], timetabling [6], [7], [8], production scheduling [9], and personal scheduling [10]. Among these problems, bin-packing is a typical problem attracting much attention. As an NP-hard problem [11], bin-packing is to pack all the given pieces into as few bins as possible [5].

We focus on the selection strategy for the perturbative hyper-heuristics in this paper. As a significant component, the selection strategy is helpful to decide the next LLH in the perturbative hyper-heuristic [3]. Most of hyper-heuristics select the LLHs individually. However, the combination of LLHs may provide more improvement than individual LLHs. For example, Thabtah & Cowling [12] propose an associative classification approach to predict which LLH to combine with the given LLH sequence.

In this paper, we investigate the frequency distribution for the combination of LLHs and propose a Frequency Distribution based Hyper-Heuristic (FDHH). First, we present the pair frequency of LLHs and employ the distribution of pair frequencies to guide the further selection of LLHs. Then, we design FDHH to solve the bin-packing problem. In FDHH, the frequency distribution is incorporated into an existing algorithm, a Simulated Annealing Hyper-Heuristic (SAHH)[5]. FDHH consists of two phases: one for generating the frequency distribution and the other for guiding the hyper-heuristic. Moreover, a frequency matrix is built to collect the pair frequencies while a reverse-frequency matrix is utilized to avoid getting trapped in the local optimal LLH sequence. Finally, experimental results on the bin-packing problem demonstrate that our FDHH can obtain optimal solutions on more instances than the original hyper-heuristic, SAHH.

The paper is organized as follows. Section 2 gives the related work. Section 3 analyses the frequency distribution and Section 4 describes FDHH. Section 5 reports the experimental results on the bin-packing problem. Finally, conclusion and future work are given in Section 6.

## 2 Related Work

### 2.1 Combination of LLHs

The combination of LLHs is a new technology to enlarge the granularity of the LLHs when selecting heuristics in hyper-heuristics. To our knowledge, the mostly related work is an associative classification based hyper-heuristics for combining an LLH with the existing ones [12]. This algorithm can be viewed as a trade-off

between the greediness degree and the randomness degree for LLHs. Moreover, some other approaches are proposed to explore the characteristics of LLHs. For example, Chakhlevitch & Cowling [13] investigate the learning strategies for choosing the subset of the fittest LLHs for hyper-heuristic design; Ren et al. [14] propose a bipartite-graph based approach to distinguish intensification and diversification sets of LLHs for reducing the search space. In this paper, our approach attempts to analyse the distribution for pair frequencies of LLHs and then to further guide the selection of LLHs.

### 2.2 Bin-Packing Problem

The bin-packing problem is a well-known NP-hard problem in real-world applications [11]. Given a set of pieces $P = \{p_1, p_2, \ldots, p_k\}$, an unlimited set of bins, a weight $a_i$ for the piece $p_i$, and an identical capacity $c$ for each bin, the bin-packing problem is to pack all pieces into bins with the goal to minimize the number of used bins. A solution $x$ to the bin-packing problem is a set of used bins with all pieces packed in the bins. We give the formal definition of the bin-packing problem in Equation (1). In the definition, both $y_j$ and $x_{i,j}$ are binary values. The value 1 of $y_j$ denotes that the $j$th bin is used; otherwise, 0 denotes not. The value 1 of $x_{i,j}$ denotes that the piece $p_i$ is assigned into the bin $j$; otherwise, 0 denotes not. The constraints suggest that the sum weight of pieces in a bin cannot exceed the capacity $c$ and each piece must be packed in one bin. The objective function is the number of the used bins.

$$\min f(x) = \sum_{j=1}^{k} y_j \tag{1}$$

$$\text{s.t. } \sum_{i=1}^{k} a_i x_{i,j} \leq c y_j, \quad j \in \{1, \ldots, k\}$$

$$\sum_{j=1}^{k} x_{i,j} = 1$$

### 2.3 LLHs

In this part, we introduce the LLHs for the bin-packing problem. These LLHs can be combined into a sequence in a hyper-heuristic and each LLH in this sequence is used for enhancing the solution quality or to get the solution out of local optima. We choose 6 simple LLHs from the LLH list in competition CHeSC [15] for the bin-packing problem. Due to the paper length limit, we briefly list their functions as follows. $h_1$, to *swap from lowest bin*; $h_2$, to *split a bin*; $h_3$, to *swap*; $h_4$, to *repack the lowest filled bin*; $h_5$, to *destroy 3 highest bins*; $h_6$, to *destroy 3 lowest bins.*

### 2.4 Simulated Annealing Hyper-Heuristic

The Simulated Annealing Hyper-Heuristic (SAHH) proposed by Bai et al. [5] is a typical hyper-heuristic for the bin-packing problem. We briefly introduce this algorithm as follows.

**Table 1.** Framework of SAHH algorithm

| Algorithm: SAHH<br>Input: $pro, H, W, t, total\_iter, LP, \beta$<br>Output: the LLH sequence. |
|---|
| (1) $s$ = Generate_Initial_Solution($pro, H$);<br>(2) while $current\_iter < total\_iter$ do<br>(2.1) $h_i$ = Stochastic_Heuristic_Selection($H, W$);<br>(2.2) $s'$ = Heuristic_Application($s, h_i$);<br>(2.3) $s$ = Simulated_Annealing_Acceptance ($s', s, t$);<br>(2.4) $t$ = Temperature_Resetting($t, \beta$);<br>(2.5) // short term learning<br>If mod($current\_iter, LP$) = 0 then<br>(2.5.1) $C$ = Performance_Calculation($LP$);<br>(2.5.2) $W$ = Weight_Resetting($C$);<br>Endwhile |

Besides the similar framework of other hyper-heuristics, SAHH adopts three special strategies: stochastic heuristic selection, simulated annealing acceptance, and short term learning. The details are shown in Table 1. Given a test instance *pro* and a set of perturbative LLHs $H = \{h_1, h_2, \ldots, h_n\}$, SAHH generates an initial solution $s$ and retains $s$ as the current solution (step (1)). Thereafter, an iteration consisting of five steps, namely selection, application, acceptance, resetting, and learning begins to iteratively update the current solution (step (2)). In step (2.1), stochastic heuristic selection strategy selects a heuristic $h_i$ according to its weight $w_i$ ($w_i \in W$). Then SAHH applies $h_i$ to the current solution $s$ and generates a new solution $s'$ (step (2.2)). In step (2.3), the simulated annealing acceptance decides whether $s'$ is accepted as the current solution depending on the current temperature $t$. Next, step (2.4) resets the temperature. In this step, SAHH checks whether the current temperature $t$ should be decreased, increased or unchanged. The current temperature is changed according to $t = t/(1 + \beta t)$ when decreasing, or $t = t/(1 - \beta t)$ when increasing. Since the performance of the LLHs varies during different periods, SAHH introduces short term learning with $LP$ as the length of one learning period (step (2.5)). In this step, SAHH calculates a performance $c_i$ of each LLHs $h_i$ and resets the weights $w_i$ ($c_i \in C$) based on $LP$ iterations. Finally, SAHH returns a LLH sequence to generate the final solution to the problem.

## 3 Frequency Distribution

In this section, we introduce the notion of pair frequency and analyse the frequency distribution using the LLH sequences of SAHH for the bin-packing problem.

### 3.1 Pair Frequency and Frequency Matrix

Given a set of LLHs $H = \{h_1, h_2, \ldots, h_n\}$ and the fixed length $m$ of sequences, the size of heuristic space is $n^m$. Then it is intractable to obtain the optimal sequence in polynomial time for large $n$ and $m$. However, we can obtain approximate optimal sequences instead. From the view of the graph theory, each sequence in the heuristic space is acquired by traversing a fully connected graph [14] whose vertexes are $n$ LLHs. When a hyper-heuristic traverses the graph, the information guiding the search from one heuristic to another is significant. We employ the pairs of LLHs (the edges in a graph) to help to express the association by their frequencies. In this paper, $h_{i,j}$ denotes the pair that starts with $h_i$ and ends with $h_j$ $(i, j \in \{1, \ldots, n\})$. Therefore, instead of analysing heuristics individually, we tend to analyse the pairs of LLHs.

We define a frequency matrix $F$ for a given heuristic sequence. In $F$, an element $F_{i,j}$ denotes the pair frequency of $h_{i,j}$. Considering a set of $n$ LLHs, $H = \{h_1, h_2, \ldots, h_n\}$, there are totally $n \times n$ different pairs of LLHs, i.e., $h_{1,1}, \ldots, h_{1,n}, h_{2,1}, \ldots, h_{2,n}, \ldots, h_{n,1}, \ldots, h_{n,n}$. Then, the size of the frequency matrix $F$ is $n \times n$. Let $L = (l_1, l_2, \ldots, l_m)$ be the sequence obtained by SAHH and $m$ is the length of $L$. It can intuitively conclude that a set of pairs defined as $LC = \{l_{1,2}, l_{2,3}, \ldots, l_{m-1,m}\}$ lies in $L$ where $l_{i,i+1}$ is the pair combined by the $i$th and $(i+1)$th LLHs in $L$. Here, $|LC| = m - 1$. We let $t_{i,j}$ be the number of occurrences of $h_{i,j}$ in $LC$. According to this knowledge, the element $F_{i,j}$ is formally defined as

$$F_{i,j} = t_{i,j}/|LC| \qquad (i, j \in \{1, \ldots, n\}) \tag{2}$$

### 3.2 Frequency Distribution Analysis

In this section, we visualise the frequency distributions based on the pair frequencies. According to Section 3.1, the values of pair frequencies require the knowledge of existing sequences to build the frequency matrix. For a given instance, SAHH is run for ten rounds independently and ten distinct sequences are obtained. These sequences are thereafter used to produce frequency matrixes respectively. Thus, we illustrate the frequency distributions from these ten matrixes in one figure. We set all the parameters of SAHH according to [5] except choosing 6 LLHs in Section 2.3.

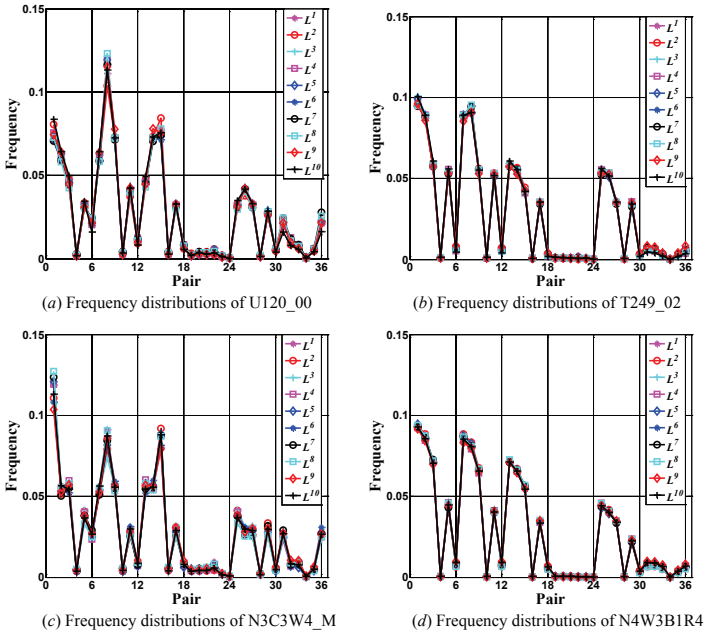
The instances in our experiments are from three widely used classes of bin-packing problem instances: Uniform (80 instances), Triplet (80 instances), and Sch_set (1210 instances) which have been shown and investigated in [5], [16] and [17]. The detailed descriptions of these classes of instances are described in Section 5. Due to the paper length limit, we only illustrate the frequency distributions for four selected instances: U120_00, T249_02, N3C3W4_M, and N4W3B1R4. Similar tendencies can be found for the other instances. The selected instances are described in Table 2.

Fig. 1 illustrates the frequency distributions of U120_00, T249_02, N3C3W4_M, and N4W3B1R4. Each sub-figure plots ten frequency distributions from ten different sequences $\{L^1, L^2, \ldots, L^{10}\}$ of the corresponding instance. The horizontal

**Table 2.** Characteristics of four selected bin-packing problem instances

| Instance | Instance class | Piece number | Weight range | Capacity |
|---|---|---|---|---|
| U120_00 | Uniform | 120 | [20, 100] | 150 |
| T249_02 | Triplet | 249 | [250, 500] | 1000 |
| N3C3W4_M | Sch_set | 200 | [30, 100] | 150 |
| N4W3B1R4 | Sch_set | 500 | [114, 168] | 1000 |

axis shows the pairs (the index in the horizontal axis is set as$(i-1)\times 6+j$ for $h_{i,j}$, e.g., Index 9 is for $h_{2,3}$, and Index 25 is for $h_{5,1}$) while the vertical axis indicates the pair frequencies.




(*a*) Frequency distributions of U120_00

(*b*) Frequency distributions of T249_02

(*c*) Frequency distributions of N3C3W4_M

(*d*) Frequency distributions of N4W3B1R4

**Fig. 1.** Illustrations of frequency distributions

From Fig. 1, we make four observations.First, for one instance, ten frequency distributions generated by ten distinct LLH sequences are quite similar, especially for T249_02 and N4W3B1R4. Second, for one instance, different pairs have different frequency values. Third, for different instances, the frequency distributions are somewhat similar but not identical across the four instances; note that all the sub-figures show that the average frequencies of pairs started with $h_1, h_2, h_3$, and $h_5$ are larger than that those started with $h_4$ and $h_6$, yet frequen-

cies of the same pair (e.g., $h_{2,1}$) are different for the four instances. Fourth, the pairs of LLHs can be classified into two categories according to the values of frequencies: pairs with large frequencies and pairs with small frequencies, e.g., the frequency 0.05 can be viewed as one boundary among the frequencies. According to sequences generated by SAHH, large pair frequencies for the combination of the two LLHs are likely to improve the solution quality while the remaining pairs tend to decrease the solution quality.

## 4 Frequency Distribution Based Hyper-Heuristic

Based on the frequency matrixes in Section 3, we propose a Frequency Distribution based Hyper-Heuristic (FDHH). In FDHH, an existing hyper-heuristic, SAHH is used to generate the pair frequencies and then SAHH is extended into a Frequency based Simulated Annealing Hyper-Heuristic (FSAHH) to guide the LLHs. In FSAHH, a reverse-frequency matrix is proposed to promote the search process. In this section, we first give a reverse version of the frequency matrix. Then, we propose FSAHH based on the frequency matrix and the reverse-frequency matrix. Finally, we present the framework FDHH, which employs both SAHH and FSAHH as sub-algorithms.

### 4.1 Reverse-Frequency Matrix

After choosing one heuristic $h_{last}$, the next LLH $h_i$ is selected according to the pair frequencies $h_{last,1}, h_{last,2}, \ldots, h_{last,n}$. If the frequencies are in the frequency matrix $F$, as shown in Section 3, pairs with larger frequency values are more likely to be chosen and this leads to get trapped in local optima.

As a result, in order to get out of local optima, sometimes, a reverse-frequency matrix should be utilized to increase the small pair frequencies and to decrease the large pair frequencies. For the reverse-frequency matrix $R$, its element $R_{i,j}$ is the reverse-frequency of pair $h_{i,j}$ $(i, j \in \{1, \ldots, n\})$. Each row $R_i$ is created by exchanging the $q$th largest element with the $q$th smallest element in $F_i$, $q \in \{1, \ldots, n/2\}$. $R_i$ and $F_i$ are reverse-frequencies and frequencies of pairs started with the LLH $h_i$. For instance, with $n = 6$, in the first row of $F$, if $F_{1,1}$ has the largest frequency value, and $F_{1,4}$ has the smallest frequency value. Then set $R_{1,1}$ to $F_{1,4}$ and $R_{1,4}$ to $F_{1,1}$. If $F_{1,2}$ has the second largest frequency value, and $F_{1,6}$ has the second smallest frequency value, we set $R_{1,2}$ to $F_{1,6}$ and $R_{1,6}$ to $F_{1,2}$. At last, if $F_{1,3}$ has the third largest frequency value, and $F_{1,5}$ has the third smallest frequency value, we set $R_{1,3}$ to $F_{1,5}$ and $R_{1,5}$ to $F_{1,3}$. Therefore, the first row $R_1$ is generated. The other rows of $R$ can be generated under the same method.

### 4.2 Frequency Based Simulated Annealing Hyper-Heuristic

Based on the frequency matrix and the reverse-frequency matrix, we propose FSAHH. FSAHH employs three new strategies: *Frequency_Based_Selection*, *Count_Value_Updating*, and *Interval_Disturb*. We show their details as follows.

**Table 3.** Framework of FSAHH algorithm

| Algorithm: FSAHH<br>Input: $pro, H, t, total\_iter, LP, \beta, maxV, F, R$<br>Output: the best solution |
|---|
| (1) initialize $v = 0$ and $ts = t$;<br>(2) $s$ = Generate_Initial_Solution($pro, H$);<br>(3) while *current_iter* < *total_iter* do<br>(3.1) $h_i$ = Frequency_Based_Selection $(H, F)$;<br>(3.2) $s'$ = Heuristic_Application$(s, h_i)$;<br>(3.3) $v$ = Count_Value_Updating $(s', s)$;<br>(3.4) $s$ = Simulated_Annealing_Acceptance $(s', s, t)$;<br>(3.5) $t$ = Temperature_Resetting$(t, \beta)$;<br>(3.6) if $v$ equals to $maxV$ then<br>$s$ = Interval_Disturb$(s, R, ts)$;<br>*current_iter* = *current_iter* + $LP$;<br>Endwhile |

*Frequency_Based_Selection*: Given a set of LLHs $H = \{h_1, h_2, \ldots, h_n\}$ and a frequency matrix $F$ (or a reverse-frequency matrix $R$), this strategy selects a LLH $h_i$ under matrix $F$ (or $R$). To start the procedure, given the last used LLH $h_{last}$, the first $h_i$ is chosen randomly; the other $h_i$ is selected under the frequency $F_{last,i}$ (or $R_{last,i}$).

*Count_Value_Updating*: Given a counter $v$ which records the number of continuously unimproved solutions, this strategy updates $v$ by comparing objective functions of the current solution $s$ and the new solution $s'$. There are four cases. In case 1, $f(s') < f(s)$, then set $v = 0$; in case 2, $f(s') > f(s)$, then set $v = v+1$; in case 3, $f(s') = f(s)$ and $s'$ is different from $s$, then set $v = 0$; in case 4, $f(s') = f(s)$ and $s' = s$, then set $v = v + 1$.

*Interval_Disturb*: This strategy is a standard simulated annealing method using a reverse-frequency matrix $R$ and a high starting temperature $t$ to escape from the local optima. Given a current solution $s$, matrix $R$, and temperature $t$, Interval_Disturb consists of 4 steps. Step (1) selects a heuristic $h_i$ by Frequency_Based_Selection strategy with $R$ as the based matrix. Step (2) applies $h_i$ to $s$ and generates a new solution $s'$. Step (3), with $t$ as the current temperature, accepts $s'$ as the current solution by Simulated_Annealing_Acceptance strategy (the same with step (2.3) in Table 1). Step (4) decreases the temperature $t$ (sets $t = t/(1 + \beta t)$). Interval_Disturb totally repeats step (1)-(4) $LP$ times and returns the best solution. $\beta$ and $LP$ are the same as those in Table 1.

We present the details of FSAHH in Table 3. It works as follows. After generating an initial solution $s$ with the same manner used in SAHH, FSAHH runs 6 steps iteratively (step (3)). In step (3.1), we utilize Frequency_Based_Selection strategy and a frequency matrix $F$ of a given test instance to select a LLH $h_i$. Then, we apply $h_i$ to the current solution $s$ to generate a new solution $s'$ in step (3.2). Next in step (3.3), the counter $v$ is updated by Count_Value_Updating

method with $s$ and $s'$ as input parameters. After that, in step (3.4) and (3.5), we adopt Simulated_Annealing_Acceptance to decide whether $s'$ should be accepted as the current solution, and Temperature_Resetting method to reset the current temperature $t$. Detailed explanations are corresponding to step (2.3) and (2.4) of Table 1, respectively. The last step is trying to escape from the local optima (step (3.6)). When $v = maxV$ ($maxV$ is set to $LP/5$ in the experiments), it can be viewed as the algorithm has already been trapped in the local optima. Then Interval_Disturb is triggered with the current solution $s$, the reverse-frequency matrix $R$, and the starting temperature $ts$ as input parameters. Note that, in Interval_Disturb, there is a loop which executes $LP$ times. Thus, after running Interval_Disturb, add $LP$ to *current_iter*. The iteration runs until *current_iter* equals to *total_iter*. At last, the best solution is returned.

### 4.3 Framework

In this section, we present the framework of FDHH, which employs SAHH and FSAHH as subroutine algorithms. We present the framework of FDHH in Table 4. For a given test instance, first, SAHH is run *learn_iter* times (*learn_iter* is set to 10 in the experiments) to achieve a set of distinct LLH sequences denoted as $LS = \{L^1, L^2, \ldots, L^{learn\_iter}\}$ which are used for learning in next steps. Then, in step (2), the average frequency matrix *aveF* is generated utilizing the sequence set *LS*. After that, in step (3), the average reverse-frequency matrix *aveR* is transferred from *aveF*. As preconditions *aveF* and *aveR* have been established, then in step (4), *FSAHH* is run *learn_iter* times with *aveF* and *aveR* as two parameters. Finally, the best solution is returned in step (4).

**Table 4.** Framework of FDHH algorithm

| Algorithm: FDHH<br>Input: $pro, H, W, t, total\_iter, LP, \beta, maxV, learn\_iter$<br>Output: the best solution $s^*$ |
|---|
| initialise $s^*$ as a random solution<br>(1) for $i = 1$ to *learn_iter*<br>$L^i$ = SAHH($pro, H, W, t, total\_iter, LP, \beta$);<br>End for<br>(2) $aveF$ = Getting_Average_Frequency_Matrix ($LS$);<br>(3) $aveR$ = Getting_Average_Reverse_Frequency_Matrix($aveF$);<br>(4) for $i = 1$ to *learn_iter*<br>$s_i$ = FSAHH($pro, H, t, total\_iter, LP, \beta, maxV, aveF, aveR$);<br>*if* ($f(s_i) < f(s^*)$) then<br>$s^* = s_i$;<br>End for |

## 5 Experimental Results

In this section, we apply our FDHH to the bin-packing problem. To evaluate our algorithm, experiments are conduct on three classes of bin-packing instances (totally 1370 instances).

The first class (Uniform class) has 80 instances totally with $c = 150$, and $a_i \in [20, 100]$. There are 4 sub classes in Uniform class: Fal_U120, Fal_U250, Fal_U500, and Fal_U1000, each of which has 20 test instances with $k = 120$, 250, 500, and 1000, respectively.

The second class (Triplet class) also has 80 instances with $c = 1000$, and $a_i \in [250, 500]$. There are 4 sub classes in Triplet class: Fal_T60, Fal_T120, Fal_T249, and Fal_T501, each of which has 20 test instances with $k = 60$, 120, 249, and 501, respectively. In a more complex class, Triplet, each bin of the optimal solution must be fully filled with 3 pieces.

The third class (Sch_set) contains 1210 instances with 3 sub classes: Sch_set1 (720 instances), Sch_set2 (480 instances), and Sch_set3(10 instances). In Sch_set1, $c \in \{100, 120, 150\}$, and $a_i$ distributes in [1, 100], [20, 100], or [30, 100]. In Sch_set2, $c = 1000$, and $a_i$ satisfies that three to nine pieces pack in one bin. For both Sch_set1 and Sch_set2, $k \in$\{50, 100, 200, 500\}. Sch_set3 is harder than Sch_set1 and Sch_set2 with $c = 100000$, $a_i \in$[20000, 35000] and $k = 200$.

Uniform class and Triplet class are introduced by [18] and Sch_set class is generated by [19]. For all instances in the Uniform and Triplet classes, the optimal solution is known [20]. Their instances and corresponding optimal objective values can be downloaded (http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/binpackinfo.html). In Sch_set, 1184 instances have been solved optimally. The instances and optimal objective values (or the best known lower bound) can be downloaded (http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm).

Experiments are performed under Win XP on a Pentium Dual Core 2.8 GHx with 4G memory PC. All the source codes are implemented in Java, compiled using JDK 6.20. We run FDHH on the 1370 test instances, and for comparison, we run SAHH on the same instances independently. The source code of SAHH is implemented according to [5]. All parameters of SAHH are extracted from [5] except the set of LLHs.

Table 5 presents the comparative results of our proposed FDHH (running one time with $learn\_iter = 10$) and SAHH (running $2 \times learn\_iter$ times). 'Instance class' column shows the names of test instance classes. 'Num' indicates the number of test instances in each sub class. 'Hits' denotes the number of instances that can arrive at the optimal objective value (or the best known lower bound). 'Max dev.' is the maximum absolute deviation from the objective values in the worst case to the optima over all instances in a class. From 'Hits' columns of Table 5, we can see that FDHH achieves 1094 optimal solutions (out of 1370) while SAHH gets 1085 optimal solutions. Besides, as shown in 'Max dev.' columns, it is easy to summarize that the worst solutions generated by FDHH is closer to optima than that generated by SAHH. Note that the numerical results of SAHH are different from those reported in [5] due to the difference between the chosen LLHs.

**Table 5.** Comparative results of SAHH and FDHH

| Instance class | | Num | SAHH | | FDHH | |
|---|---|---|---|---|---|---|
| | | | Hits | Max dev. | Hits | Max dev. |
| Uniform | Fal_U120 | 20 | 17 | 1 | 18 | 1 |
| | Fal_U250 | 20 | 20 | 0 | 20 | 0 |
| | Fal_U500 | 20 | 20 | 0 | 20 | 0 |
| | Fal_U1000 | 20 | 16 | 3 | 17 | 2 |
| Triplet | Fal_T60 | 20 | 0 | 1 | 0 | 1 |
| | Fal_T120 | 20 | 0 | 1 | 0 | 1 |
| | Fal_T249 | 20 | 0 | 3 | 0 | 1 |
| | Fal_T501 | 20 | 0 | 3 | 0 | 2 |
| Sch_Set | Sch_Set1 | 720 | 669 | 2 | 672 | 1 |
| | Sch_Set2 | 480 | 340 | 2 | 343 | 1 |
| | Sch_Set3 | 10 | 3 | 2 | 4 | 1 |
| All | - | 1370 | 1085 | 3 | 1094 | 2 |

## 6 Conclusion and Future Work

In this paper, we study the frequency distributions of LLH sequences and design the frequency distribution based hyper-heuristic (FDHH). To build the frequency distribution for the algorithm design, we propose the notation of pair frequency to investigate the characteristics of the combination of LLHs. Experimental results on the bin-packing problem indicate that FDHH can obtain optimal solutions on more instances than the original hyper-heuristic, SAHH. The experience on the frequency distribution in this paper can be drawn on to design other hyper-heuristics.

In future work, we plan to design a selection strategy based on the combination of multiple LLHs. On the other hand, the multiple granularity combination may be more effective than the single granularity. Since the multiple granularity of combination of LLHs enlarges the complexity of our algorithm, it is necessary to design a new strategy to dynamically decide the size of the combination of the LLHs. Moreover, it is useful to give a large empirical study or develop the theoretical analysis for providing a relatively exact approach for deciding the size of the combination.

## References

1. Burke, E.K., Hart, E., Kendall,G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An Emerging Direction in Modern Search Technology. In: Glover, F., Kochenberger, G. (eds) Handbook of Metaheuristics, pp. 457–474. Kluwer (2003)
2. Ochoa, G., Vaquez-Rodríguez, J.A., Petrovic, S., Burke, E.K.: Dispatching Rules for Production Scheduling: a Hyper-heuristic Landscape Analysis. In: Proceedings of the IEEE CEC, pp. 1873–1880. Trondheim, Norway (2009)

3. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: A Survey of Hyper-heuristics. Technical Report, School of Computer Science and Information Technology, University of Nottingham, Computer Science (2009)
4. Ross, P., Marin-Blazquez, J.G., Schulenburg, S., Hart, E.: Learning a Procedure that Can Solve Hard Bin-packing Problems: A new GA-based Approach to Hyper-heuristics. In: Proceedings of the GECCO, pp. 1295–1306. Springer, Berlin (2003)
5. Bai, R., Blazewicz, J., Burke, E.K., Kendall, G., McCollum, B.: A Simulated Annealing Hyper-heuristic Methodology for Flexible Decision Support. Technical report, School of CSiT, University of Nottingham (2007)
6. Qu, R., Burke, E.K.: Hybridisations within a Graph Based Hyper-heuristic Framework for University Timetabling Problems. JORS. 60, 1273–1285 (2008)
7. Qu, R., Burke, E.K., McCollum, B.: Adaptive Automated Construction of Hybrid Heuristics for Exam Timetabling and Graph Colouring Problems. EJOR. 198, 392–404 (2008)
8. Bilgin, B., Ozcan, E., Korkmaz, E.E.: An Experimental Study on Hyper-heuristics and Final Exam Scheduling. In: Proceedings of the International Conference on the Practice and Theory of Automated Timetabling, pp. 123-140. Springer, Berlin (2007)
9. Vazquez-Rodriguez, J.A., Petrovic, S., Salhi, A.: A Combined Meta-heuristic with Hyper-heuristic Approach to the Scheduling of the Hybrid Flow Shop with Sequence Dependent Setup Times and Uniform Machines. In: Proceedings of the 3rd Multidisciplinary International Scheduling Conference, pp. 506–513. Paris, France (2007)
10. Han, L., Kendall, G.: Guided Operators for a Hyper-heuristic Genetic Algorithm. In: Proceedings of AI-2003: Advances in Artificial Intelligence, pp.807-820. Perth, Australia (2003)
11. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons (1990)
12. Thabtah, F. Cowling, P.: Mining the Data from a Hyperheuristic Approach Using Associative Classification. Expert Systems with Applications. 34 (2), pp. 1093–1101 (2008)
13. Chakhlevitch, K., Cowling, P.: Choosing the Fittest Subset of Low Level Heuristics in a Hyperheuristic Framework. In: Proceedings of 5th European Conference on EvoCop, pp. 25-33. Springer, Lausanne, Switzerland (2005)
14. Ren, Z., Jiang, H., Xuan, J., Luo, Z.: Ant Based Hyper Heuristics with Space Reduction: A Case Study of the p-Median Problem. In: 11th International Conference on PPSN, pp.546–555. Springer, Krakow, Poland (2010)
15. Cross-domain Heuristic Search Challenge, http://www.asap.cs.nott.ac.uk/chesc2011 /index. html
16. Fleszar, K., Hindi, K.S.: New Heuristics for One-dimensional Bin-packing. Computers and Operations Research. 29(7), 821-839 (2002)
17. Alvim, A. C. F., Ribeiro, C.C., Glover, F., Aloise, D.J.: A Hybrid Improvement Heuristic for the One Dimensional Bin Packing Problem. Journal of Heuristics. 10, 205-229 (2004)
18. Falkenauer, E.: A Hybrid Grouping Genetic Algorithm for Bin Packing. Journal of Heuristics. 2, 5-30 (1996)
19. Scholl, A., Klein, R., Jurgens, C.: BISON: A Fast Hybrid Procedure for Exactly Solving the One Dimensional Bin Packing Problem. Computers & Operations Research. 24(7), 627-645 (1997)
20. Valerio de Carvalho, J.M.: Exact Solution of Bin-packing Problems Using Column Generation and branch-and-bound. Annals of Operations Research. 86, 629-659 (1999)