Knowledge-Based Systems 205 (2020) 106264

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

# Hyper-heuristic local search for combinatorial optimisation problems

Ayad Turky [a,*], Nasser R. Sabar [b], Simon Dunstall [c], Andy Song [a]

[a] Computer Science and Information Technology, RMIT University, Melbourne, VIC 3000, Australia
[b] Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3086, Australia
[c] CSIRO Data 61, Docklands, Australia

ARTICLE INFO

*Article history:*
Received 9 November 2019
Received in revised form 7 June 2020
Accepted 13 July 2020
Available online 24 July 2020

*Keywords:*
Hyper-heuristic
Local search algorithm
Google machine reassignment
Multi-capacity bin packing

ABSTRACT

Local search algorithms have been successfully used for many combinatorial optimisation problems. The choice of the most suitable local search algorithm is, however, a challenging task as their performance is highly dependent on the problem characteristic. In addition, most of these algorithms require users to select appropriate internal neighbourhood structures to obtain desirable performance. No single local search algorithm can consistently perform well with a fixed setting, for different types of problems or even different instances of the same problem. To address this issue, we propose a hyper-heuristic framework which incorporates multiple local search algorithms and a pool of neighbourhood structures. This framework is novel in three respects. Firstly, a two-stage hyper-heuristic structure is designed to control the selection of a local search algorithm and its internal operators. Secondly, we propose an adaptive ranking mechanism to choose the most appropriate neighbourhood structures for the current local search algorithm. The proposed mechanism uses the entropy to evaluates the contribution of the local search in terms of quality and diversity. It adaptively adjusts the pool of candidate neighbourhood structures. Thirdly, we use a population of solutions within the proposed framework to effectively navigate different areas in the solutions search space and share solutions with local search algorithms. To ensure different solutions is allocated in different regions of the search space, we propose a distance-based strategy for population updating process that allowing solutions to share local search algorithms. We have evaluated the performance of the proposed framework using two challenging optimisation problems: Multi-Capacity Bin Packing benchmark instances and Google Machine Reassignment benchmark instances. The results show the effectiveness of the proposed framework, which outperformed state-of-the-art algorithms on several problem instances.

## 1. Introduction

Combinatorial Optimisation Problems (COPs) arise in many real-world applications such as resource allocation, scheduling, routing, production planning and economic systems [1]. COPs can be defined as the process of searching for the best or optimal solutions that can minimise the cost or maximise the objective. COPs typically have a huge and often heavily constrained search space. That makes them challenging and costly to solve. Many COPs are known as NP-hard problems, hence impractical for exact methods. Instead, meta-heuristic algorithms have been used to deal with the COPs because they are able to find reasonably good solutions within an acceptable timeframe [2–9].

Local search (LS) algorithms are a type of meta-heuristic which have been shown to be very effective on a range of COPs [10–14]. LS uses various internal operators to explore the neighbouring areas around current solutions to find better quality ones. Examples of well-known LS algorithms are Hill Climbing, Simulated Annealing, Tabu Search, Iterated Local Search and Great Deluge [1]. The key difference between them is how the LS navigates the solution search spaces.

Empirical and theoretical studies show that the success of LS strongly depends on the type of problem and the internal operators of the search. To this end, several LS internal operators have been introduced in the literature. However, the decision of which LS should be used with what operators are very challenging and require a time-consuming trial-and-error process. A human expert is often used to manually customise the selected LS, including the search strategy and the operators, to better fit with a given problem. Although setting the right combination is crucial, customisation implies that when encountering a new problem or a new instance of the known problem, the optimal setting is likely to be different and requires further adjustment to find it. This is also consistent with the No Free Lunch Theorem [15] which proves there is no single LS algorithm with the

* Corresponding author.
*E-mail addresses:* ayad.turky@rmit.edu.au (A. Turky), n.sabar@latrobe.edu.au (N.R. Sabar), Simon.Dunstall@data61.csiro.au (S. Dunstall), andy.song@rmit.edu.au (A. Song).

https://doi.org/10.1016/j.knosys.2020.106264
0950-7051/© 2020 Elsevier B.V. All rights reserved.

same operators that would be able to perform the best on all problems.

Therefore, an adaptive method, which can select the algorithm and its operators automatically, would be highly beneficial. To this end, we propose a hyper-heuristic framework to address this challenge. The proposed framework involves multiple LS algorithms and a pool of operators. It performs the selection of the LS algorithm and operators dynamically during the process of solving a problem. The appropriate mix of the LS algorithm with the operators is determined adaptively during the search process. The proposed framework has the following three features that distinguish it from existing methods. Firstly, the proposed framework has a two-stage hyper-heuristic structure to automatically select the LS algorithm and its internal operators based on the current search status. Secondly, we propose an adaptive quality and diversity ranking mechanism to choose the most appropriate neighbourhood structures. Thirdly, we use a population of solutions to navigate vast areas of the search space. A distance-based updating strategy is designed accordingly, which allows solutions to be placed in different regions of the search space and enable solutions to be shared among different local search algorithms.

The main objectives of this study are:

- To establish an online two-stage hyper-heuristic local search framework that can automatically select the most appropriate local search algorithm and its internal operators.
- To propose an adaptive ranking mechanism to rank local search algorithm and adaptively adjust the pool of operators. So good candidate algorithms can be selected more effectively.
- To incorporate population of solutions in the search to improve the adaptiveness and effectiveness of the search. An updating strategy is to be proposed to diversify the search.

The following are the main contributions of this work:

1. An adaptive two-stage hyper-heuristic local search framework to effectively solve different optimisation problems.
2. An adaptive ranking selection mechanism to control the selection of local search algorithms and their internal operators.
3. A distance-based strategy for population updating to guide the search to promising areas in the solution space.
4. A population-based local search framework to deal with various problem sizes and complexities.
5. Extensive computational experiments to evaluate the performance of the proposed hyper-heuristic framework and to assess the impact of the proposed improvements. Two well-known challenging combinatorial optimisation problems are involved: Multi-Capacity Bin Packing Problems and Google Machine Reassignment Problems. The results show the excellent performance of the proposed framework in comparison with state-of-the-art algorithms on both problems.

The rest of the paper is organised as follows. Section 2 gives the formal definition of both problems. Section 3 explains the proposed framework. Section 4 shows the experiment settings and the results. Finally, the conclusions and final remarks are presented in Section 5.

## 2. Problem description

Resource allocation is of great importance in industry and business. The aim is to find the best possible assignment to optimise the desired objective. Resource allocation problems often are combinatorial optimisation problems and inherently NP-hard. In this work, we use two complex resource allocation problems, namely Multi-Capacity Bin Packing Problem (MCBPP) and Google Machine Reassignment Problem (GMRP), to evaluate the performance of the proposed hyper-heuristic framework. In the following subsections, we present the definitions and mathematical models of both problems (MCBPP and GMRP).

### 2.1. Multi-Capacity Bin Packing Problem (MCBPP)

Bin packing is one of the most studied problems in the family of NP-hard problems. It consists of packing a set of items into a minimum number of bins. MCBPP is a generalisation of the 1D bin packing problem in which each object $i \in I$ (with n=$|I|$) has two attributes, a *weight* $w_i$ and a *length* $l_i$ (with $w_i \geq 0$, $l_i \geq 0$ and $w_i + l_i > 0$), and bins have weight and length capacities, denoted as $W$ and $L$, respectively. This problem has a wide range of applications in the real-world. For instance, apart from the tradition application in loading and scheduling [16], an important application of MCBPP is in the emerging field of cloud computing. Cloud computing attempts to manage computing resources on cloud computing platforms [17]. MCBPP can be seen as an extension for cloud computing packing processes [18].

In general, the objective of the MCBPP is to pack the set of objects $n$ in as few bins as possible, without exceeding the capacity of bins for any resource [19]. The problem formulation is as follows:

$$\min\ z = \sum_{j \in J} y_j \tag{1}$$

$$\sum_{j \in J} x_{ij} = 1, \qquad i \in I \tag{2}$$

$$\sum_{i \in I} w_i x_{ij} \leq W y_j, \qquad j \in J \tag{3}$$

$$\sum_{i \in I} l_i x_{ij} \leq L y_j, \qquad j \in J \tag{4}$$

$$x_{ij} \in \{0, 1\}, y_j \in \{0, 1\} \qquad i \in I, j \in J \tag{5}$$

where $J$ represents the number of bins indexed by $j$; $y_j = 1$ if and only if the bin is used; $x_{ij} = 1$ if the item $i$ is packed into the bin $j$. Formula (2) is a constraint which ensures all the items being assigned to a bin. Constraints in Formulae (3) and (4) ensure that such an assignment does not exceed the bin capacities. Formula (1) is the objective function which is to minimise the number of bins used. Several algorithms have been proposed for MCBPP in the literature, including branch-and-bound [16], heuristic and genetic algorithm [20], heuristics inspired by first-fit decreasing algorithm [21], approximation algorithms [18]. However, despite the success of these algorithms, no single algorithm managed to obtain the best results overall tested instances. The branch-and-bound algorithms can get the optimal values but only suitable for small-sized instances. The first-fit decreasing algorithm is specifically designed to generate the initial solutions only. The genetic algorithms need a human expert to select the best evolutionary operators.

### 2.2. Google Machine Reassignment Problem (GMRP)

GMRP is a recent combinatorial optimisation problem proposed in 2012 at ROADEF/EURO challenge [22]. GMRP involves a set of machines and a set of processes. Each machine has a set of resources (e.g. CPU, RAM, etc.). Initially, each process is allocated to a machine. A process can be moved to another machine to improve the usage of the machine. GMRP contains two kinds of constraints: hard and soft constraints. Moving a process from one machine to another is considered feasible if all hard constraints are satisfied. The soft constraints can be violated, but violations should be minimised as much as possible. The goal of solving GMRP is to search for a feasible solution that has a minimum total weighted violation of soft constraints.

In this work, the description of GMRP is adopted from [22] and [23]. Let M= $\{m1, m2, \dots, m_k\}$ be the set of machines and P=$\{p1, p2, \dots, p_w\}$ be the set of processes. The solution can represented by a vector *Map* with length equal to $w$, where *Map(p)* corresponds to the machine assigned to process *p*. The allocation of processes must satisfy following hard constraints:

- *Capacity constraints*: the sum of requirements of resource of all processes does not exceed the capacity of the allocated machine.
- *Conflict constraints*: processes of the same service must be allocated into different machines.
- *Transient usage constraints*: if a process is moved from one machine to another, it requires an adequate amount of capacity on both machines.
- *Spread constraints*: the set of machines is partitioned into locations. The processes of the same service should be allocated to machines in a number of distinct locations.
- *Dependency constraints*: the set of machines are partitioned into neighbourhoods. Then, if there is a service depends on another service, the process of the first one should be assigned to the neighbouring machine of the second one or vice versa.

A solution that meets all the hard constraints is called a feasible solution. On the other hand, soft constraints do not affect the feasibility of the resultant solution but its quality. To achieve a high-quality solution, the soft constraints must be satisfied as much as possible. The soft constraints of GMRP are:

- *Load cost. SC*$(m, r)$: represents the safety capacity of a resource $r \in R$ on a machine $m \in M$. The loaded cost is defined per resource and corresponds to the used capacity above the safety capacity; more formally:

$$f_1(r) = \sum_{m \in M} max(0, U(m, r) - SC(m, r))$$

- *Balance cost*: In order to balance the availability of resources, let the set of triples is $B \subseteq R^2XN$ [22]. For a given triple $b = (r_1^b, r_2^b, target^b) \in B$, the balance cost is:

$$f_2(b) = \sum_{m \in M} max(0, target^b.A(m, r_1^b) - A(m, r_2^b))$$

- *Process move cost*: represents the cost of moving a process from its current machine to a new one as follow:

$$f_3 = \sum_{p \in P} bool(Map_0(p) \neq Map(p)).PMC(p)$$

- *Service move cost*: represents the maximum number of moved processes over services:

$$f_4 = \max_{s \in S}(|\{p \in s | Map(p) \neq Map_0(p)\}|)$$

- *Machine move cost*: represents the sum of all moves weighted by relevant machine cost:

$$f_5 = \sum_{p \in P} MMC(Map_0(p), Map(p))$$

The total objective cost is a weighted sum of all previous costs is calculated as follows:

$$f(Map) = \sum_{r \in R} wt_1(r).f_1(r) + \sum_{b \in B} wt_2(b) + f_2(b) + wt_3.f_3 + wt_4.f_4 + wt_5.f_5 \tag{6}$$

where $wt_1(r)$, $wt_2(b)$, $wt_3$, $wt_4$ and $wt_5$ define the importance of each individual cost.

Various meta-heuristic and hybrid algorithms have been proposed for GMRP. These are large neighbourhood search methods [24], constraint programming based large neighbourhood search methods [25,26], variable neighbourhood search algorithm [27], multi-start iterated local search [28], iterated local search algorithm [29], simulated annealing algorithm [30,31], multi-neighbourhood local search algorithm [23], late acceptance hill-climbing [32], steepest descent algorithm [33] and memetic algorithm [34]. The above algorithms are single solution-based methods, and a few of them use constraint programming as a core component. The single solution-based algorithms are easy to implement, but they work well only on smooth search spaces. Also, to attain a high-quality solution, an expert is needed to customise several algorithmic components. Furthermore, the same algorithm cannot be used to solve different problem domains. The constraint programming-based algorithms need several changes to be used for large-scale problem instances. This could be the main reason for why no one of these algorithms achieved the best-known results for all tested instances.

More details about all the participated teams in ROADEF/EURO challenge 2012 can be found in [35]. Note that the quality of a solution is evaluated by the given solution chequer, which returns fitness measure to the best solution generated by our proposed algorithm. Another important aspect of this challenge is the time limit. It was stated that *"The maximum execution time will be fixed to 5 min by instance on a core2duo E8500 3.16Mhz with 4Go RAM on Debian 64 or Win7 64 bits"*. All methods have to finish within the 5-minute timeframe to ensure the fairness of the comparison.

## 3. Proposed framework

This work proposes a hyper-heuristic (HH) framework to automatically select a local search algorithm (LS) and the internal operators. The proposed HH framework uses a pool of LS algorithms and a pool of corresponding operators. It adaptively determines the appropriate combination of LS and operators at each decision points during the problem-solving process. Fig. 1 shows the flowchart of the HH framework, which has three key components: two-stage hyper-heuristic (denoted as HH-LS and HH-OP), the ranking mechanism and a population of solutions.

The proposed framework starts with the parameter settings, including the population size and the LS algorithms. Then an initial population of solutions are generated. Next, it starts the main loop of iterations. Each iteration has two stages: HH-LS to select LS and HH-OP to choose the internal operators for the selected LS. The selected LS and operators are applied to the current solution to generate a new one. The new solution will be accepted if it shows improvement compared to the original one. Then the framework will update the rank of the selected LS and operators. If the stopping criterion is satisfied, the search will stop and return the best solution. Otherwise, the population will be updated, and a new iteration will be started.

In the following subsections, we discuss the main components of the proposed framework in details.

### 3.1. Population of solutions

A pure single solution-based meta-heuristic algorithm is not very effective in handling large and heavily constrained search spaces such as those considered in this paper. To address this issue, we use a population of solutions within the proposed hyper-heuristic framework. The use of a population of solutions enables parallel search. That helps LS handle complex search landscape and adapt to different situations quicker, as the search

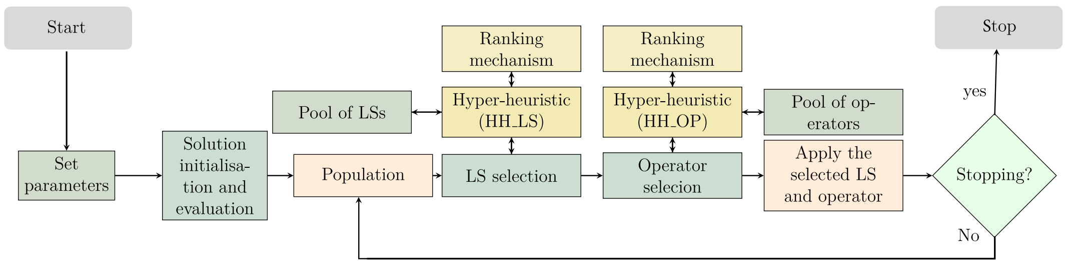

**Fig. 1.** The flowchart of the proposed hyper-heuristic framework.

points are scattered around in different areas of the search space. To maintain a proper balance between exploration and exploitation of the search paths, we propose a distance-based strategy to update the population of solutions. The proposed strategy ensures that the current population of solutions consists of high-quality and diverse solutions. The distance-based updating strategy uses two metrics: (1) solution quality, which is calculated using the objective function of the COP problem, such as the number of bins in MCBPP; and (2) the distance between solutions, that measures dissimilarity between the current solution and other solutions in the population. Two solutions are identical if the distance between them is zero. The distance calculation formula is presented in Section 3.5. To move the search towards high quality and high diversity areas; in this work, the quality and distance metrics are considered as a multi-objective (bi-criteria) problem. Pareto-domination is used to add a new solution in the population. Solution $S$ is Pareto-dominated by solution $A$ if the quality of $A$ is better than or equal to $S$ and the distance between $A$ and the current population of solutions is greater than $A$. Based on this Pareto-domination, the most dominated solution found in the current population will replace the current solution. If there is no domination, we use the FIFO updating strategy, that is first-in-first-out. By having this bi-criteria updating strategy, the solutions in the population tend to be spread along the Pareto-front. In this work, all solutions in the population are randomly generated.

### 3.2. Two-stage Hyper heuristics

Hyper-heuristic (HH) is a high-level search methodology that search for problem solving methods rather than problem solutions [36–38]. Hence HH is less problem dependent and can deliver solutions to a diverse set of problem domains. HH solves a given optimisation problems by automating the process of selection/producing heuristics [39–41]. HH approaches often have two levels: high-level heuristics and low-level heuristics. The low-level heuristic is comprised of a set of heuristics that can be directly applied to the problem in hand [39,42]. The high-level heuristic has a selection or even construction strategy, which determines how low-level heuristic should be used to solve the current problem instance. It also determines the acceptance criterion, that is whether to accept a solution generated by the selected low-level heuristic. Typical HH methodology consists of the following steps:

- Step 1: Call the selection strategy to identify a low-level heuristic from a given pool.

$$LLH_i = LLH_i + ((f_c - f_n)/(f_c + f_n)) \tag{7}$$

$$LLH_j = LLH_j - ((f_c - f_n)/(f_c + f_n)/(L - 1)), \forall j \neq i, j, i = \{1..L\} \tag{8}$$

$$LLH_i = LLH_i - ((f_c - f_n)/(f_c + f_n)) \tag{9}$$

$$LLH_j = LLH_j + ((f_c - f_n)/(f_c + f_n)/(L - 1)), \forall j \neq i, j, i = \{1..L\} \tag{10}$$

- Step 2: Randomly select a solution from the population.
- Step 3: Apply the selected low-level heuristic to that solution and call the objective function to calculate the quality of the generated solution.
- Step 4: Apply the acceptance criterion to decide whether to accept that generated solution.
- Step 5: Update the hyper-heuristic parameters and the solutions.
- Step 6: Check the stopping condition. If met, stop the search, and return the best solution. Otherwise, go back to Step 1.

In this work, a two-stage hyper-heuristic framework (denoted as HH-LS and HH-OP) is designed for the above process. The first stage (HH-LS) controls the selection of LS, while the second one (HH-OP) chooses internal operators for the selected LS. The framework alternates between these two stages to effectively solve the given problem instances. In the following subsections, we discuss the main components of HH-LS and HH-OP.

### 3.3. HH-LS

HH-LS is a multi-local-search hyper-heuristic component which is used in the first stage and responsible for LS selection (see Fig. 1). HH-LS takes as an input a pool of LSs and then uses the selection strategy to decide which one should be selected for the current problem instances. The high-level heuristic and the low-level heuristics of HH-LS are:

**High-Level Heuristics**. The high-level heuristic has two components: selection strategy and the acceptance criterion. The selection strategy is responsible for selecting a low level of heuristic (LS) from the given pool. It invokes two tasks credit assignment and LS selection.

- **Credit assignment:** The primary role of this mechanism is to record how well each low-level heuristic performed. In our case, these are local search algorithms. In this study, both reward and punishment are used. If the applications of the $i$th low-level heuristic (LS) leads to an improvement in the objective function, we reward this LS using Formula (7) and punish other LSs using Formula (8). Otherwise, we punish the $i$th LS by Formula (9) and reward others by Formula (10). In the formulas, $f_c$ and $f_n$ are the performance indicators of the current and new solution, respectively. It should be noted that in this work, $f$ represents the quality and diversity of the solution obtained by the applied LS algorithm. The quality is calculated by the objective (or cost) function. The diversity is determined by the proposed ranking mechanism (See Section 3.5).
- **LS Selection:** The selection is made by Multi-Armed Bandit (MAB). MAB is an on-line selection mechanism that use the recorded performance of all LSs to select one LS using Eq. (11). The recorded performances of all LSs are retrieved

from the credit assignment mechanism.

$$max_{i=1\ldots N_{op}}\left(p_{i(t)}+c\sqrt{\frac{2\log\sum_{j=1}^{N_{op}}n_{j(t)}}{n_{i(t)}}}\right) \quad (11)$$

where $t$ is the current generation, $N_{op}$ is the number of LS algorithms, and $c$ is the scaling factor which is set to 7. The pseudocode of the multi-armed bandit (MAB) is shown in Algorithm 1.

**Algorithm 1:** The pseudocode of MAB

```
for $i = 1$ to $LLH$ do
    $n_i \leftarrow 0$;
    $p_i \leftarrow 0$;
end
while (Not_Terminated) do
    if one or more LLHs not applied yet then
        $LLH \leftarrow$ Randomly select one $LLH$ from the given set;
        Apply the selected $LLH$;
    end
    else
        Use Eq. (11) to pick one $LLH$;
        Apply the selected $LLH$;
    end
    Update $n_i$, $n_i \leftarrow n_i + 1$;
    Update $p_i$ using Eqs. (7 to 10);
end
```

The second component acceptance criterion, which is the Monte Carlo (MC) acceptance criterion for this paper, the same as that in [42]. In MC, a new solution will be accepted if it is better than the initial one or satisfying MC acceptance rule: $R < \exp(\delta)$ where $R$ is a random number between zero and one, and $\delta$ is the difference in the quality.

**Low-Level Heuristics**. The low-level heuristics here are in a pool of LS algorithms. These LSs use different procedures to navigate the search space. The utilised LSs are:

- **Simulated Annealing (SA).** SA is a stochastic single solution-based algorithm proposed by [43]. It uses a probabilistic rule that allows the acceptance of worse solutions to escape from the local optima. SA starts with an initial solution, $S_0$, and then generates a neighbourhood solution, $S_1$. Replace the new solution ($S_1$) with current solution ($S_0$) if its better or satisfying the probability condition ($P$):

$$P = exp^{\frac{-(f(x')-f(x))}{t}} \quad (12)$$

where $f(x)$ is the fitness of the current solution, $f(x')$ is the new solution and $t$ is the current temperature. The temperature will be updated during the search as follows $t = t^{*}\,\alpha$.

- **Iterated Local Search (ILS)**. ILS is a simple, yet effective, single solution-based heuristic introduced by [44]. ILS has been successfully used to solve various optimisation. It starts with an initial solution, $S_0$, and then iteratively explore its neighbour. ILS escapes from a local optimum point by perturbing the current solution instead of generating a new one. ILS iteratively applies the following three procedures: local search procedure to explore the neighbour of the given solution, a perturbation procedure to provide a starting solution for the local search procedure and acceptance criterion to either accept or reject the generated solution.
- **Late Acceptance Hill Climbing (LAHC)**. LAHC is a recent single solution-based heuristic [45]. It always accepts an improving local search move but also accept the worse solution if it is not worse than the quality of a saved solution which was the current one several steps before. Given an initial solution, $S_0$, randomly generates a neighboured solution, $S_1$. Replace $S_1$ with the $S_0$ if the quality of $S_1$ is better than $S_0$ or better than $f_v$ where $f_v$ is the quality of $v^{it}$ solution saved in list $L$ that contains qualities of solutions obtained in recent iterations. $v$ is calculated as follows:

$$v = I \mod L_{size} \quad (13)$$

where $L_{size}$ is the size of $L$ and $I$ is the iteration counter. At each iteration, LAHC will insert the quality of the current solution into the beginning of $L$ and removes the last one from the end.
- **Great Deluge (GD)**. GD is a single solution-based heuristic was introduced by [46]. It uses an initial solution as a starting basis and then generated a neighbourhood solution. GD always accepts improved solutions while worse solutions may also be accepted if it is better than the threshold. Given an initial solution, $S_0$, generates a neighbourhood solution, $S_1$. Replace $S_1$ with the $S_0$ if the quality of $S_1$ is better than $S_0$ or lower than the *level*. Initially, the value of *level* is set equal to the fitness value of the initial solution. At each iteration, the value of *level* is decreased by $\varepsilon$ as follows:

$$level = level - \varepsilon \quad (14)$$

and

$$\varepsilon = (f(S_1) - f(best_{sol}))/NI \quad (15)$$

where $NI$ is the number of iterations and the search process will stop when the *level* value is lower than the best solution found so far.
- **Steepest Descent (SD)**. SD a single solution-based heuristic which generates a neighbourhood solution, $S_1$, by randomly selecting one item (or job) from the initial solution, $S_0$, and then move it into a different place (bin or machine) [1]. SD replaces $S_1$ with the $S_0$ if the quality of $S_1$ is better than $S_0$. If not, $S_1$ will be rejected, and a new iteration will be started. The search process will be terminated after a fixed number of iterations.

### 3.4. HH-OP

HH-OP is used in the second stage to select an operator for the selected LS by HH-LS (see Fig. 1). HH-OP uses various neighbourhood operators as low-level heuristics. It takes as an input a pool of operators and then uses the selection strategy to decide which one should be applied with the selected LS. The high-level heuristic and the low-level heuristics of the proposed HH-OP are:

- **High-Level Heuristics**. The high-level heuristic of HH-OP involves a selection strategy to select a low-level heuristic (operator). In this work, we use the reinforcement learning approach as our selection strategy to decide which operator should be picked from the given pool. The utilised strategy maintains a probabilities distribution for all operators in the pool. Based on these probabilities, we use the roulette wheel selection mechanism to select one operator for LS. Then based on the quality of the obtained solution, the probabilities (P) of all operators are updated as the search progress using Eqs. (16) and (17). Eq. (16) is used to update the probability of the selected operator, while Equation (17) updates the probabilities of the rest of the operators. In Eqs. (16) and (17) $\alpha^{+}$ and $\alpha^{-}$ represent the reward and penalty values and set between zero and one. $\beta_{OP}$ represents the improvement strength obtained by the applied operator $OP$ in which 1 is used for success and 0 otherwise. $|M|$ is the number of operators in the pool. In this selection strategy,

**Table 1**
Parameter settings of the HH Framework.

| Parameter | Tested range | Suggested value |
|---|---|---|
| Population size ($P$) | – | 4 |
| Stopping condition | – | 300 s |
| Initial temperature ($t$) for SA | $10^4$-$10^{10}$ | $10^4$ |
| $\alpha$ for SA | 0.2–0.9 | 0.6 |
| Local search termination criterion for ILS | 5–50 | 10 |
| Perturbation size for ILS | 2%–20% of processes | 5% of processes |
| List size ($L_{size}$) for LAHC | 5–30 | 20 |
| Iteration counter ($I$) for LAHC | 3–15 | 10 |
| Number of iterations $NI$ for GD | 100–1500 | 1000 |
| Number of iterations $NI$ for SD | 5–30 | 10 |

**Table 2**
The computational results of the compared algorithms.

| # | Instance | RN-LS | HH_1 | HH_2 | HH |
|---|---|---|---|---|---|
| 1 | 25A | 89 | 78 | 75 | **69** |
| 2 | 50A | 153 | 148 | 141 | **135** |
| 3 | 50B | 159 | 155 | 151 | **145** |
| 4 | 100A | 283 | 278 | 275 | **256** |
| 5 | 100B | 290 | 284 | 275 | **264** |
| 6 | 200A | 526 | 521 | 519 | **503** |
| 7 | a1_1 | 44,306,501 | 44,306,501 | 44,306,501 | **44,306,501** |
| 8 | a2_5 | 310,243,696 | 308,367,604 | 308,367,510 | **307,150,823** |
| 9 | b_1 | 3,305,899,959 | 3,301,109,217 | 3,301,109,203 | **3,291,069,367** |
| 10 | b_10 | 18,512,416,409 | 18,227,132,678 | 18,227,132,531 | **18,048,187,124** |
| 11 | x_1 | 3,044,618,079 | 3,044,593,197 | 3,044,591,048 | **3,044,417,080** |
| 12 | x_10 | 17,815,989,954 | 17,815,989,117 | 17,815,989,098 | **17,815,981,141** |

we increase the probability of the applied operator if it leads to a better solution and decreases it otherwise. Eqs. (16) and (17) ensure that all operators have a chance to be selected at least once during the search process.

$$P_{OP,t+1} = P_{OP} + \alpha^{+}\beta_{OP}(1 - P_{OP}) - \alpha^{-}(1 - \beta_{OP})P_{OP} \tag{16}$$

$$P_{OP,t+1} = P_{OP} - \alpha^{+}\beta_{OP}P_{OP} + \alpha^{-}(1 - \beta_{OP})\left(\frac{1}{|M| - 1} - P_{OP}\right) \tag{17}$$

- **Low-Level Heuristics**. This component has a pool of problem-specific operators. These operators aim to generate neighbourhood solutions by modifying the current one, providing no violation of constraints. In this work, we propose a ranking mechanism to exclude bad operators from the current pool to allow the HH-OP to focus on selecting the most effective operator (See Section 3.5). The pool consists of
  - **Single swap**: Randomly select two processes (items) from two different machines (bins) and interchanges them.
  - **Double swap**: Randomly select four processes (items) from two different machines (bins) and interchanges them.
  - **Single move**: Randomly select a process (item) from a machine (bin) and moves it to a different machine (bin).
  - **Double move**: Randomly select two processes (items) from a machine (bin) and moves them to a different machine (bins).
  - **Swap–Move**: Apply swap operator followed by the move operator.
  - **Move–Swap**: Apply move operator followed by the swap operator.
  - **Move-Big**: A process (item) with large size is selected and moved to a different machine (bin). The size of such a process (item) is equal to the total resources required by this process (item).

$$\begin{array}{c|ccccc}
 & \multicolumn{5}{c}{\text{location index}} \\
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 3 & 4 & 2 & 5 & 6 \\
2 & 1 & 1 & 3 & 5 & 5 \\
3 & 3 & 4 & 2 & 1 & 2 \\
4 & 4 & 5 & 4 & 2 & 4 \\
5 & 3 & 6 & 1 & 4 & 3
\end{array}$$

**Fig. 2.** An example of frequency matrix.

  - **Swap-Move-Big**: Apply swap followed by the big move operator.

### 3.5. Ranking mechanism

In this work, we propose a ranking mechanism for HH framework using entropy and information theory. It calculates the diversity of the solutions obtained during the execution of HH-LS and HH-OP. The ranking associates a frequency matrix with each solution. The matrix records how many times the item (or job) has been allocated to the same position. An example of a frequency matrix for one solution is shown in Fig. 2. In this example, the matrix involves five items (or jobs) and five locations. The matrix can be read as follows: item (or job) 1 has been assigned to location 1 three times, to location 2 four times, to location 3 two times, to location 4 five times and to location 5 six times; and so on for the other items (or jobs). We then use entropy equations (Eqs. (18) and (19)) to calculate the diversity of a given solution.

$$div_i = \frac{\sum_{j=1}^{e} \frac{e_{ij}}{m} \cdot \log \frac{e_{ij}}{m}}{-\log e} \tag{18}$$

$$div = \frac{\sum_{i=1}^{2} div_i}{e} \tag{19}$$

**Table 3**
The computational results of the compared algorithms on MCBPP instances.

| Instance | RN-LS | | | HH_1 | | | HH_2 | | | HH | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Avg | Std | Time | Avg | Std | Time | Avg | Std | Time | Avg | Std | Time |
| 25A | 109.31 | 10.89 | 244 | 95.01 | 9.77 | 231 | 89.12 | 5.42 | 329 | **77.83** | **1.22** | **192** |
| 50B | 198.55 | 8.42 | 225 | 194.36 | 6.56 | 169 | 189.02 | 6.13 | 161 | **154.3** | **3.07** | **129** |
| 100A | 302.08 | 10.97 | 255 | 293.47 | 7.21 | 191 | 292.45 | 7.18 | 185 | **285.53** | **1.8** | **120** |
| 100B | 298.44 | 6.12 | 239 | 294.51 | 5.01 | 203 | 291.08 | 4.96 | 171 | **282.06** | **1.89** | **139** |
| 200A | 597.99 | 8.07 | 300 | 569.11 | 5.83 | 267 | 561.71 | 5.66 | 213 | **524.6** | **3.38** | **189** |

**Table 4**
The computational results of the compared algorithms on GMRP instances.

| Instance | RN-LS | | HH_1 | | HH_2 | | HH | |
|---|---|---|---|---|---|---|---|---|
| | Avg | Time | Avg | Time | Avg | Time | Avg | Time |
| a1_1 | 44,306,911.04 | 289 | 44,306,613.12 | 276 | 44,306,612.03 | 263 | **44,306,581.00** | **151** |
| a2_5 | 310,243,956.08 | 247 | 308,367,946.31 | 224 | 308,367,942.77 | 217 | **307,518,303.50** | **169** |
| b_1 | 3,305,899,967.11 | 274 | 3,301,109,375.38 | 206 | 3,301,109,361.31 | 189 | **3,291,520,458.00** | **156** |
| b_10 | 18,512,416,839.41 | 169 | 18,227,132,714.51 | 158 | 18,227,132,703.47 | 153 | **18,048,620,264.00** | **137** |
| x_1 | 3,044,618,263.11 | 293 | 3,044,994,288.48 | 277 | 3,044,799,148.93 | 262 | **3,044,648,400.00** | **213** |
| x_10 | 17,815,989,966.47 | 291 | 17,815,989,427.01 | 278 | 17,815,989,394.77 | 271 | **17,815,985,753.00** | **216** |

where $e_{ij}$ is the frequency of assigning item $i$ to location $j$; $m$ is the number of items; $div_i$ is the entropy for item $i$ and $div$ is the entropy for one solution $(0 \geq div \leq 1)$. In this work, the entropy $div$ value is used as an indicator of solution diversity. The proposed ranking mechanism uses $div$ in both HH-LS and HH-OP (see Fig. 1). In HH-LS, $div$ is used with the credit assignment component of the selection strategy to evaluate the performance of each LS by considering the quality and diversity.

In HH-OP, using many operators can empower the search performance to explore a wider area of the search space. Nevertheless, HH-OP might faeces some difficulties in determining the best operator in a short period. Also, it is very difficult to know in advance whether the operator pool is sufficient or next. Hence the ranking management mechanism is proposed to control the size and diversity of the pool of operator. It uses a fixed-length list to keep the top-performing operators. Obviously, the size of the list is much less than the total number of operators. Initially, all operators have the same probability of being included. The HH-OP only selects the operator from this list, which will be updated based on the performance of these operators. Then a roulette wheel selection mechanism is used to add good operators into the list. Apparently, an operator with better performance is more likely to be selected. The quality of each operator is the combination of its past performance and the $div$ value.

## 4. Experiments and results

The evaluation of our HH framework on MCBPP and GMRP is described in this section. Firstly, we introduce the datasets, then the calibration of the parameters, followed by the evaluation of the effectiveness of the proposed HH framework. Finally, the comparison with state-of-the-art algorithms on both MCBPP and GMRP benchmarks.

### *4.1. Benchmark instances*

Well-known benchmark instances were used for both MCBPP and GMRP. For MCBPP, 10 different types of instances from [16,19,20] were used. Each type has 40 instances of four different sizes, e.g. 10 instances for each size. We focus on classes 1, 6, 7, 9 and 10, that is 200 instances all together because the other five classes are known as simple instances. They are easily solvable by simple greedy heuristics [19]. For the GMRP benchmark, there are 30 instances provided by Google. They are categorised in three groups, namely $a$, $b$ and $x$. These instances have different characteristics in terms of the number of machines, the number of processes and neighbourhood structures.

**Table 5**
Statistical test of the compared algorithms on MCBPP and GMRP instances.

| Instance | RN-LS | HH_1 | HH_2 |
|---|---|---|---|
| 25A | S+ | S+ | S+ |
| 50B | S+ | S+ | S+ |
| 100A | S+ | S+ | S+ |
| 100B | S+ | S+ | S+ |
| 200A | S+ | S+ | S+ |
| a1_1 | S = | S = | S = |
| a2_5 | S+ | S+ | S+ |
| b_1 | S+ | S+ | S+ |
| b_10 | S+ | S+ | S+ |
| x_1 | S+ | S+ | S+ |
| x_10 | S+ | S+ | S+ |

**Table 6**
Results of HH for MCBPP instances.

| Size | Class | BKLB | BKUB | HH best | HH avg | Std | T (s) |
|---|---|---|---|---|---|---|---|
| 25 | 1 | 69 | 69 | **69** | 77.83 | 1.22 | 192 |
| 25 | 6 | 101 | 101 | 105 | 108.73 | 2.21 | 154 |
| 25 | 7 | 96 | 96 | **96** | 105.86 | 3.14 | 140 |
| 25 | 9 | 73 | 73 | **73** | 91.63 | 2.27 | 145 |
| 24 | 10 | 80 | 80 | 86 | 91.43 | 2.01 | 181 |
| 50 | 1 | 135 | 135 | **135** | 154.3 | 3.07 | 129 |
| 50 | 6 | 214 | 215 | 220 | 224.66 | 3.35 | 98 |
| 50 | 7 | 196 | 197 | 205 | 212.4 | 1.08 | 150 |
| 50 | 9 | 144 | 145 | **145** | 160 | 2.3 | 211 |
| 51 | 10 | 170 | 170 | 181 | 185.4 | 1.19 | 165 |
| 100 | 1 | 255 | 260 | **256** | 285.53 | 1.8 | 120 |
| 100 | 6 | 405 | 410 | 419 | 422.33 | 1.57 | 173 |
| 100 | 7 | 398 | 405 | 411 | 419.4 | 2.46 | 201 |
| 100 | 9 | 257 | 267 | **264** | 282.06 | 1.89 | 139 |
| 99 | 10 | 330 | 330 | 340 | 343.66 | 3.66 | 154 |
| 200 | 1 | 503 | 510 | **503** | 524.6 | 3.38 | 189 |
| 200 | 6 | 803 | 811 | 823 | 827.96 | 2.13 | 132 |
| 200 | 7 | 799 | 802 | 812 | 815.33 | 2.18 | 196 |
| 200 | 9 | 503 | 513 | **513** | 527.36 | 2.38 | 184 |
| 201 | 10 | 670 | 670 | 681 | 688.4 | 1.5 | 117 |

### *4.2. Parameters calibration*

The proposed HH framework has few tuneable parameters. The values of these parameters were set based on preliminary experiments over both MCBPP and GMRP problems. In our initial experiments, we tested the proposed algorithm 31 independent runs using different parameter combinations. The best values for all parameters are recorded. The population size in the experiment is equal to the number of threads. Each LS algorithm has two parameters except GD, which has only one parameter. The

**Table 7**
Best Solutions on MCBPP Instances achieved by HH.

| # | Size | Class 1 | | | Class 6 | | | Class 7 | | | Class 9 | | | Class 10 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | LB | UB | HH | LB | UB | HH | LB | UB | HH | LB | UB | HH | LB | UB | HH |
| 1 | S1 | 6 | 6 | **6** | 10 | 10 | **10** | 9 | 9 | **9** | 7 | 7 | **7** | 8 | 8 | **8** |
| 2 | | 7 | 7 | **7** | 10 | 10 | **10** | 9 | 9 | **9** | 7 | 7 | **7** | 8 | 8 | **8** |
| 3 | | 7 | 7 | **7** | 10 | 10 | **10** | 10 | 10 | **10** | 7 | 7 | **7** | 8 | 8 | **8** |
| 4 | | 7 | 7 | **7** | 10 | 10 | 11 | 10 | 10 | **10** | 7 | 7 | **7** | 8 | 8 | 9 |
| 5 | | 7 | 7 | **7** | 10 | 10 | 12 | 10 | 10 | **10** | 7 | 7 | **7** | 8 | 8 | **8** |
| 6 | | 7 | 7 | **7** | 10 | 10 | **10** | 10 | 10 | **10** | 7 | 7 | **7** | 8 | 8 | **8** |
| 7 | | 7 | 7 | **7** | 10 | 10 | **10** | 10 | 10 | **10** | 7 | 7 | **7** | 8 | 8 | 10 |
| 8 | | 7 | 7 | **7** | 10 | 10 | **10** | 9 | 9 | **9** | 8 | 8 | **8** | 8 | 8 | **8** |
| 9 | | 7 | 7 | **7** | 10 | 10 | 11 | 10 | 10 | **10** | 8 | 8 | **8** | 8 | 8 | 9 |
| 10 | | 7 | 7 | **7** | 11 | 11 | **11** | 9 | 9 | **9** | 8 | 8 | **8** | 8 | 8 | 10 |
| 1 | S2 | 13 | 13 | **13** | 21 | 21 | **21** | 21 | 21 | **21** | 14 | 14 | **14** | 17 | 17 | **17** |
| 2 | | 13 | 13 | **13** | 21 | 21 | **21** | 18 | 18 | **18** | 14 | 14 | **14** | 17 | 17 | 20 |
| 3 | | 13 | 13 | **13** | 21 | 21 | **21** | 21 | 21 | 22 | 14 | 14 | **14** | 17 | 17 | 20 |
| 4 | | 13 | 13 | **13** | 21 | 21 | 22 | 18 | 18 | **18** | 14 | 14 | **14** | 17 | 17 | **17** |
| 5 | | 13 | 13 | **13** | 21 | 21 | 22 | 21 | 21 | **21** | 14 | 14 | **14** | 17 | 17 | **17** |
| 6 | | 14 | 14 | **14** | 22 | 22 | **22** | 18 | 18 | 23 | 14 | 15 | 15 | 17 | 17 | 19 |
| 7 | | 14 | 14 | **14** | 21 | 22 | 25 | 21 | 22 | 22 | 15 | 15 | **15** | 17 | 17 | **17** |
| 8 | | 14 | 14 | **14** | 22 | 22 | **22** | 18 | 18 | **18** | 15 | 15 | **15** | 17 | 17 | **17** |
| 9 | | 14 | 14 | **14** | 22 | 22 | **22** | 22 | 22 | **22** | 15 | 15 | **15** | 17 | 17 | 20 |
| 10 | | 14 | 14 | **14** | 22 | 22 | **22** | 18 | 18 | 20 | 15 | 15 | **15** | 17 | 17 | **17** |
| 1 | S3 | 25 | 26 | **25** | 40 | 41 | 42 | 40 | 41 | 41 | 25 | 26 | 26 | 33 | 33 | 36 |
| 2 | | 26 | 26 | **26** | 41 | 41 | 45 | 39 | 40 | 42 | 26 | 27 | 27 | 33 | 33 | 34 |
| 3 | | 26 | 26 | **26** | 41 | 41 | 42 | 41 | 41 | **41** | 26 | 27 | **26** | 33 | 33 | **33** |
| 4 | | 25 | 26 | **25** | 40 | 41 | 42 | 39 | 40 | 43 | 25 | 26 | 26 | 33 | 33 | 35 |
| 5 | | 25 | 26 | **25** | 40 | 41 | 41 | 40 | 41 | 41 | 25 | 26 | 26 | 33 | 33 | 34 |
| 6 | | 25 | 26 | 26 | 40 | 41 | 41 | 39 | 40 | 40 | 26 | 27 | **26** | 33 | 33 | **33** |
| 7 | | 25 | 26 | **25** | 40 | 41 | 41 | 40 | 41 | 41 | 26 | 27 | **26** | 33 | 33 | **33** |
| 8 | | 26 | 26 | **26** | 41 | 41 | **41** | 39 | 40 | 41 | 26 | 27 | 27 | 33 | 33 | 34 |
| 9 | | 26 | 26 | **26** | 41 | 41 | **41** | 41 | 41 | **41** | 26 | 27 | 27 | 33 | 33 | **33** |
| 10 | | 26 | 26 | **26** | 41 | 41 | 43 | 40 | 40 | **40** | 26 | 27 | 27 | 33 | 33 | 35 |
| 1 | S4 | 50 | 51 | **50** | 80 | 81 | 82 | 80 | 80 | 81 | 50 | 51 | 51 | 67 | 67 | 68 |
| 2 | | 50 | 51 | **50** | 80 | 81 | 81 | 80 | 80 | **80** | 50 | 51 | 51 | 67 | 67 | 68 |
| 3 | | 50 | 51 | **50** | 80 | 81 | 84 | 80 | 80 | **80** | 50 | 51 | 51 | 67 | 67 | 68 |
| 4 | | 50 | 51 | **50** | 80 | 81 | 83 | 80 | 80 | **80** | 50 | 51 | 51 | 67 | 67 | 68 |
| 5 | | 50 | 51 | **50** | 80 | 81 | 81 | 80 | 80 | **80** | 50 | 51 | 51 | 67 | 67 | 68 |
| 6 | | 50 | 51 | **50** | 80 | 81 | 82 | 79 | 80 | 83 | 50 | 51 | 51 | 67 | 67 | 69 |
| 7 | | 50 | 51 | **50** | 80 | 81 | 82 | 80 | 81 | 83 | 50 | 51 | 51 | 67 | 67 | 68 |
| 8 | | 51 | 51 | **51** | 81 | 81 | 84 | 79 | 80 | 84 | 51 | 52 | 52 | 67 | 67 | 69 |
| 9 | | 51 | 51 | **51** | 81 | 81 | 82 | 81 | 81 | **81** | 51 | 52 | 52 | 67 | 67 | 68 |
| 10 | | 51 | 51 | **51** | 81 | 82 | 82 | 80 | 80 | **80** | 51 | 52 | 52 | 67 | 67 | **67** |

$t$ value in SA controls the acceptance ratio of worse solutions, and it gradually decreases by the value of $\alpha$. The actual parameter settings are presented in Table 1.

### 4.3. Effectiveness evaluation

This experiment examines the benefit of using the proposed improvements on HH performance. Thus, to evaluate the impact of each of the integrated components within the proposed HH, three different algorithms were implemented and tested. These are:

- RN-LS which uses a random selection for the local search algorithms and the associated neighbourhood structures (operators).
- HH_1 is similar to the proposed HH but without the population of solutions.
- HH_2 is similar to the proposed HH but without the ranking mechanism.

We randomly selected 12 instances for the purpose of evaluating the effectiveness: 6 instances from MCBPP and 6 instances from GMRP. To ensure a fair comparison, all algorithms (RN-LS, HH_1, HH_2 and HH) used the same initial solution, number of runs, stopping condition and computer resources. All algorithms were executed for 31 independent runs overall instances. For each instance, we report the best (Best), the average (Avg), standard deviation (Std) and the time required to find the best solution. The best results achieved by RN-LS, HH_1, HH_2 and HH are reported in Table 2. The best results obtained by the corresponding algorithms are highlighted in bold. From Table 2, one can clearly see that HH outperforms RN-LS, HH_1 and HH_2 across all instances.

In Fig. 3, we plot the deviation from the best-known results for the selected instances. For MCBPP instances (Figure (a)) the deviation is calculated from the lower bound results that were obtained by relaxing the problem constraints; while for GMRP instances (Figure (b)) the deviation is calculated from the best-known results published in the literature. From Figure (a), we can see that HH results are the lowest ones compared to those of other algorithms (RN-LS, HH_1, and HH_2) in which, overall instances, the results are very close to the lower bounds. For GMRP instances (Figure (b)), HH matched the best-known results for all instances. One can also see that HH_1, and HH_2 results are almost the same, which clearly indicates the benefit of the hyper-heuristic components when compared to the random selection method. The results in both figures (Figure (a) and Figure (b)) demonstrate that the performance of RN-LS, HH_1, and HH_2 varies from one domain to another, whereas HH is stable on both domains.

In Tables 3 and 4, we present the average (Avg) and standard deviation (Std) results of HH and the compared algorithms (RN-LS, HH_1 and HH_2). As shown in Tables 3 and 4, the average results and the standard deviation of HH are better than RN-LS, HH_1 and HH_2 across all tested instances. To confirm these positive results, we also applied the Wilcoxon statistical test with a significance level of 0.05, where $S+$ means HH is statistically better than other algorithms, $S-$ means that it is not significant, as shown in Table 5. Table 5 reveals that HH is statistically better than RN-LS, HH_1 and HH_2 on all instances.

We now compare the computational time (in seconds) consumed by the compared algorithms (HH, RN-LS, HH_1 and HH_2) find the best results for all tested instances, as shown in Tables 3 and 4. It is clear from the tables that the computational time of HH is lower than RN-LS, HH_1 and HH_2 for all tested instances. The reported results demonstrate that the proposed HH achieves high-quality solutions with lower computational times when compared to RN-LS, HH_1 and HH_2 across all instances. This positive results clearly indicate the benefits of the integrated components within the proposed HH.

### 4.4. Comparing HH results with the state-of-the-art

This section compares the results obtained by HH with the state-of-the-art algorithms applied to MCBPP and GMRP.

#### 4.4.1. Results on MCBPP benchmark instances

This section presents the results of the proposed HH and the current state-of-the-art solutions: Best Known Lower Bounds (BKLBs) and Upper Bounds (BKUBs). Table 6 shows these results which are aggregated by problem class; i.e., for each class, the accumulated number of bins for the 10 instances is reported. Columns one to four present the problem size, the problem class, the best-known lower and upper bounds. To the right of these columns are the best, the average number of bins, the standard deviation and computational time obtained by HH. The best results in the row are highlighted in bold. As can be seen from the results, HH matches the known lower bounds in 9 cases. The best solutions of HH are reported in Table 7 in details in the column (HH) along with the previous lower (LB) and upper bounds (UB).

**Table 8**
Comparison between the proposed HH with the state-of-the-art algorithms — Part I.

| Instance | HH | MNLS | VNS | CLNS | Best known |
|---|---|---|---|---|---|
| a1_1 | **44,306,501** | **44,306,501** | **44,306,501** | **44,306,501** | **44,306,501** |
| a1_2 | **777,532,177** | 777,535,597 | 777,536,907 | 778,654,204 | **777,532,177** |
| a1_3 | **583,005,717** | **583,005,717** | 583,005,818 | 583,005,829 | **583,005,717** |
| a1_4 | **244,875,200** | 248,324,245 | 251,524,763 | 251,189,168 | 244,875,206 |
| a1_5 | **727,578,308** | 727,578,309 | 727,578,310 | 727,578,311 | 727,578,309 |
| a2_1 | 162 | 225 | 199 | 196 | **161** |
| a2_2 | **720,671,512** | 793,641,799 | 720,671,548 | 803,092,387 | 720,671,548 |
| a2_3 | 1,190,908,040 | 1,251,407,669 | 1,190,713,414 | 1,302,235,463 | **1,190,713,414** |
| a2_4 | **1,680,368,562** | 1,680,744,868 | 1,680,615,425 | 1,683,530,845 | 1,680,368,578 |
| a2_5 | **307,150,823** | 337,363,179 | 309,714,522 | 331,901,091 | 307,150,825 |
| b_1 | **3,291,069,367** | 3,354,204,707 | 3,307,124,603 | 3,337,329,571 | 3,291,069,369 |
| b_2 | 1,015,496,168 | 1,021,230,060 | 1,015,517,386 | 1,022,043,596 | 1,015,496,187 |
| b_3 | **156,691,279** | 157,127,101 | 156,978,411 | 157,273,705 | **156,691,279** |
| b_4 | **4,677,792,538** | 4,677,895,984 | 4,677,961,007 | 4,677,817,475 | 4,677,792,539 |
| b_5 | 922,944,567 | 923,427,881 | 923,610,156 | 923,335,604 | 922,944,697 |
| b_6 | **9,525,851,397** | 9,525,885,495 | 9,525,900,218 | 9,525,867,169 | 9,525,851,483 |
| b_7 | **14,834,456,020** | 14,842,926,007 | 14,835,031,813 | 14,838,521,000 | 14,834,456,201 |
| b_8 | **1,214,291,129** | 1,214,591,033 | 1,214,416,705 | 1,214,524,845 | 1,214,291,143 |
| b_9 | **15,885,437,254** | 15,885,541,403 | 15,885,548,612 | 15,885,734,072 | 15,885,437,256 |
| b_10 | 18,048,187,124 | 18,055,765,224 | 18,048,499,616 | 18,049,556,324 | **18,048,187,105** |
| x_1 | **3,044,417,080** | 3,060,461,509 | – | – | 3,044,418,078 |
| x_2 | 1,002,390,081 | 1,010,050,981 | – | – | **1,002,379,317** |
| x_3 | 493,938 | 493,917 | – | – | **69,970** |
| x_4 | **4,721,586,143** | 4,721,727,496 | – | – | 4,721,591,023 |
| x_5 | 521,050 | 518,250 | – | – | **54,132** |
| x_6 | 9,546,956,909 | 9,546,966,175 | – | – | **9,546,936,159** |
| x_7 | **14,252,476,502** | 14,259,657,575 | – | – | 14,252,476,508 |
| x_8 | 80,107 | 83,711 | – | – | **29,193** |
| x_9 | **16,125,531,230** | 16,125,675,266 | – | – | 16,125,562,162 |
| x_10 | **17,815,981,141** | 17,824,568,855 | – | – | 17,815,989,054 |

**Table 9**
Comparison between the proposed HH with the state-of-the-art algorithms — Part II.

| Instance | HH | LNS | MILS | SA | Best known |
|---|---|---|---|---|---|
| a1_1 | **44,306,501** | 44,306,575 | **44,306,501** | 44,306,935 | **44,306,501** |
| a1_2 | **777,532,177** | 788,074,333 | 780,499,081 | 777,533,311 | **777,532,177** |
| a1_3 | **583,005,717** | 583,006,204 | 583,006,015 | 583,009,439 | **583,005,717** |
| a1_4 | **244,875,200** | 278,114,660 | 258,024,574 | 260,693,258 | 244,875,206 |
| a1_5 | **727,578,308** | 727,578,362 | 727,578,412 | 727,578,311 | 727,578,309 |
| a2_1 | 162 | 1,869,113 | 167 | 222 | 161 |
| a2_2 | **720,671,512** | 858,367,123 | 970,536,821 | 877,905,951 | 720,671,548 |
| a2_3 | 1,190,908,040 | 1,349,029,713 | 1,452,810,819 | 1,380,612,398 | **1,190,713,414** |
| a2_4 | **1,680,368,562** | 1,689,370,535 | 1,695,897,404 | 1,680,587,608 | 1,680,368,578 |
| a2_5 | **307,150,823** | 385,272,187 | 412,613,505 | 310,243,809 | 307,150,825 |
| b_1 | **3,291,069,367** | 3,421,883,971 | 3,516,215,073 | 3,455,971,935 | 3,291,069,369 |
| b_2 | 1,015,496,168 | 1,031,415,191 | 1,027,393,159 | 1,015,763,028 | 1,015,496,187 |
| b_3 | **156,691,279** | 163,547,097 | 158,027,548 | 215,060,097 | **156,691,279** |
| b_4 | **4,677,792,538** | 4,677,869,484 | 4,677,940,074 | 4,677,985,338 | 4,677,792,539 |
| b_5 | 922,944,567 | 940,312,257 | 923,857,499 | 923,299,310 | 922,944,697 |
| b_6 | **9,525,851,397** | 9,525,862,018 | 9,525,913,044 | 9,525,861,951 | 9,525,851,483 |
| b_7 | **14,834,456,020** | 14,868,550,671 | 15,244,960,848 | 14,836,763,304 | 14,834,456,201 |
| b_8 | **1,214,291,129** | 1,219,238,781 | 1,214,930,327 | 1,214,563,084 | 1,214,291,143 |
| b_9 | **15,885,437,254** | 15,887,269,801 | 15,885,617,841 | 15,886,083,835 | 15,885,437,256 |
| b_10 | 18,048,187,124 | 18,092,883,448 | 18,093,202,104 | 18,049,089,128 | **18,048,187,105** |
| x_1 | **3,044,417,080** | 3,119,249,147 | 3,209,874,890 | – | 3,044,418,078 |
| x_2 | 1,002,390,081 | 1,018,164,308 | 1,018,646,825 | – | **1,002,379,317** |
| x_3 | 493,938 | 4,784,450 | 1,965,401 | – | **69,970** |
| x_4 | **4,721,586,143** | 4,721,702,912 | 4,721,786,173 | – | 4,721,591,023 |
| x_5 | 521,050 | 391,923 | 615,277 | – | **54,132** |
| x_6 | 9,546,956,909 | 9,546,945,537 | 9,546,992,887 | – | **9,546,936,159** |
| x_7 | **14,252,476,502** | 14,330,862,773 | 14,701,830,252 | – | 14,252,476,508 |
| x_8 | 80,107 | 98,054 | 309,080 | – | **29,193** |
| x_9 | **16,125,531,230** | 16,128,419,926 | 16,125,753,242 | – | 16,125,562,162 |
| x_10 | **17,815,981,141** | 17,861,616,489 | 17,867,789,754 | – | 17,815,989,054 |

### *4.4.2. Results on GMRP benchmark instances*

The performance of HH is compared with the state-of-the-art algorithms: MNLS [23], VNS [27], CLNS [24], LNS [25], MILS [28], SA [47], EM-LAHC [32], ESA [48], GE-SA [31], RILS [29], MA [34] and CHSA [49]

Tables 8–11 present the results of these algorithms for 30 instances. The best solution for each instance is highlighted in bold. An empty cell indicates that the algorithm has not tested on this instance. Overall, HH achieved better results for 14 out of 30 tested instances. Although Tables 8–11 suggest that the proposed HH did not find new best solutions for some instances, nevertheless, the differences between the best-known values and HH for these instances are quite small or even equal in some cases.

**Table 10**
Comparison between the proposed HH and the state-of-the-art algorithms — Part III.

| Instance | HH | EM-LAHC | ESA | GE-SA | Best known |
|---|---|---|---|---|---|
| a1_1 | **44,306,501** | **44,306,501** | **44,306,501** | **44,306,501** | **44,306,501** |
| a1_2 | **777,532,177** | 777,533,309 | 777,533,310 | 777,533,310 | **777,532,177** |
| a1_3 | **583,005,717** | 583,005,810 | 583,005,814 | 583,005,813 | **583,005,717** |
| a1_4 | **244,875,200** | 251,015,185 | 251,015,178 | 250,866,958 | 244,875,206 |
| a1_5 | **727,578,308** | 727,578,310 | 727,578,311 | 727,578,314 | 727,578,309 |
| a2_1 | 162 | 166 | 167 | 181 | 161 |
| a2_2 | **720,671,512** | 720,671,543 | 720,671,545 | 720,671,552 | 720,671,548 |
| a2_3 | 1,190,908,040 | 1,192,054,462 | 1,194,261,501 | 1,193,311,446 | **1,190,713,414** |
| a2_4 | **1,680,368,562** | 1,680,587,596 | 1,680,587,592 | 1,680,587,593 | 1,680,368,578 |
| a2_5 | **307,150,823** | 310,287,633 | 310,243,641 | 310,243,857 | 307,150,825 |
| b_1 | **3,291,069,367** | 3,305,899,993 | 3,305,899,957 | 3,307,124,640 | 3,291,069,369 |
| b_2 | 1,015,496,168 | **1,010,949,451** | 1,015,489,174 | 1,015,517,397 | 1,015,496,187 |
| b_3 | **156,691,279** | 156,978,421 | 156,978,415 | 156,978,402 | **156,691,279** |
| b_4 | **4,677,792,538** | 4,677,819,387 | 4,677,819,354 | 4,677,819,137 | 4,677,792,539 |
| b_5 | 922,944,567 | 923,299,306 | 923,299,290 | 923,311,250 | 922,944,697 |
| b_6 | **9,525,851,397** | 9,525,859,949 | 9,525,859,941 | 9,525,857,758 | 9,525,851,483 |
| b_7 | **14,834,456,020** | 14,835,122,152 | 14,835,122,181 | 14,835,031,806 | 14,834,456,201 |
| b_8 | **1,214,291,129** | 1,214,416,691 | 1,214,416,703 | 1,214,416,698 | 1,214,291,143 |
| b_9 | **15,885,437,254** | 15,885,545,683 | 15,885,545,712 | 15,885,548,592 | 15,885,437,256 |
| b_10 | 18,048,187,124 | 18,048,499,611 | 180,512,416,401 | 18,048,499,610 | **18,048,187,105** |
| x_1 | 3,044,417,080 | – | – | – | 3,044,418,078 |
| x_2 | 1,002,390,081 | – | – | – | **1,002,379,317** |
| x_3 | 493,938 | – | – | – | **69,970** |
| x_4 | **4,721,586,143** | – | – | – | 4,721,591,023 |
| x_5 | 521,050 | – | – | – | **54,132** |
| x_6 | 9,546,956,909 | – | – | – | **9,546,936,159** |
| x_7 | **14,252,476,502** | – | – | – | 14,252,476,508 |
| x_8 | 80,107 | – | – | – | **29,193** |
| x_9 | **16,125,531,230** | – | – | – | 16,125,562,162 |
| x_10 | **17,815,981,141** | – | – | – | 17,815,989,054 |

**Table 11**
Comparison between the proposed HH with the state of the art algorithms — Part IV.

| Instance | HH | RILS | MA | CHSA | Best known |
|---|---|---|---|---|---|
| a1_1 | **44,306,501** | – | **44,306,501** | **44,306,501** | **44,306,501** |
| a1_2 | **777,532,177** | – | 777,533,308 | 777,532,181 | **777,532,177** |
| a1_3 | **583,005,717** | – | 583,005,810 | 583,005,717 | **583,005,717** |
| a1_4 | **244,875,200** | – | 250,866,958 | 244,875,201 | 244,875,206 |
| a1_5 | **727,578,308** | – | 727,578,310 | 727,578,309 | 727,578,309 |
| a2_1 | 162 | – | 164 | 162 | 161 |
| a2_2 | **720,671,512** | – | 720,671,537 | 720,671,541 | 720,671,548 |
| a2_3 | 1,190,908,040 | – | 1,193,311,432 | 1,190,908,042 | **1,190,713,414** |
| a2_4 | **1,680,368,562** | – | 1,680,596,746 | 1,680,368,567 | 1,680,368,578 |
| a2_5 | **307,150,823** | – | 312,124,226 | 307,150,825 | 307,150,825 |
| b_1 | **3,291,069,367** | 3,511,150,815 | 3,302,947,648 | 3,291,245,601 | 3,291,069,369 |
| b_2 | 1,015,496,168 | 1,017,134,891 | 1,011,789,473 | 1,015,482,891 | 1,015,496,187 |
| b_3 | **156,691,279** | 161,557,602 | 158,102,214 | 156,757,941 | **156,691,279** |
| b_4 | **4,677,792,538** | 4,677,999,380 | 4,677,819,137 | 4,677,819,387 | 4,677,792,539 |
| b_5 | 922,944,567 | 923,732,659 | 923,311,250 | **922,944,510** | 922,944,697 |
| b_6 | **9,525,851,397** | 9,525,937,918 | 9,525,857,758 | **9,525,851,397** | 9,525,851,483 |
| b_7 | **14,834,456,020** | 14,835,597,627 | 14,836,237,140 | 14,834,456,193 | 14,834,456,201 |
| b_8 | **1,214,291,129** | 1,214,900,909 | 1,214,411,947 | 1,214,291,143 | 1,214,291,143 |
| b_9 | **15,885,437,254** | 15,885,632,605 | 15,885,546,811 | 15,885,437,301 | 15,885,437,256 |
| b_10 | 18,048,187,124 | 18,052,239,907 | 18,051,241,638 | 18,048,271,541 | **18,048,187,105** |
| x_1 | **3,044,417,080** | 3,341,920,446 | – | 3,044,418,078 | 3,044,418,078 |
| x_2 | 1,002,390,081 | 1,008,340,365 | – | 1,002,390,081 | **1,002,379,317** |
| x_3 | 493,938 | 1,359,493 | – | 493,938 | **69,970** |
| x_4 | **4,721,586,143** | 4,721,833,040 | – | 4,721,586,145 | 4,721,591,023 |
| x_5 | 521,050 | 385,150 | – | 521,050 | **54,132** |
| x_6 | 9,546,956,909 | 9,547,002,140 | – | 9,546,956,909 | **9,546,936,159** |
| x_7 | **14,252,476,502** | 14,253,835,332 | – | **14,252,476,502** | 14,252,476,508 |
| x_8 | 80,107 | 96,936 | – | 80,107 | **29,193** |
| x_9 | **16,125,531,230** | 16,125,780,091 | – | 16,125,531,251 | 16,125,562,162 |
| x_10 | **17,815,981,141** | 17,819,116,915 | – | 17,815,981,144 | 17,815,989,054 |

The results are also depicted in Figs. 4–6 for Group a, Group b, and Group x, respectively. From the figures, we can make the following observations:

- For Group a instances (Fig. 4), HH either improved or matched the best-known values for all tested instances, whereas the results of other algorithms are varied across all instances. One can clearly see that the performance of other algorithms is not stable.
- For Group b instances (Fig. 5), HH excels on all instances. However, on a few instances, EM-LAHC obtained the same results as HH. One can also see that few algorithms (GE-SA, VNS, MILS, SA, MA, CHSA) obtained very competitive results compared to those of HH.

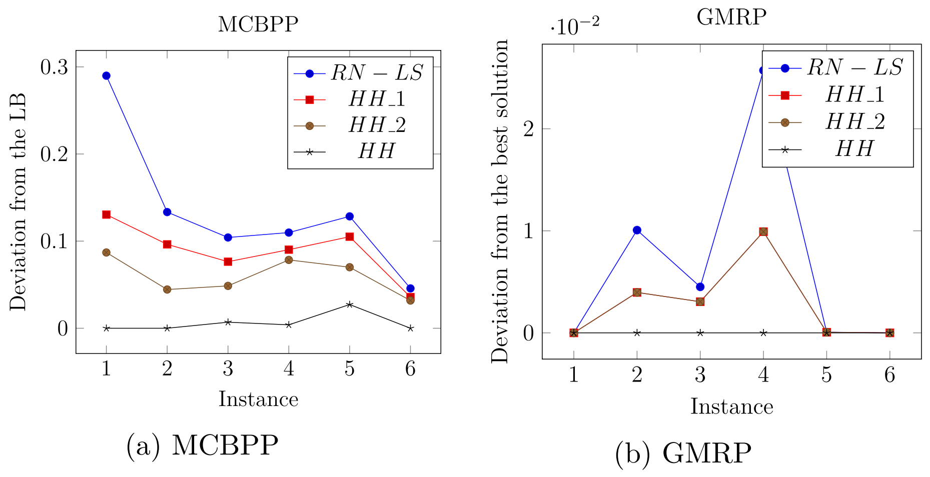

**Fig. 3.** MCBPP and GMRP.

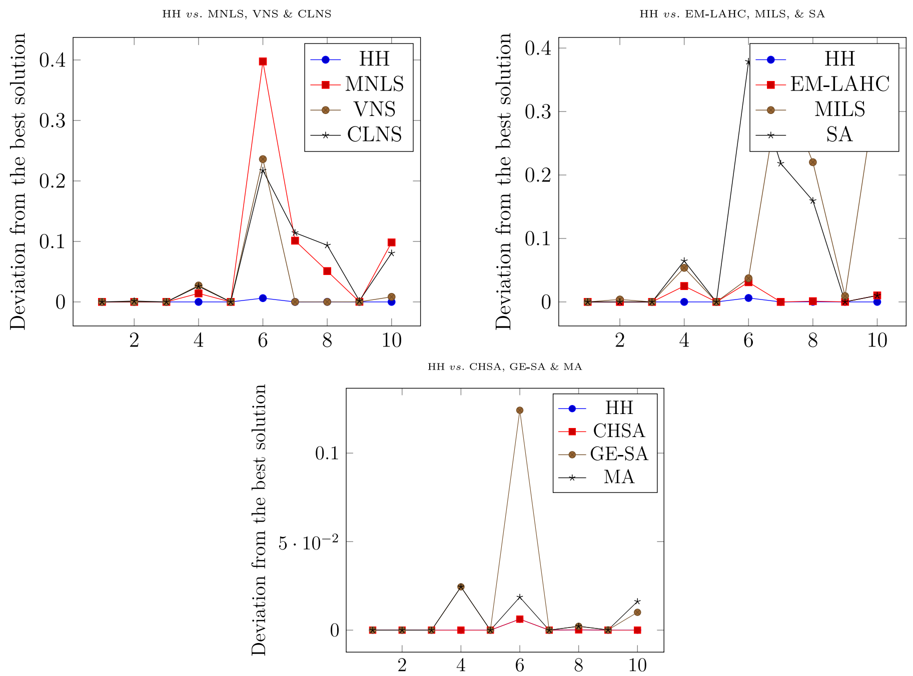

**Fig. 4.** GMRP instance a.

- For Group x instances (Fig. 6), not many algorithms have been tested on Group x instances. Our HH obtained better results when compared to other ones. In many cases, it either obtained the best results or matched the best-known values.

## 5. Conclusions

In this study, we proposed a hyper-heuristic framework for solving combinatorial optimisation problems. It consists of a high-level heuristic component and two low-level heuristic components which are for selecting local search algorithms and operators for the selected algorithm. This framework is a population-based approach meaning multiple solutions participate in the same search process. In addition, we designed a ranking mechanism to improve search. Two complex and well-known combinatorial optimisation problems are involved in the evaluation, namely Multi Capacity Bin Packing (MCBPP) and Google Machine Reassignment problem (GMRP) which is even more challenging than MCBPP. Through our experiments, we can see that when the key components such as ranking or population were removed from the HH framework; the performance will suffer. The comparison between the hyper-heuristic framework with the start-of-the-art algorithms on both MCBPP and GMRP benchmark instances clearly shows the benefit of the proposed HH methodology. Hence, we can conclude that, with the supporting component, the proposed hyper-heuristic can select more suitable local search algorithm and the optimal operator/parameter for the algorithm. This indicates that hyper-heuristic based local search is effective and efficient and can be a stronger candidate for solving complex optimisation problems. The main limitation of this work is the proper interaction between the two stages in which the allocated resource should be adaptively assigned. The work does not stop here as the reason behind this success should be revealed. Details search trajectories will be mapped out to gain insight into how the high level forms low-level heuristics and how the heuristic formulation responds to different search landscape at different stages of the search. Based on these

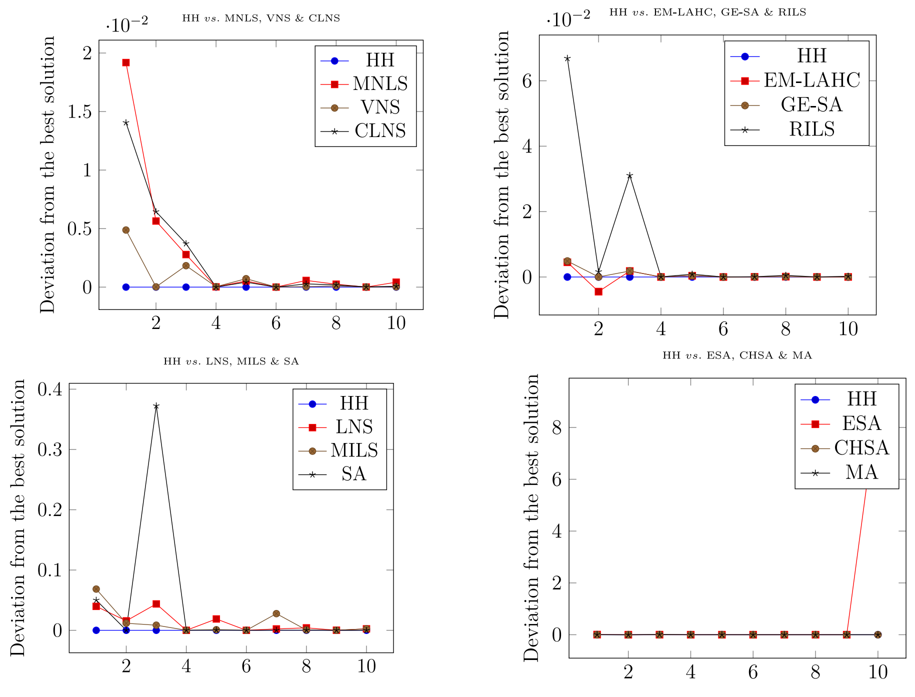

**Fig. 5.** GMRP instance b.

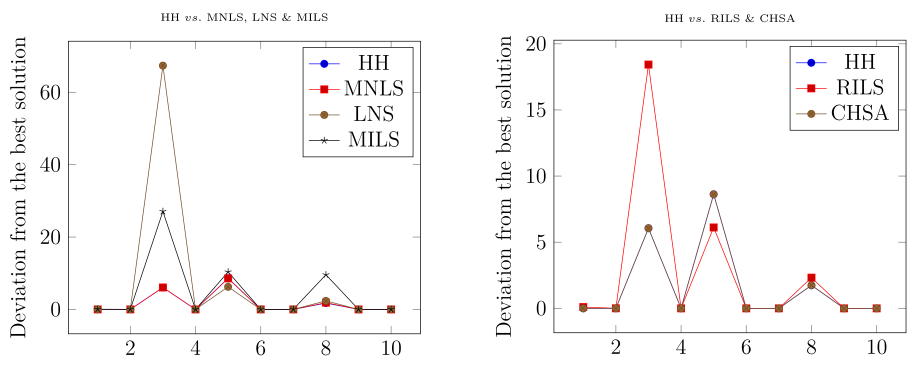

**Fig. 6.** GMRP instance x.

insights, the hyper-heuristic components can be improved to fit for more types of optimisation problems.

## CRediT authorship contribution statement

**Ayad Turky:** Writing - original draft, Writing - review & editing, Software, Visualization, Investigation. **Nasser R. Sabar:** Investigation, Writing - review & editing, Supervision. **Simon Dunstall:** Funding acquisition, Supervision. **Andy Song:** Writing - review & editing, Funding acquisition, Supervision.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgement

This work was supported by Data61 (CSIRO), Australia.

## References

[1] E.-G. Talbi, Metaheuristics: From Design to Implementation, Vol. 74, John Wiley & Sons, 2009.
[2] H.T.T. Binh, P.D. Thanh, T.B. Thang, New approach to solving the clustered shortest-path tree problem based on reducing the search space of evolutionary algorithm, Knowl.-Based Syst. 180 (2019) 12–25.
[3] X. Cai, M. Hu, D. Gong, Y.-n. Guo, Y. Zhang, Z. Fan, Y. Huang, A decomposition-based coevolutionary multiobjective local search for combinatorial multiobjective optimization, Swarm Evol. Comput. (2019).
[4] X. Cai, H. Qiu, L. Gao, C. Jiang, X. Shao, An efficient surrogate-assisted particle swarm optimization algorithm for high-dimensional expensive problems, Knowl.-Based Syst. 184 (2019) 104901.
[5] J.A. Castellanos-Garzón, E. Costa, J.M. Corchado, et al., An evolutionary framework for machine learning applied to medical data, Knowl.-Based Syst. 185 (2019) 104982.
[6] D. Gong, Y. Han, J. Sun, A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems, Knowl.-Based Syst. 148 (2018) 115–130.
[7] Y.-J. Gong, J.-J. Li, Y. Zhou, Y. Li, H.S.-H. Chung, Y.-H. Shi, J. Zhang, Genetic learning particle swarm optimization, IEEE Trans. Cybern. 46 (10) (2016) 2277–2290.
[8] H. Han, W. Lu, J. Qiao, An adaptive multiobjective particle swarm optimization based on multiple adaptive methods, IEEE Trans. Cybern. 47 (9) (2017) 2754–2767.
[9] S. Zeng, R. Jiao, C. Li, X. Li, J.S. Alkasassbeh, A general framework of dynamic constrained multiobjective evolutionary algorithms for constrained optimization, IEEE Trans. Cybern. 47 (9) (2017) 2678–2688.

[10] H. Guan, Z. An, A local adaptive learning system for online portfolio selection, Knowl.-Based Syst. (2019) 104958.
[11] H.R. Lourenço, O.C. Martin, T. Stützle, Iterated local search: Framework and applications, in: Handbook of Metaheuristics, Springer, 2019, pp. 129–168.
[12] N.R. Sabar, A. Bhaskar, E. Chung, A. Turky, A. Song, A self-adaptive evolutionary algorithm for dynamic vehicle routing problems with traffic congestion, Swarm Evol. Comput. 44 (2019) 1018–1027.
[13] Y. Wu, W. Ma, Q. Miao, S. Wang, Multimodal continuous ant colony optimization for multisensor remote sensing image registration with local search, Swarm Evol. Comput. (2017).
[14] X. Xue, J. Chen, Using compact evolutionary tabu search algorithm for matching sensor ontologies, Swarm Evol. Comput. 48 (2019) 25–30.
[15] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.
[16] F.C. Spieksma, A branch-and-bound algorithm for the two-dimensional vector packing problem, Comput. Oper. Res. 21 (1) (1994) 19–25.
[17] M. Guazzone, C. Anglano, R. Aringhieri, M. Sereno, Distributed coalition formation in energy-aware cloud federations: A game-theoretic approach (extended version), 2013, CoRR, vol. abs/1309.2444.
[18] H. Shachnai, T. Tamir, Approximation schemes for generalized two-dimensional vector packing with application to data placement, J. Discrete Algorithms 10 (2012) 35–48.
[19] M. Monaci, P. Toth, A set-covering-based heuristic approach for bin-packing problems, INFORMS J. Comput. 18 (1) (2006) 71–85.
[20] A. Caprara, P. Toth, Lower bounds and algorithms for the 2-dimensional vector packing problem, Discrete Appl. Math. 111 (3) (2001) 231–262.
[21] R. Panigrahy, K. Talwar, L. Uyeda, U. Wieder, Heuristics for vector bin packing, 2011, www.research.microsoft.com.
[22] ROADEF/EURO Challenge 2012: Machine Reassignment. http://challenge.roadef.org/2012/en/.
[23] Z. Wang, Z. Lü, T. Ye, Multi-neighborhood local search optimization for machine reassignment problem, Comput. Oper. Res. 68 (2016) 16–29.
[24] D. Mehta, B. O.S.ullivan, H. Simonis, Comparing solution methods for the machine reassignment problem, in: Principles and Practice of Constraint Programming, Springer, 2012, pp. 782–797.
[25] F. Brandt, J. Speck, M. Völker, Constraint-based large neighborhood search for machine reassignment, Ann. Oper. Res. (2012) 1–29.
[26] Y. Malitsky, D. Mehta, B. OSullivan, H. Simonis, Tuning parameters of large neighborhood search for the machine reassignment problem, in: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer, 2013, pp. 176–192.
[27] H. Gavranović, M. Buljubašić, E. Demirović, Variable neighborhood search for google machine reassignment problem, Electron. Notes Discrete Math. 39 (2012) 209–216.
[28] R. Masson, T. Vidal, J. Michallet, P.H.V. Penna, V. Petrucci, A. Subramanian, H. Dubedout, An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems, Expert Syst. Appl. 40 (13) (2013) 5266–5275.
[29] R. Lopes, V.W. Morais, T.F. Noronha, V.A. Souza, Heuristics and matheuristics for a real-life machine reassignment problem, Int. Trans. Oper. Res. 22 (1) (2015) 77–95.
[30] G.M. Portal, M. Ritt, L.M. Borba, L.S. Buriol, Simulated annealing for the machine reassignment problem, Ann. Oper. Res. (2012) 1–22.
[31] N.R. Sabar, A. Song, Grammatical evolution enhancing simulated annealing for the load balancing problem in cloud computing, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, ACM, 2016, pp. 997–1003.
[32] A. Turky, N.R. Sabar, A. Sattar, A. Song, Parallel late acceptance hill-climbing algorithm for the google machine reassignment problem, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2016, pp. 163–174.
[33] A. Turky, N.R. Sabar, A. Song, Neighbourhood analysis: A case study on google machine reassignment problem, in: Australasian Conference on Artificial Life and Computational Intelligence, Springer, 2017, pp. 228–237.
[34] N.R. Sabar, A. Song, M. Zhang, A variable local search based memetic algorithm for the load balancing problem in cloud computing, in: European Conference on the Applications of Evolutionary Computation, Springer, 2016, pp. 267–282.
[35] H.M. Afsar, C. Artigues, E. Bourreau, S. Kedad-Sidhoum, Machine reassignment problem: the ROADEF/EURO challenge 2012, Ann. Oper. Res. (2016) 1–17.
[36] M.A. Ahandani, M.T.V. Baghmisheh, M.A.B. Zadeh, S. Ghaemi, Hybrid particle swarm optimization transplanted into a hyper-heuristic structure for solving examination timetabling problem, Swarm Evol. Comput. 7 (2012) 21–34.
[37] J. Lin, Z.-J. Wang, X. Li, A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem, Swarm Evol. Comput. 36 (2017) 124–135.
[38] S.A. van der Stockt, A.P. Engelbrecht, Analysis of selection hyper-heuristics for population-based meta-heuristics in real-valued dynamic optimization, Swarm Evol. Comput. 43 (2018) 127–146.
[39] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg, Hyper-heuristics: An emerging direction in modern search technology, in: Handbook of Metaheuristics, Springer, 2003, pp. 457–474.
[40] N.R. Sabar, M. Ayob, G. Kendall, R. Qu, Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems, IEEE Trans. Evol. Comput. 19 (3) (2015) 309–325.
[41] N.R. Sabar, M. Ayob, G. Kendall, R. Qu, A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems, IEEE Trans. Cybern. 45 (2) (2015) 217–228.
[42] N.R. Sabar, A. Turky, A. Song, A. Sattar, Optimising deep belief networks by hyper-heuristic approach, in: Evolutionary Computation (CEC), 2017 IEEE Congress on, IEEE, 2017, pp. 2738–2745.
[43] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, et al., Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[44] H.R. Lourenço, O. Martin, T. Stützle, A beginners introduction to iterated local search, in: Proceedings of MIC, 2001, pp. 1–6.
[45] E.K. Burke, Y. Bykov, A late acceptance strategy in hill-climbing for exam timetabling problems, in: PATAT 2008 Conference, Montreal, Canada, 2008.
[46] G. Dueck, New optimization heuristics: The great deluge algorithm and the record-to-record travel, J. Comput. Phys. 104 (1) (1993) 86–92.
[47] M.R.P. Ritt, An Algorithmic Study of the Machine Reassignment Problem (Ph.D. thesis), UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2012.
[48] A. Turky, N.R. Sabar, A. Song, An evolutionary simulating annealing algorithm for google machine reassignment problem, in: The 20th Asia-Pacific Symposium on INTELLIGENT and EVOLUTIONARY SYSTEMS, in: Proceedings in Adaptation, Learning and Optimization, Springer, 2016.
[49] A. Turky, N.R. Sabar, A. Song, Cooperative evolutionary heterogeneous simulated annealing algorithm for google machine reassignment problem, Genet. Program. Evolvable Mach. (2017) 1–28.